# Supercomputing Frontiers and Innovations

**2020, Vol. 7, No. 1**

## Scope

- Enabling technologies for high performance computing
- Future generation supercomputer architectures
- Extreme-scale concepts beyond conventional practices including exascale
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Distributed operating systems, kernels, supervisors, and virtualization for highly scalable computing
- Scalable runtime systems software
- Methods and means of supercomputer system management, administration, and monitoring
- Mass storage systems, protocols, and allocation
- Energy and power minimization for very large deployed computers
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Parallel performance and correctness debugging
- Scientific visualization for massive data and computing both external and in situ
- Education in high performance computing and computational science

## Editorial Board

# Contents

**State of the Art and Future Trends in Data Reduction for High-Performance Computing**
K. Duwe, J. Lüttgau, G. Mania, J. Squar, A. Fuchs, M. Kuhn, E. Betke, T. Ludwig ..... 4

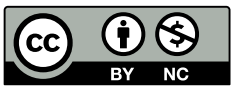
**Development of Computational Pipeline Software for Genome/Exome Analysis on the K computer**
K. Aoyama, M. Kakuta, Y. Matsuzaki, T. Ishida, M. Ohue, Y. Akiyama ..... 37

**Supercomputing Technologies as Drive for Development of Enterprise Information Systems and Digital Economy**
O.V. Loginovsky, A.L. Shestakov, A.A. Shinkarev ..... 55

**Online MPI Process Mapping for Coordinating Locality and Memory Congestion on NUMA Systems**
M. Agung, M.A. Amrizal, R. Egawa, H. Takizawa ..... 71

**Tools for GPU Computing – Debugging and Performance Analysis of Heterogenous HPC Applications**
M. Knobloch, B. Mohr ..... 91

**Building a Vision for Reproducibility in the Cyberinfrastructure Ecosystem: Leveraging Community Efforts**
D. Chapp, V. Stodden, M. Taufer ..... 112



DOI: 10.14529/jsfi200101

# State of the Art and Future Trends in Data Reduction for High-Performance Computing

***Kira Duwe***[1]***, Jakob Lüttgau***[1]***, Georgiana Mania***[2]***, Jannek Squar***[1]***,***
***Anna Fuchs***[1]***, Michael Kuhn***[1]***, Eugen Betke***[3]***, Thomas Ludwig***[3]



Research into data reduction techniques has gained popularity in recent years as storage capacity and performance become a growing concern. This survey paper provides an overview of leveraging points found in high-performance computing (HPC) systems and suitable mechanisms to reduce data volumes. We present the underlying theories and their application throughout the HPC stack and also discuss related hardware acceleration and reduction approaches. After introducing relevant use-cases, an overview of modern lossless and lossy compression algorithms and their respective usage at the application and file system layer is given. In anticipation of their increasing relevance for adaptive and in situ approaches, dimensionality reduction techniques are summarized with a focus on non-linear feature extraction. Adaptive approaches and in situ compression algorithms and frameworks follow. The key stages and new opportunities to deduplication are covered next. An unconventional but promising method is recomputation, which is proposed at last. We conclude the survey with an outlook on future developments.

*Keywords: data reduction, lossless compression, lossy compression, dimensionality reduction, adaptive approaches, deduplication, in situ, recomputation, scientific data set.*

## Introduction

Breakthroughs in science are increasingly enabled by supercomputers and large scale data collection operations. Applications span the spectrum of scientific domains from fluid-dynamics in climate simulations and engineering, to particle simulations in astrophysics, quantum mechanics and molecular dynamics, to high-throughput computing in biology for genome sequencing and protein-folding. More recently, machine learning augments the capabilities of researchers to sift through large amounts of data to find hidden patterns but also to filter unwanted information.

Unfortunately, the development of technologies for storage and network throughput and capacity does not match data generation capabilities, ultimately making I/O a now widely anticipated bottleneck. This is only in part a technical problem, as technologies to achieve arbitrary aggregate throughput performance and capacity exist but cannot be deployed economically. Besides being too expensive at the time, their energy consumption poses a challenge in exascale systems adding to the operational cost [51]. As data centers are expected to become a major contributor to global energy consumption [133], data reduction techniques are an important building block for efficient data management.

Also, to capture as much data as possible as efficiently as possible, they are going to become far more important in the future. Especially as multiple projections across scientific domains estimate increasing data volumes for a variety of reasons: For one, Data generation capabilities are on the rise, due to improving compute capabilities as supercomputers become more powerful but also more broadly available. The increase in CPU performance encourages researchers to increase model resolution, but also to consider more simulations for uncertainty quantification,

[1]Hamburg University, Hamburg, Germany
[2]Deutsches Elektronen-Synchrotron (DESY), Hamburg, Germany
[3]German Climate Computing Center (DKRZ), Hamburg, Germany

both increasing the amount of generated data. Higher-resolution instrument and detector outputs, in addition to an exploding number of small scale measurement stations, are a second factor leading to increased data ingest. This shows in particular in remote sensing for earth observation and astronomy as well in detectors used in high-energy physics.

In many scientific contexts, data is stored in self-describing data formats such as NetCDF and HDF5. Some subdomains like bioinformatics may also employ text-based (blast, fasta) formats, that can be handled similarly with respect to compression. But to cope with the magnitude of the data, established techniques such as compression on written data are not sufficient. One alternative growing in popularity are in situ approaches, which bypass the storage systems for many post-processing tasks. Often in situ lossy compression is applied as data is generated, but more advanced in situ analysis relies on triggers preserving only records for essential events. In some cases, data volumes are reduced by several orders of magnitude [60]. CERN, for example, employs both combinatorial in situ and off situ techniques to filter and to track the particles produced in the proton-proton collision [15]. Selecting only the interesting events worthy of offline analysis means keeping 1/200000 events, which occur every second [60]. In situ processing can also benefit data visualization. At the German Climate Computing Center (DKRZ), use cases for in situ techniques include data reduction as well as feature detection, extraction, and tracking, which are also used to steer simulation runs [123]. Uptake in situ methods makes image data formats more relevant. Thus, our survey also covers data reduction for images.

Besides in place reduction at the application layer, optimization and data reduction opportunities can be found along the data path. A typical HPC stack spans multiple I/O layers including parallel distributed files systems, the network, and node-local storage. Additional leveraging points can be found in the memory hierarchy, spanning caches, RAM, NVRAM, NVMe down to HDDs and tape for long-term storage.

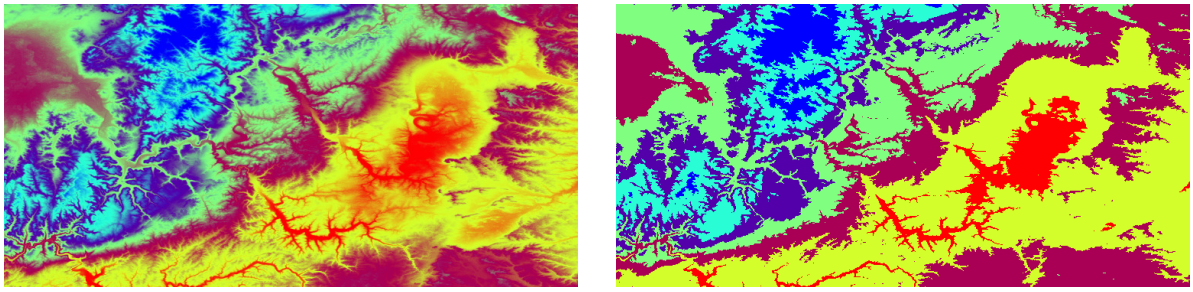
This survey covers a large variety of data reduction techniques. Mathematical backgrounds are, therefore, not covered in-depth. Additional details about fundamental compression techniques are discussed by Li et al. [82]. We extend their work by presenting additional frameworks, tools, and algorithms for compression. Some of them emerged in the last two years. In addition, we evaluate a more extensive variety of data reduction techniques considering dimension reduction, adaptive approaches, deduplication, in situ analysis, and recomputation.

The remainder of this paper is structured as follows: In section 1, an overview of lossless as well as lossy compression algorithms is given. Section 2 introduces dimensionality reduction techniques, forming the basis for the adaptive approaches described in section 3. In section 4, deduplication is explained. Section 5 details the algorithms and frameworks for in situ analysis. Recomputation approaches are collected in section 6. An outlook on future developments is provided in section 7.

## 1. Compression

Compression reduces the size of a dataset by removing redundancies to maximize its entropy. The ratio between the original and compressed data is denoted as the compression ratio (CR), where a higher ratio is better. Lossless compression is used whenever the reconstruction has to be byte-exact and has been widely explored in scientific domains. In general, the decompression is faster than the compression, which often fits the typical HPC workflow: Data is calculated and compressed once but read and decompressed several times [93]. Compression can be integrated into different system layers, such as the application layer or the file system and block layer. While

the user can decide whether to deal with lossy or lossless compression at the application layer, transparent compression within the system out of the user's control has to be always lossless. Data like source codes or binaries must be compressed only lossless since their reconstruction must be accurate.



(a) Before truncation (32-bit floating-point) (b) After truncation (8-bit integer)

**Figure 1.** Truncation reduces data size (conversion from 32-bit floats into 8-bit integers results in a factor of 4) but also leads to a visible degredation of data quality

Lossless compression usually provides a poor CR for HPC applications due to their extensive usage of floating-point data [77]. Relief can be achieved by using lossy compression since it allows to trade in data quality for data size. Many techniques for lossy compression have been developed with a focus on multimedia data such as audio or image data, but they can be reused for scientific data too [82]. However, losing data quality is not acceptable for all users and limits possible use cases. Figure 1 gives an example of a crude quantization: The original 32-bit floating-point data is mapped to 8-bit integers. The file size reduction is about a factor of 4 and comes at the price of a visible degradation of the data quality.

### 1.1. Lossless Compression

Lossless compression at the storage end is one of the obvious use cases in HPC systems, which aims for more storage capacity. While some of the local file systems like btrfs [83], NTFS or ZFS [154] provide compression services, the distributed parallel file systems used in the HPC field are often limited concerning this service. Ceph and GlusterFS both run in user space and offer server-side compression. Spectrum Scale (previously GPFS) by IBM also offers compression of cold data with zlib and recently introduced LZ4 for sequential read workload [13]. The most popular HPC file system Lustre indirectly benefits from compression when using the ZFS backend, which supports multiple compression algorithms. The data is then kept compressed not only on disk, which saves storage space but also in ZFS's own cache, which allows more diskless I/O. Lately, ZFS also gained support for hardware compression [62].

Compression can also improve the relative network throughput, I/O completion time and therefore speed up application runtimes. IOFSL is a project of Argonne National Laboratory aiming to provide a software layer at the file system interface. It has been extended by compression services, which can increase network bandwidth [152]. Another improvement for network throughput has been achieved by implementing an optimization of the two-phase collective I/O technique for ROMIO, the most popular MPI-IO implementation [45]. By reducing the data from group sources, Mellanox switches allow the collective operation processing to be offloaded and therefore improve the network speed in-flight.

Furthermore, hardware compression is used for better CPU utilization [21, 29]. Also, the energy costs of running an HPC cluster can be reduced by compression [10, 72].

Another use case apart from I/O and persistent data is transparent compression of working data within the application. zram is a kernel module that creates compressed block devices and compresses the process memory without its knowledge [99]. It is commonly used for temporary files or as a swap disk, while live data stays uncompressed. The CR is limited since only single pages are compressed. As the benefit of transparent compression depends on the specific data, it is not a universal approach. Additional studies have been done to reduce the performance gap between processor and memory calls by introducing cache compression [9, 33]. A wide range of scientific applications and tools already involves compression mechanisms for computation or out-processing, like LAMMPS, HDF5 and MapReduce [40, 120].

There are different potential usages of lossless compression with different requirements on the algorithms. The following sections describe the basics of entropy- and dictionary-based compression and mention the most popular and useful lossless algorithms for the HPC field.

### *1.1.1. Entropy-based*

Entropy-based encoding works by replacing unique symbols within the input data with a unique and shorter prefix code. The more often a symbol occurs, the shorter the prefix should be to ensure the best CR [94]. The most known techniques are arithmetic and Huffman codings. Huffman coding replaces words in a way that no word is a prefix of any other word in the system [111]. It creates a binary tree of nodes, which represents the symbols and their frequencies. At first, the data has to be scanned and the frequencies must be calculated. The nodes are either inserted into the binary tree or combined depending on the frequencies [20]. While Huffman coding splits input data into components that are encoded separately, arithmetic coding encodes the entire input into a number. This coding aims to compress close to the entropy limit while Huffman coding performs badly when the probabilities do not equal fractions with powers of two in their denominators [25]. Both codings are rarely used as stand-alone algorithms but can be adapted and combined with more elaborated techniques, some of which follow.

### *1.1.2. Dictionary-based*

Another popular method is based on dictionaries and aims for partitioning the original input data to phrases (non-overlapping subsets of the original data) and the corresponding, possibly shortest codewords. This encoding is also known as substitution and has two main stages: dictionary construction (finding phrases and codewords) and parsing (replacing phrases by codewords) [125]. The dictionary has to be available for both the compressor and the decompressor. Dictionary codes can be classified into static and dynamic (or sometimes adaptive) constructs. Static dictionaries are created before the input processing and stay the same for the complete run. In case of the dynamic method, the dictionary is updated during parsing and the two stages (construction and parsing) mostly interleave. Byte pair encoding is a simple way of compression where the most common symbols are replaced by a symbol, different from the original alphabet [134]. The table of replacements is called *dictionary*.

**LZ Family** Lempel-Ziv is one of the most known dynamic dictionary compression methods. *LZ77* assumes and exploits that data is likely to be repeated. When repeated, a word can be

replaced by a pointer to the last occurrence accompanied by the number of matched characters [132]. The dictionary is then a part of the previously encoded sequences. The input is analyzed through a sliding window, that consists of search and look ahead buffers. LZ77 is suffix-complete, which means any suffix of a phrase is a phrase itself. The performance is limited by the number of comparisons needed for finding the matching pattern.

LZ77's successor *LZ78* constructs the dictionary differently. It begins with a single symbol entry in the dictionary, which grows by concatenating the first symbol of the following input after every parsing step. This algorithm uses greedy parsing, replacing the longest phrase with a prefix match by a codeword. In opposite to LZ77, LZ78 is prefix-complete.

Due to the explicit dictionary, the patterns are potentially held until the end of the input, which results in ever-growing dictionary buffers. There are several approaches on how to limit their sizes or optimize the dictionary building. These modifications of the general method and the development of optimizations continue today. The development of variants to the original method and respective optimizations continue today. All the modifications are very fast at decompressing, just like their ancestors.

**LZ77 Variants** *LZSS* is an algorithm developed by Storer and Szymanski that improves the look-ahead buffer by storing it in a circular queue and introduces a binary tree for the search buffer. *LZS* is based on LZSS and uses Huffman encoding for the length-distance pair [126]. *DEFLATE* is based on LZSS but uses a chained hash table to find duplicated sequences. The matched lengths and distances are compressed with two Huffman trees. There are hardware implementations of a novel adaptive version of DEFLATE [141] and an FPGA approach [48]. The DEFLATE format is used in ZIP, gzip, Zopfli, zlib and many other algorithms, which also have hardware implementations [1, 116]. *LZJB* is based on *LZRW1*, which uses hash tables among other techniques. While LZRW1 could overrun buffers by either reading past the input or writing past the output, LZJB does not. As a result, it is more suitable for file system usage, e.g., in ZFS [155]. It also allows a larger match length with smaller look-behind buffer and is, therefore, faster with less memory usage. *LZMA* is the default algorithm used in the 7z compression software and is similar to DEFLATE but uses delta filtering with range, instead of Huffman encoding [81]. MAFISC is an HDF5 compression filter based on LZMA. *LZX* uses a history buffer up to 2 MiB and combines Huffman coding techniques with shorter codes. The three most recent matches are then compressed with LZMA [100].

*Snappy* is developed by Google and was open-sourced in 2011. The input is cut into fix sized (64 KiB) blocks and the encoder is byte-oriented. Work on an FPGA version of Snappy has been done [118]. *LZFSE* is an Apple compressor with finite-state entropy, which combines a dictionary compression scheme with a technique based on asymmetric numeral systems [137]. *Brotli* is the Google approach to replace Zopfli and is not DEFLATE-compatible [8]. The compressed file is represented by a collection of meta-blocks. These are composed of a data part, which is simply compressed by LZ77 and a header describing how to decode the data part.

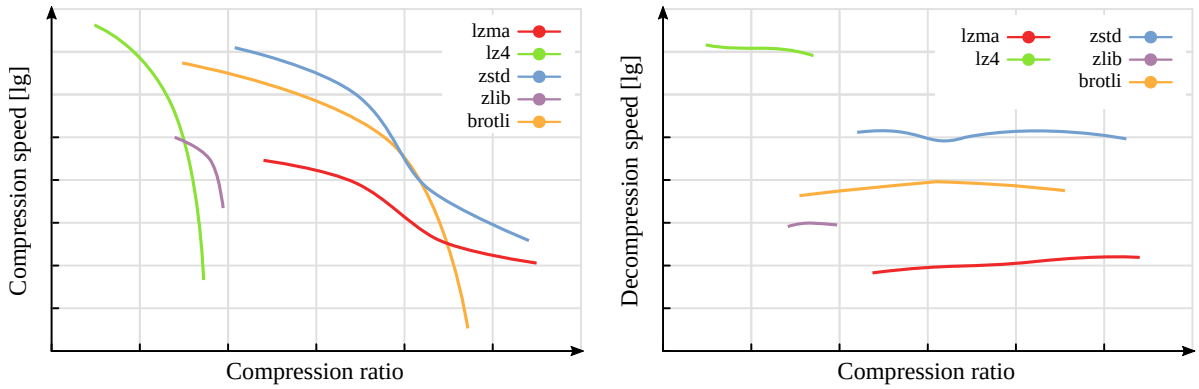
*Bloclz* is the default algorithm in Blosc, a high-performance compressor optimized for binary data. It is based on FastLZ, which itself is inspired by *LZV* and *LZF* algorithms. They all favor CPU efficiency over CR. *LZO* uses a quick hash table for lookups and has additional optimizations to output tokens [160]. The multi-core performance is explored in an additional work [69]. *LZ4* is another LZ77 variant with a fixed, byte-oriented encoding with no other codings. It provides an extremely fast decompressor, which can reach up to half of the memcpy throughput.

LZ4 has further variants. One is the fast mode that trades CR for compression speed and the high compression (HC) mode. There are also modifications for real-time hardware as well as general hardware implementations [80, 90]. *Lizard*, previously called LZ5, uses expansions to LZ4 and can optimally combine them with Huffman coding. It is also a part of the Blosc compressor library. *Zstd* supports a large search window (eight times larger than zlib) and involves an entropy coding stage, using fast Finite State Entropy or Huffman coding. The implementation works well with modern processors and compilers and is multi-threaded [131, 140].

**LZ78 Variants** One of the modifications is *LZW*, introduced by the initial authors of LZ78 and Terry Welch [151]. A dictionary is initialized to all possible symbols and the input is then processed symbol by symbol and concatenated to a string that is searched in the dictionary.

This process continues as long as matches are found and the dictionary is updated to the new concatenation when the word was missing in the dictionary [130]. The GIF encoding is based on LZW. There are also hardware-accelerated versions of LZW [127]. *LZWL* is an LZW extension for working with syllables or complete words. Other variants of this encoding are *LZMW*, *LZAP* and *LZWL*. LZMW does not reinitialize the full dictionary like LZW does, but removes the last used phrase instead. Here, concatenation is not performed on one new symbol and the match but on one match with another match [76]. LZAP is another modification of LZW, which adds all the prefixes of the unknown word instead of a concatenation of one prefix with this word. It allows for better CR with potentially faster dictionary growth and more frequent updates [142].

Other compression algorithms do not use dictionaries but smartly combine several techniques, especially in order to achieve a high CR. One of such is *bzip2*, which compresses files using the Burrows-Wheeler block-sorting compression algorithm [17], MTF (move to front), RLE (run-length encoding) of MTF result, Huffman coding, Unary base-1 encoding of Huffman table selection, Delta encoding of Huffman-code bit lengths and sparse bit arrays. Due to its low performance, parallelization of bzip2 has been explored in [53].




(a) (b)

**Figure 2.** Qualitative comparison of some compression algorithms regarding the de-/compression speed and ratio (the larger, the better) for different compression levels [140]

Intel's *QAT* (QuickAssist Technology) provides hardware-based algorithms for cryptography and compression. De-/compression is offloaded to the QAT module (available on chipset or external PCIe cards). QAT can speed up different algorithms like DEFLATE, LZS and can be extended for many other algorithms. The data has to be physically contiguous, which is a hard requirement exceedingly few systems are able to fulfill without additional efforts. Hardware

compression has the potential to be much faster, but due to limited buffers (mostly one kernel page of 4 KiB), software solutions may still preferable [62].

#### *1.1.3. Efficiency*

Depending on the needs within the HPC systems, the most suitable algorithm varies. While the application and network layer mostly require very fast algorithms, the storage backend can afford slower throughput for a higher CR but greatly benefits from a fast decompressor for the read performance. Figure 2 shows a few of the algorithms measured on a specific data set with different compression levels. The results are extremely dependent on the input data but often allow a qualitative insight into the performance nevertheless.

### 1.2. Lossy Compression

Data in HPC is mostly generated by numerical simulations of natural processes, which follow the principle of locality so that adjacent data is likely to be highly correlated. Floating-point data is hard to compress nonetheless since it often already features high entropy.

Applying lossless compression on large data sets does not always satisfy external requirements such as guaranteed I/O performance for live visualization or limited assigned storage, due to long compression times or low CRs. If a controlled loss of data quality is an option, lossy compression can be applied instead. Using lossy compression, CRs of over 400:1 are possible [41] – even though a CR beyond 64:1 will degrade precision of reconstructed data [84]. Lossy compression does not always outperform lossless compression in case of constrained error margins [93]. Lossy compression is common in computer graphics [7, 16] and used by many visualizations primarily meant to be interpreted by humans but are not fed back to numerical computations.

Unbiased compression methods which stay within the data's noise or analysis's error margin can be applied without degradating quality [40, 73]. More use cases for lossy compression include the reduction of I/O time or the acceleration of checkpoint handling [30]. Lossy compression methods are usually used in a multilayer-compression approach, though specific algorithms reduce the data size sufficiently on their own: First, lossy compression reduces the data diversity so that the lossless compressors applied afterwards work more efficiently. There are different approaches to combining these techniques as well as distinctions in their implementation.

Many file formats add native support for lossy compression like for example NetCDF4/HDF5 [40], GRIB2, XTC/TNG [95], and Zarr.

#### *1.2.1. Methods*

**Preprocessing** These methods are easy to apply but lag behind regarding the CR or quality of the reconstructed data. A naive lossy compression step is *truncatation*, which simply omits the least significant bits. In the case of floating-points, these are the last bits of the mantissa.

More subtle approaches are *Bit Shaving* and *Bit Grooming*, which do not change the data type itself but tamper with the bit representation of floating-point data. Similar to truncation, Bit Shaving modifies the most insignificant bits by changing them to zero. It thereby induces a bias as the new values always underestimate the original values. To correct this bias Bit Grooming applies a bitmask, in which zeros and ones are alternating to balance the error [159]. For a given number of significant digits of a float to be preserved, Bit Grooming tends to spare too

many bits from being changed. A potential remedy for this is *Digit Rounding*, which substitutes the static bitmask of Bit Grooming with a dynamic and more granular bitmask [40].

Normalization can be used to save bits within floats, as values close to zero require fewer bits in the mantissa while preserving range and precision.

Another method is *quantization*, where the values of the original data are first subdivided into intervals and each point of an interval is then mapped onto the same representative. In literature, the representatives are called *codewords* and the interval scheme *codebook*. The mapping can be done fore single values (*scalar quantization*) or a set of values (*vector quantization*). Special attention should be paid to how the codebook is constructed because it is more computational intensive compared to the lookup.

A collection of algorithms to generate codebooks is presented in [63]. If the codebook generates intervals of varying length, the error bound cannot be guaranteed; to control the error the intervals' length must be uniform [143].

*Linear packing* uses quantization by mapping normalized 4-Byte floats onto 2-Byte integers. Preserving the original dynamic range, a linear transformation between the original and modified data is stored in addition. While this introduces overhead, the data size is still about halved in return. This approach can be further improved by *Layer-packing*, which applies linear packing on slices of multi-dimensional data. The slicing is performed alongside the so-called thick dimension, i.e. the dimension with the highest range of values. For example, many variables of a 3D atmospheric model undergo heavy changes along the vertical dimension but are comparatively quite stagnant at the same height. In this case, the dataset would be split into horizontal slices. The precision of the linear packed slices is improved at the cost of additional overhead [136].

**Transform compression** This kind of method relies on a frequency transformation of the spatio-temporal original data signal into a new domain. The coefficients of the transformed signal may then be classified regarding their significance. While no loss of precision is applied in theory because transformations are invertible, transformations on floating-point data typically lead to rounding errors. In literature those transformations are therefore also addressed as *near-lossless* methods [82]. Transform compression becomes definitely lossy as soon as insignificant small coefficients are eliminated. The most important transformation methods are variants of discrete Fourier transforms (fast Fourier transform [FFT], discrete cosine transform [DCT], modified discrete cosine transform [MDCT], discrete sine transform [DST], modified discrete sine transform [MDST]) and wavelet transforms. An extensive overview of these transformation methods can be found in [149].

**Prediction** A good derivation of a data predictor allows estimating the value of adjacent data. To reproduce the original data, the predictor and the differences, also called residuals, need to be stored, which are minimal if the predictor has been well fit. Those residuals feature less entropy and, if they are small enough, they can be represented by smaller datatypes [114]. If the residuals are preprocessed before their compression, this method becomes also irreversible.

Some relevant schemes to perform the prediction are linear predictors (e.g. *Lorenzo predictor* [64], *mean-integrated Lorenzo predictor* [84]), *differential pulse-code modulation* [32], and *motion compensation* [139]. The latter one is of special interest for molecular dynamics since the simulated particles move along trajectories or stand still and can, therefore, be approximated by for example a low-polynomial function [114].

### *1.2.2. Selection of lossy compressors*

Even though lossy compression is not omnipresent in HPC yet, there are some compressors available. Well-known are SZ and ZFP since they achieve a good performance [41], but we will also discuss other compressors, which stand out in their field. The last four entries are frameworks, which provide a collection of lossy compressors.

*SZ [41]:* Three steps are performed on the original data. At first, a multidimensional prediction model is applied, such as preceding neighbor fitting that assumes the current value to equal the preceding value. There are also linear and quadratic polynomial fittings. Afterwards, prediction points are quantized. Data, which cannot be fitted by the quantized predictors, is normalized and then truncated. At last, further compression is performed by using DEFLATE.

*ZFP [86]:* The original 3D data is chunked into $4 \times 4 \times 4$ blocks. All values in a block are normalized to the maximum in this block so that the adjusted values are in the range of $[-1, +1]$. Since high-order floats are eliminated now, the values are transformed into a Q3.60 fixed-point representation, which increases the numerical stability. Then, an orthogonal transform is applied and the resulting coefficients are sorted. Because the transformation results in many insignificant coefficients, they are well compressible.

*FPZIP [87]:* The data is at first approximated via a Lorenzo predictor and then truncated. Intervals of predictions and residuals are then encoded. Most of the time, ZFP outperforms FPZIP when the lossy compression is used [143].

*ISABELA [77]:* To smooth the data, it is first sorted and predicted afterwards using B-splines. Since ISABELA does in situ processing, it is discussed in detail in section 5.1.2.

*TTHRESH [19]:* The Tucker decomposition, a higher-order singular value decomposition, is performed. Then the decomposition core is flattened and the coefficients are compressed. Ballester-Ripoll et al. showed that the degradation of data quality is managed well at high CRs since high peak signal-to-noise ratios are preserved. However, their compression scheme suffers from lower (de-)compression speed and disadvantageous random access times in return. Another drawback is the insufficient support for 2D datasets [30].

*NUMARCK [35]:* Motion compensation (forward predictive coding) between checkpoints is used and data is then approximated by utilizing machine learning. NUMARCK is therefore discussed in detail in section 3.1.2.

*SSEM [128]:* This compressor uses a wavelet transform and vector quantization to speedup the general checkpoint creation time of an HPC application.

*FRaZ [148]:* This framework includes SZ, ZFP and MGARD [3–5]. The lossy compressors can be controlled by setting the upper bound of the absolute error. Some already feature a fixed-rate mode, but usually, this mode comes with a catch, e.g. it neglects the absolute error bound. FRaZ determines the correct setting of the absolute error for an indicated combination of lossy compressor and a particular dataset with only limited overhead.

*VAPOR [113]:* VAPOR allows to explore and interact with large datasets stepwise. To make this possible, it utilizes a progressive data model to divide the dataset into areas of interest and apply lossy compression (wavelet transforms). Currently, VAPOR3 is being developed and according to its roadmap, additional compressors shall be included [150].

*Z-checker [144]:* The primary goal of this framework is to allow users checking and understanding the data features and how lossy compression would affect the data quality. The virtualization allows to quickly explore datasets and evaluate the impact of several lossy compressors on specific data and compression properties.

*Deep neural network [119]:* This approach uses a neural network to analyze dataset features such as the hit ratio of prediction methods in order to estimate the CR for different lossy compressors, currently SZ and ZFP. More details are presented in section 3.2.

# 2. Dimensionality Reduction

Scientific data such as simulation or detector output usually has a high dimensionality, often requiring reduction for appropriate handling. Ideally, the result of dimensionality or dimension reduction (DR) resembles the intrinsic dimensionality of the data, thus preserving those features necessary for characterization [49]. Besides mitigating the curse of dimensionality [23], DR reduces the multi-collinearity, thereby improving the interpretation of parameters while decreasing the computation time as well as the data size. Also, visualizing results is considerably simplified with a smaller feature space. DR approaches can be separated into feature selection and feature extraction (FE). The former focuses on selecting a characteristic subset while feature extraction, also referred to as feature projection, transforms the data into a representation of fewer dimensions [31]. As neither the geometry of the data nor the intrinsic dimensionality is known, DR is an ill-posed problem enforcing the assumption of certain data properties [98].

Following the taxonomy of DR techniques depicted in Fig. 3 by Maaten et al., FE approaches are split into convex and non-convex techniques. The main difference being that non-convex techniques can model the existence of multiple local optima, while convex approaches require global and local optimum to coincide.

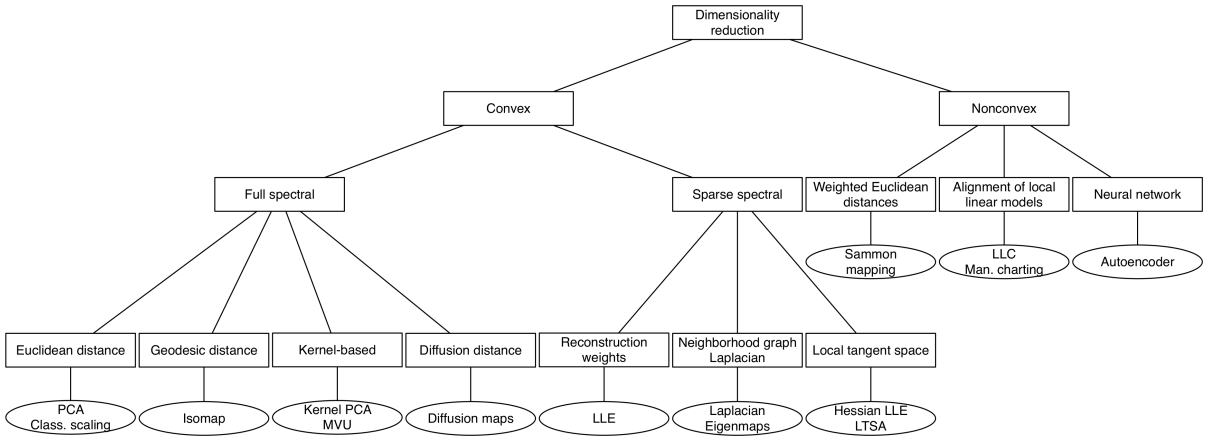



**Figure 3.** Taxonomy of dimensionality reduction techniques (FE) from [98]

## 2.1. Feature Selection

Feature selection approaches devide into wrappers, filters, and embedded methods [57]. Prominent wrappers are greedy search strategies, namely *backward elimination* and *forward selection*. The first removes the least promising variables while the latter continuously incorporates variables into larger subsets. Filter methods often focus on features like variance or correlation as they are easy to compute [57]. In contrast to wrappers, they are not adjusted to a specific model and result in a more general selected set. Random forests have proven to be useful for ranking feature importance [37]. Cliff et al. proposed an iterative random forest implementation optimized for HPC. An embedded feature selector based on the linear dependency of input and output is *Least absolute shrinkage and selection operator (Lasso)*. Yamada

et al. proposed *Hilbert-Schmidt Independence Criterion Lasso (HSIC Lasso)* considering also non-linear dependencies [158]. The global optimum of HSIC Lasso can be efficiently determined, making it a scalable technique suitable for very high-dimensional data where the dimensionality is magnitudes higher than the sample size.

## 2.2. Linear Feature Extraction

*Principal Component Analysis (PCA)*, also called classical scaling, finds a linear low-dimensional projection maximizing the data variance, that is finding a low number of representative linear combinations (principal components) [39]. PCA can either be performed by using the covariance matrix to construct the eigenvectors or through *singular value decomposition (SVD)* of a normalized data matrix. The computational complexity of PCA, determined by the number of datapoints $n$ and their dimensionality $D$, is $\mathcal{O}(D^3)$, when $D < n$. Therefore, there is a number of proposed optimizations. One is to use the autocorrelation and the large scale structure of data produced by climate science or a similar domain where the values do not vary abruptly over time and neighboring fields are not completely independent [24]. Martel et al. show how the iterative Jacobi method can be used to speed up the eigenvalue decomposition of PCA on HPC systems using hardware acceleration [102].

Several approaches are based on PCA and SVD, such as factor analysis [98], Partial Least Squares (PLS) or Maximum Covariance Analysis (MCA). PLS and MCA proofed valuable for the analysis of multi-temporal datasets [24]. The *Linear Discriminant Analysis (LDA)* is closely related to PCA but focuses on the discrimination between classes in contrast to PCA that does not consider any underlying class structure of the data for the computation of the principal components [103]. Opposed to LDA, the *Generalized discriminant analysis (GDA)* is a nonlinear technique using kernel function operators [22]. *Random Projection (RP)* is based on the Johnson-Lindenstrauss lemma stating it is possible to map vectors of high-dimensional space onto an $\mathcal{O}(n \log n)$ dimensional space while the pairwise distances are approximately preserved [157]. While RP can be computed efficiently, high distortions are possible. An interesting proposal to counter this is to first increase the dimensionality to have a better feature representation and then to reduce it with RP [96]. *Non-negative Matrix Factorization (NMF)* is an approach, often used in domains as astronomy, that has good interpretability due to the non-negative entries in the factorized matrices [31].

## 2.3. Non-Linear Convex Feature Extraction

*Kernel PCA* operates in a high-dimensional space constructed through a kernel function [98]. Therefore, the eigenvectors are based on the kernel matrix, not the covariance matrix. *Isomap* improves classical scaling by using the geodesic distance instead of the Euclidean distance, thereby considering the distribution of neighboring data points on a manifold [146]. When data points lie on or near a curved manifold (Swiss roll dataset), the Euclidean distance may differ considerably from their distance over the manifold. However, Isomap is topologically instable, possibly constructing incorrect connections in the neighborhood graph [98]. Also, non-convex manifolds can pose problems. *Maximum Variance Unfolding (MVU)* expands on Kernel PCA by aiming at learning the kernel matrix through defining a neighborhood graph. MVU differs from Isomap because it preserves the local geometry, ultimately trying to unfold the manifold. *Diffusion Maps (DM)* use Markov random walks on the data graph to determine the diffusion

distances in that it is more likely to walk to a point nearby [98]. As it integrates over all paths, the diffusion distance is more robust to noise or short-circuiting than the geodesic distance. The computational complexity is $\mathcal{O}(n^3)$ for Kernel PCA, Isomap and DM, and $\mathcal{O}((nk)^3)$ for MVU with $k$ nearest neighbors. *Local Linear Embedding (LLE)* and *Laplacian Eigenmaps (LE)* are closely related to Kernel PCA and Isomap as they all solve an eigenproblem. Their crucial difference lies in the type and scope of local property preservation. LE differs from *Hessian LLE* only in the way the differential operator on the manifold is defined. These sparse spectral methods can be computed in $\mathcal{O}(pn^2)$, where $p$ is the ratio of nonzero elements to the total number of elements.

### 2.4. Non-Linear Non-Convex Feature Extraction

*Sammon Mapping (SM)* improves classical scaling by weighting each pair's input to the cost function so that the local structure is retained better. It has been successfully applied to geospatial data [98]. *Locally Linear Coordination (LLC)* and *Manifold Charting (MC)* globally align local linear models. *T-distributed Stochastic Neighbor Embedding (t-SNE)* is optimized for the visualization of large high-dimensional datasets outperforming SM, LLC and MC [97]. This is accomplished by projecting each data point to a two- or three-dimensional map, such that a high similarity leads to short distances. *Uniform Manifold Approximation and Projection (UMAP)* improves t-SNE by better preserving the global structure by using local Riemann manifold approximations and representing it with a fuzzy topological structure [106]. *Multilayer Autoencoders (AE)* are feed-forward neural networks that force the model to compress the data due to their architecture. AEs consist of two networks, one encoder and one decoder reconstructing the data. Their number of hidden layers is odd and they share weights between the input and output layer [98]. By pretraining, the network with *Restricted Boltzmann Machines (RBM)*, the existence of local optima in the objective function can be dealt with. AEs on their own do not cope well with very high-dimensional data as the number of weights is too large. However, by using PCA beforehand, this limitation can be overcome. A comparison of the respective advantages of AEs in contrast to PCA is given in [47]. The computational complexity depends on the target dimensionality $d$, the number of iterations $i$ and the number of weights in a neural network $w$. It is $\mathcal{O}(in^2)$ for SM, $\mathcal{O}(inw)$ for AE, $\mathcal{O}(imd^3)$ for LLC and MC.

In conclusion, an extensive study by Maaten et al. shows that linear and non-linear techniques need to be evaluated on both artificial and real-world datasets as non-linear FE approaches outperform linear ones for complex non-linear data while they perform poorly on natural datasets [98]. They outline future goals to include the development of objective functions that are not impaired by trivial optimal solutions. Also, they suggest that prospective techniques do not base their data representation on neighborhood graphs to model local properties, thereby mitigating the curse of dimensionality. Chao et al. remark that further research has to be done in terms of scalability as well as the ability to cope with missing input values [31]. Furthermore, integrating heterogeneous data will continue to pose a considerable challenge.

## 3. Adaptive Approaches

Adaptive approaches have the ability to change behavior depending on a specific problem to achieve the best possible results. They may act fully automatic or require some degree of manual intervention. In this section, we consider deep learning-based data reduction techniques

and meta-compressors, whose behavior depend on data. The former learn from data on how to achieve optimal data reduction strategy. The latter decide which is the best compression algorithm for a particular data type.

## 3.1. Compression Based on Machine Learning

Machine learning based compression involves two key steps: modeling and coding. While coding can be regarded as a solved problem, there is still no optimal solution for modeling. The difficulty in modeling is building the most compact representation of data. A group of compression algorithms applies various machine learning techniques to get closer to the optimum.

### *3.1.1. Media compression*

In the last years, researchers did considerable progress in media compression with machine learning. Some solutions have already impressive characteristics and may be used as alternatives to traditional approaches. They cover lossy, lossless image and video compression.

Full resolution lossy image compression with recurrent neural networks (RNN) by G. Toderici et al. supports variable compression rates [147]. In the experiments, compression with RNNs (LSTM, associative LSTM) outperforms JPEG for most bit rates. End-to-end optimized image compression in [18] is optimized for a better rate-distortion performance than the standard JPEG and JPEG2000 compression. The evaluation shows a better visual image quality at all bit rates. Real-time adaptive lossy image compression outperforms JPEG, JPEG2000 and WebP [121]. CocoNet is a deep learning approach that learns and maps pixel coordinates to colors [27]. A trained CocoNet-network is able to memorize one single picture and can be used for advanced image processing. Altough, CocoNet is used for image representation, it has also a high potential for image compression. A Learned Lossless Image Compression (L3C) outperforms PNG up to 1.4, WebP and JPEG2000 up to 1.08 in compression ratio [110]. The encoder represents a compressed images as as a set of extracted features, which can be assembled again to an image by a trained predictor. The parallel nature of the approach allows a performant implementation on parallel computer architecture. Deep Learning Video Coding (DLVC) is a deep learning approach, that achieves a CR of more than 2.5 compared to H.265/HEVC, at the same quality level [85, 88]. Internally, DVLC uses a set of different deep learning-based, in particular two CNN-based filters, and different non-learning-based coding techniques.

### *3.1.2. Data compression*

There is also a number of data compressors that can be used as general purpose compressors or are already adapted to HPC community needs. DeepZip uses the capability of neural networks to create arbitrary complex mappings [55]. Together with arithmetic encoders, it works as a powerful lossless compression algorithm for sequential data, like text and genomic datasets. In experiments, it outperforms GZip on real data and achieves near-optimal performance on synthetic data. NUMARCK (Northwestern University Machine learning Algorithm for Resiliency and ChecKpointing) exploits the fact that in many scientific applications, subsequent checkpoints contain insignificant changes [35]. K-Means algorithms optimize forward predictive coding, which codes a checkpoint with reference to a past checkpoint, resulting in lossy compression. DeepSZ is a lossy neural network compressor [67]. It involved key steps, like network pruning, error bound assessment, optimization for error bound configuration, and compressed

model generation, featuring a high compression ratio and low encoding time. Compared with other state-of-the-art methods, DeepSZ can improve the compression ratio by up to 1.43, the DNN encoding performance by up to 4.0 (with four Nvidia Tesla V100 GPUs), and the decoding performance by up to 6.2. Neural networks have also been used in [115] to compress data gathered by Internet of Things devices in a lossy fashion.

Machine learning can also be used in ways that are not traditionally called compression. For instance, in [70], machine learning was used to replace traditional data structures for B-trees, hash maps and bloom filters by other types of models, including deep neural networks. Benchmarks on real-world data show, that neural networks can be up to 70% faster and consume order-of-magnitude less memory than B-trees.

### 3.2. Meta-Compressors

Meta-compressors are high-level data compressors with support of two or more compression algorithms. Algorithm selection can be user-defined, selected according to user requirements (semi-automatic), or can be fully transparent to users (automatic). In automatic and semi-automatic meta-compressors, a decision unit performs usually two tasks. First, it executes a compressibility check on digital data to decide, if compression will be beneficial. Secondly, it selects the most suitable compression algorithm for this particular data set. The most frequently used compressibility checkers are sample-based, but there is also a trend for more advanced deep-learning-based solutions. Sample-based compressibility checkers perform compression tests on sample data and choose an algorithm with the highest compression ratio. Deep-learning-based solutions are more advanced and usually outperform sample-based compressibility checkers [42, 119]. To predict the compressibility, a supervised model is trained with example data of a compression algorithm. A trained model is able to do a pre-analysis of data and check if it will benefit from compression or not, without costly compression. Deep-learning-based decision units can potentially be integrated into file systems with multi-algorithms support. Currently, there is no support in major HPC file systems.

C-Blosc2 is a high-performance lossless meta-compressor optimized for binary data [11]. It supports BloscLz, LZ4, LZ4HC, Zstd, Lizard, and zlib compression algorithms. Aside from achieving the best compression ratios, it is designed for fast data transmission to processor cache and speed-up memory-bound computations. The support of a 64-bit address space allows C-Blosc2 to access large sparse and sequential data, either in-memory or on-disk. Scientific Compression Library (SCIL) supports lossy and lossless compression [74]. Users set high-level instructions on how to handle data, e.g., they can define accuracy, absolute/relative tolerance, significant digits/bits, or relative error for lossy compression. Adaptive Compression Scheme (ACOMPS) is a relatively young research project [138]. It selects the best lossless (LZO, ZLIB, BZIP2, FPC, ISOBAR) or lossy (ZFP, SZ, ISABELA) compression method independently for each variable in a dataset. Another promising research on online selection of two popular lossy compression algorithms, ZFP and SZ, was done in [145].

## 4. Deduplication

Strategies to reduce data can also extend to a higher-level perspective than byte streams, files or objects. Especially, for large scale data management systems and data centers, opportunities to discover large chunks of identical data exist [108, 156]. A common strategy often referred to

as *deduplication* is therefore to hash and index data, and then only reference already existing fragments instead of keeping duplicate copies.

In scientific computing and HPC, redundant or duplicate data can occur in a variety of contexts [108]. Many scientific workflows utilize similar data. Input data is often downloaded by individual users and kept in their project or user directories. Across a large user base, this can amount to significant data volumes. Similar strategies are already employed by large email systems, where emails going to multiple inboxes are not stored multiple times on the server. Kaiser et al. note potential savings across scientific domains ranging from 37% to 99%, with a particularly dominant factor being null regions [68]. Not limited to HPC are backups and database snapshots, which effectively implement deduplication for incremental backups, although they can also be coupled with lower-level strategies. Similarly, images of software environments, as used with virtual machines (VMs) and containerization, typically can share identical system files and directory structures [75]. When base system images are provided, it is possible to only store one copy of the base image instead of keeping copies for each user. A related application comes with software stacks outside of containerization. Here, tooling support for building software like Spack provides the opportunity to offer flexibility to customize software environments, while relying on a selection of maintained site-wide packages as much as possible [50].

Finally, deduplication is commonly used to speed up or avoid unnecessary data transmission. As such wide area network (WAN) optimizations often include deduplication support [112]. In a scientific context, this, for example, becomes relevant as data is replicated between sites through national and international scientific networks.

Taking a closer look at how deduplication is typically implemented, Xia et al. identify five key stages common across many approaches that still apply today [156]. Data is typically first split up into smaller chunks to improve the chance of finding matches. The most important factor here is the granularity of chunks, as such it is common to find both block-level as well as file/object-level deduplication. In the next phase, actual data gets typically hashed to obtain a fingerprint, which allows efficient comparison even across a network. A hash is defined as a function that maps an arbitrary sequence of data to a fixed-size value. To minimize collisions, and thus risking to loose data, cryptographic hash functions are typically being employed. The fingerprints are then stored into index structures to allow efficient lookup. Finally, it is common to store compressed variants of the chunks to leave no opportunity to save space unused. To store and receive the (compressed) chunks, some data management on top of actual storage media is required. There can also be operational benefits to deduplication, which can help with the endurance of SSDs as unnecessary write cycles that damage the cells can be avoided.

Applying deduplication does introduce overheads for keeping index structures and to perform matching. ZFS deduplication, for example, requires additional memory to hold lookup tables [154]. Different storage solutions employ both software-, hardware-based or hybrid approaches, in addition to inline and out-of-band deduplication. Software-based approaches used to offer more flexibility at the cost of performance. Research into approaches using FPGAs such as CIDR might allow hardware-accelerated deduplication without sacrificing flexibility [6].

There is a number of security and privacy considerations to be aware of when employing deduplication. Deduplication can reduce data safety as the impact of data corruption increases. Some cloud providers used deduplication, which allowed to feign ownership of a file by transmitting a wrong hash, and this way extract sensible data. Research into proof of ownership methods offers strategies to mitigate this [59]. Similarly, when dealing with encrypted data, deduplication

cannot be applied across users as logically identical data would have different signatures. This notion of local and global deduplication applies also to scientific data, as reductions can be performed at the application, the node or the system level with different trade-offs [68].

# 5. In Situ Processing

Idle cycles are usually available on computing nodes due to the discrepancy between the computing capabilities and the I/O transfer rates. A solution to address this inefficiency is to use the available idle nodes for performing different computations on the data, which is already in memory. This process is called *in situ processing*.

## 5.1. Algorithms and Techniques

In situ algorithms usually focus on data analysis and data reduction for different purposes including logging, filtering, compression and visualization. Despite the fact that some of the generic algorithms could potentially be applied in situ, the majority of them do not satisfy the main restriction imposed by this context: use the available CPU(s) and available memory while keeping the impact on the application's performance at a negligible level. Therefore, specific techniques were developed or adapted for this environment. This section will look into some of them together with their applicabilities.

### *5.1.1. Data analysis*

Efficient in situ data analysis techniques include feature extraction, feature tracking and region growing techniques. These are particular to each application because they depend on the data structure and data flow. Machine learning solutions for dimensionality reduction, clustering or classification are gaining much traction in recent years. Using deep convolutional autoencoders for in situ data reduction proves good results for feature extraction from large carbon particle simulation datasets [89].

### *5.1.2. Data compression*

Data is usually compressed for visualization and storage reduction. Due to a high level of entropy, the simulated data is a difficult candidate for the majority of the general compression algorithms presented in the previous sections. A study of state-of-the-art compression algorithms and their efficiencies for in situ environments recommends lossy methods like FPZIP, scalar quantization, Discrete Fourier and Wavelet Transforms and tailored in situ methods like ISOBAR [129] and ISABELA [38, 78]. The last two approaches are briefly explained next.

*In Situ Orthogonal Byte Aggregate Reduction (ISOBAR)* uses a preconditioner to enhance the performance of a general lossless compressor. The main components of the ISOBAR preconditioner are the analyzer, which evaluates the data compressibility at the byte level and the partitioner, which splits the data into compressible bytes, incompressible bytes, and metadata. The compressible chunks are then compressed using a lossless compressor chosen by the EUPA-selector based on an efficient linearization strategy and user's preference for either a high compression ratio or high compression throughput. In the end, the merger reassembles the compressed bytes, the incompressible bytes and the metadata into the final output.

*In situ Sort-And-B-spline Error-bounded Lossy Abatement (ISABELA)* obtains a higher degree of data reduction than the lossless ISOBAR but at the price of accuracy. The method splits the data into fixed-sized windows and applies a preconditioner to sort the data from each window into a monotonically increasing curve. These curves are later approximated using cubic B-splines, which can be easily modeled with a low number of coefficients.

Wavelet compression paired with state-of-the-art floating-point compressors, MPI and OpenMP, provides a performant parallel and distributed solution for in situ compression [58]. Reallocating the I/O resources dynamically based on the coefficient magnitudes of the wavelet method and Shannon entropy leads to a new method that displays good results for simulations with imbalanced data complexity [101].

A more novel method employs Generative Adversarial Network to compress data from computational fluid dynamics simulations [91]. The authors use the discriminative neural network to compress the data on the compute nodes while the generative network handles the reconstruction on the visualization nodes.

### *5.1.3. Data visualization*

In situ data visualization is extremely important in large, complex applications because it allows not only following the real-time simulation but also steering it if required. Kress describes the in situ visualization technologies and surveys the most popular libraries and frameworks with their advantages and disadvantages in [71]. The main technologies are briefly compared next.

**Table 1.** The briefly comparison of the main technologies

| Technology / Characteristic | Tightly Coupled | Loosely Coupled |
|---|---|---|
| Visualization and simulation share the CPU(s) | yes | no |
| Visualization and simulation share the memory | yes | no |
| Data duplication (double RAM usage) | no | yes |
| Visualization needs network access to retrieve the data | no | yes |
| Coordination between visualization and simulation | minimal | intensive |
| Computational steering capabilities | yes | limited |
| Visualization added costs | RAM and CPU | nodes and network |
| Scalable | no | yes |
| Fault tolerance | no | yes |

There is a hybrid approach combining flexibility and computational steering support. The choice which of the three technologies is more appropriate for a given use case is based on the hardware architecture, the available resources and the particularities of the simulation data.

A comprehensive comparison between the two main visualization tools which set the ground for the majority of the in situ frameworks – ParaView [2] and VisIt [36] is described in [122].

## 5.2. Frameworks and Infrastructure

While in situ processing is best performed at the application level of the software stack, middleware and toolkits can increase their performance by facilitating the data extraction. Examples of such frameworks include EPIC [44] and Freeprocessing [46]. A solution based on I/O layer components was proposed in [26]. Remote direct memory access (RDMA) can also contribute by enabling zero-copy, high-throughput and low-latency networking for HPC clusters [38].

Well-known frameworks focused on visualization include ParaView Catalyst [14], Strawman [79], VisIt LibSim [153] and Damaris/Viz [43]. Besides visualization capabilities, some frameworks offer computational steering too. CUMULVS [52] and ISAAC [105] are just two of them. Other in situ scenarios like clustering, covered by KeyBin2 [34] and compression, covered by CubismZ [58] complete the list of functionalities offered by this environment.

## 6. Recomputation and Reproducibility

A rather unconventional approach for reducing the amount of data that has to be stored is recomputation, that is, instead of storing experiment or simulation results within a storage system, they are recomputed on demand. This technique can be combined with in situ methods.

To be able to recompute results, a hard prerequisite is reproducibility. Reproducing experiments requires all necessary input data, software, scripts etc. to be archived and documented. Popper is a convention for creating reproducible scientific publications and makes use of best practices established open-source software development [66]. While Popper is tuned towards scientific publications, this also includes all experiments that have been performed for such a publication and can, therefore, be easily adapted for general experiments. Experiments may either be deployed locally with Docker [61] or in the cloud with Ansible [104].

The ReScience initiative has launched a new peer-reviewed journal that tries to tackle replication problems by using a new publication approach [124]. The whole review and publication process is hosted on GitHub and anyone with a GitHub account is able to comment on a submitted publication. Reviewers then try to replicate the submitted results. The authors define reproducibility as the ability to arrive at the same results as a publication when using the same code and data as used for the publication itself. Replication, however, means that new code can be written for a computational model or method to obtain the same results.

The DataONE Data Package standard provides a specification for packaging together data, software and visualizations as well as accompanying metadata [107]. In combination with the WholeTale project, arbitrary computational results can be reproduced [28]. For instance, the computing environment can be described with a Dockerfile that can be used by interested researchers to rebuild the exact environment.

There are also domain-specific frameworks and toolkits. For instance, [65] introduces an R-based framework called climate4R that aims to standardize the tools used for accessing, harmonizing and post-processing climate data. It provides access to both local and remote data and features built-in support for a wide range of remote data sources. SwarmRob is a toolkit for making robotics research reproducible [117]. In addition to providing a toolkit, the authors propose a new workflow that is separated into research and review phases. In the research phase, authors specify needed services in a Container Definition File as well as the services' configurations and interactions in an Experiment Definition File. The experiment can then be performed by the SwarmRob toolkit using these two files. In the review phase, reviewers are able use the same infrastructure to reproduce the authors' results. DUGONG is a preconfigured Docker container for bioinformatics and computational biology research [109]. It packages over 3,000 dependencies and applications, including Jupyter Notebook. By providing a common software base for research, results can be reproduced more easily.

Scientific applications often have a multitude of dependencies that have to be provided in specific configurations and versions that are not provided by operating system distributions. Therefore, scientists often have to resort to installing dependencies from source manually. Their

manual processes are problematic with regard to reproducibility. A convenient way to manage all dependencies required by the actual application is package managers. EasyBuild helps manage complex scientific software stacks by providing recipes for popular packages and additional features [12]. Spack works in a similar way but specializes in supporting combinatorial builds such that software can be used easily with a wide range of different configurations and versions [50]. Moreover, Spack computes hashes for each of its packages, further helping reproducibility.

Singularity is a container platform that allows running unprivileged users to run containers on typical HPC systems due to its integration with common batch schedulers such as SLURM [54]. It has built-in support for signing and verification of containers as well as portability and reproducibility. Even though applications running within the containers are isolated from the host environment, Singularity still allows access to host hardware such as GPUs and InfiniBand for improved performance. The authors of [135] make use of Singularity and Spack to provide an in situ software stack that can be moved between HPC machines easily. These containers can then be run on systems that support executing containers. Such an approach may also be used for reproducibility by providing the full container image to interested third parties.

In addition to containerizing application code, input and output data also needs to be handled since it is typically read from or written to a traditional shared storage system. Data Pallets are an approach for encapsulating output data within containers [92]. These Data Pallets can be then be passed from one step of a workflow to the next. Unique container and dependency IDs are used for provenance. Moreover, no application changes are necessary since the workflow system can take care of managing Data Pallets.

To summarize, there are several approaches to improve the reproducibility of experiments. However, to be able to recompute data, it is necessary to be able to regenerate bit-identical results. This introduces additional requirements since even changes to the underlying hardware architecture might cause minuscule changes in the output data. If bit-identical results are not necessary, available reproducibility approaches can already be used to recompute data even over longer periods of time. Care must be taken to ensure that applications stay executable by updating the infrastructure and archived artifacts to the current state of the art. Therefore, recomputation introduces additional overhead regarding software and data management.

## Conclusion

As data volumes continue to grow, but the gap between computational throughput and I/O bandwidth widens, data reduction techniques are becoming more important. Many established technologies continue to remain highly relevant such as lossless/lossy compression or deduplication, which are routinely applied across different layers. In some areas, general-purpose solutions seem unlikely to meet future requirements, which is why research increasingly explores specialized and adaptive approaches that find the most appropriate representation depending on the data. As new systems and applications are designed with asynchronous processing pipelines mind, in situ approaches are gaining momentum. Often, they allow reducing data volumes by several orders of magnitude by filtering, aggregation or transformation at the edge or in transit. This paradigm shift in programming and the promotion of reproducible research also allows considering recomputation as a viable alternative for data that is accessed infrequently.

Many of the covered technologies do not exist in isolation but are combined. Compression remains one of the most popular and obvious methods that can be employed throughout the entire HPC stack in software and hardware. As a result, hardware acceleration is expected

to considerably improve performance and help reduce overheads. The overall performance of lossless compression algorithms often depends on the input, requiring research into adaptive approaches. When applying lossy compression, scientists are often hesitating which information can be safely discarded without interfering with the intentions of their workflow and without hurting unanticipated use too much. Here, keeping provenance information is important, but adoption is primarily hindered by a lack of guidance on the disadvantages and advantages of lossy compression for users. To spare users from needing to evaluate which specialized approaches are most suitable, frameworks to automate this decision are gaining popularity, but research into adaptive data-aware solutions is needed.

Currently, adaptive approaches can be categorized into methods using heuristics to predict and pick the best performing methods on the one hand and methods to transform data using machine learning or dimensionality reduction on the other. Predictors achieve good performance on specific data sets but are typically trained on a selection of training datasets, which introduces a bias and thus limits generalization. This also extends to methods for adaptive data transformation and dimensionality reduction, where most methods exploit specific structural properties but overlook others. A promising exception are approaches based on autoencoders which learn a compact representation without supervision. Specialized methods are useful, but a dialog in the community to curate representative datasets for data reduction is needed too.

Even though great progress has been achieved in the last 10 years with respect to in situ techniques, the disparity between computational throughput and I/O bandwidth has not been solved yet. Firstly, efforts are still being done in increasing the compression rates for the lossless algorithms and to bound the precision loss in lossy algorithms. Secondly, the HPC clusters become more heterogeneous with the addition of GPUs. This forces the in situ algorithms to be redesigned in order to efficiently use the new hardware which otherwise would not meet the low CPU usage requirements. Last but not least, deep learning gets increasingly more attention due to the very promising results obtained by the usage of the generative models. While continuing the efforts to improve well-known data analysis and visualization mechanisms, new ideas like in situ virtual reality are starting to emerge [56]. To enable further improvement, the development of additional scientific benchmarks to evaluate data reduction techniques are necessary.

## Acknowledgements

Parts of this publication were enabled by the following projects: CoSEMoS[4], funded by the German Research Foundation (DFG) under grant KU 3584/1-1; the Helmholtz graduate school for the structure of matter DASHH[5]; the Intel Parallel Computing Center on Enhanced Adaptive Compression in Lustre, funded by Intel[6]; i_SSS[7], funded by BASF SE. Bull Cooperation with the research department of DKRZ to improve I/O for climate applications on the Mistral HPC.



[4] `https://cosemos.de`
[5] `https://www.dashh.org/`
[6] `https://wr.informatik.uni-hamburg.de/research/projects/ipcc-l/start`
[7] `https://wr.informatik.uni-hamburg.de/research/projects/i_sss/start`

## References


1. Abdelfattah, M.S., Hagiescu, A., Singh, D.: Gzip on a chip: high performance lossless data compression on FPGAs using OpenCL. In: McIntosh-Smith, S., Bergen, B. (eds.) Proceedings of the International Workshop on OpenCL, IWOCL 2013 & 2014, 13-14 May 2013, Georgia Tech, Atlanta, GA, USA / 12-13 May 2014 Bristol, UK. pp. 4:1–4:9. ACM (2014), DOI: 10.1145/2664666.2664670

2. Ahrens, J.P., Geveci, B., Law, C.C.: ParaView: An End-User Tool for Large-Data Visualization. In: Hansen, C.D., Johnson, C.R. (eds.) The Visualization Handbook, pp. 717–731. Academic Press / Elsevier (2005), DOI: 10.1016/b978-012387582-2/50038-1

3. Ainsworth, M., Tugluk, O., Whitney, B., et al.: Multilevel techniques for compression and reduction of scientific data - the univariate case. Computat. and Visualiz. in Science 19(5-6), 65–76 (2018), DOI: 10.1007/s00791-018-00303-9

4. Ainsworth, M., Tugluk, O., Whitney, B., et al.: Multilevel Techniques for Compression and Reduction of Scientific Data - The Multivariate Case. SIAM J. Scientific Computing 41(2), A1278–A1303 (2019), DOI: 10.1137/18M1166651

5. Ainsworth, M., Tugluk, O., Whitney, B., et al.: Multilevel Techniques for Compression and Reduction of Scientific Data-Quantitative Control of Accuracy in Derived Quantities. SIAM J. Scientific Computing 41(4), A2146–A2171 (2019), DOI: 10.1137/18M1208885

6. Ajdari, M., Park, P., Kim, J., et al.: CIDR: A cost-effective in-line data reduction system for terabit-per-second scale SSD arrays. In: 25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, 16-20 Feb. 2019, Washington, DC, USA. pp. 28–41. IEEE (2019), DOI: 10.1109/HPCA.2019.00025

7. Akenine-Möller, T., Ström, J.: Graphics Processing Units for Handhelds. Proceedings of the IEEE 96(5), 779–789 (2008), DOI: 10.1109/JPROC.2008.917719

8. Alakuijala, J., Farruggia, A., Ferragina, P., et al.: Brotli: A general-purpose data compressor. ACM Trans. Inf. Syst. 37(1), 4:1–4:30 (2019), DOI: 10.1145/3231935

9. Alameldeen, A.R., Wood, D.A.: Adaptive Cache Compression for High-Performance Processors. In: 31st International Symposium on Computer Architecture, ISCA 2004, 19-23 June 2004, Munich, Germany. pp. 212–223. IEEE Computer Society (2004), DOI: 10.1109/ISCA.2004.1310776

10. Alforov, Y., Ludwig, T., Novikova, A., et al.: Towards Green Scientific Data Compression Through High-Level I/O Interfaces. In: 30th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2018, 24-27 Sept. 2018, Lyon, France. pp. 209–216. IEEE (2018), DOI: 10.1109/CAHPC.2018.8645921

11. Alted, F.: Blosc2-Meets-Rome. `https://blosc.org` (2019), accessed: 2020-02-17

12. Alvarez, D., Cais, A.Ó., Geimer, M., et al.: Scientific Software Management in Real Life: Deployment of EasyBuild on a Large Scale System. In: 2016 Third International Workshop on HPC User Support Tools, HUST@SC 2016, 13 Nov. 2016, Salt Lake City, UT, USA. pp. 31–40. IEEE Computer Society (2016), DOI: 10.1109/HUST.2016.009

13. Amlekar, S.: Compression support in Spectrum Scale 5.0.0. https://developer.ibm.com/storage/2018/01/11/compression-support-spectrum-scale-5-0-0/ (2018), accessed: 2020-02-20

14. Ayachit, U., Bauer, A.C., Geveci, B., et al.: ParaView Catalyst: Enabling In Situ Data Analysis and Visualization. In: Weber, G.H. (ed.) Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV 2015, 15-20 Nov. 2015, Austin, TX, USA. pp. 25–29. ACM (2015), DOI: 10.1145/2828612.2828624

15. Azzurri, P.: Track Reconstruction Performance in CMS. Nuclear Physics B - Proceedings Supplements 197(1), 275–278 (2009), DOI: 10.1016/j.nuclphysbps.2009.10.084

16. Baker, A.H., Hammerling, D., Turton, T.L.: Evaluating image quality measures to assess the impact of lossy data compression applied to climate simulation data. Comput. Graph. Forum 38(3), 517–528 (2019), DOI: 10.1111/cgf.13707

17. Balkenhol, B., Kurtz, S.: Universal Data Compression Based on the Burrows-Wheeler Transformation: Theory and Practice. IEEE Trans. Computers 49(10), 1043–1053 (2000), DOI: 10.1109/12.888040

18. Ballé, J., Laparra, V., Simoncelli, E.P.: End-to-end Optimized Image Compression. CoRR abs/1611.01704 (2016), http://arxiv.org/abs/1611.01704

19. Ballester-Ripoll, R., Lindstrom, P., Pajarola, R.: TTHRESH: Tensor Compression for Multidimensional Visual Data. CoRR abs/1806.05952 (2018), http://arxiv.org/abs/1806.05952

20. Barbay, J.: Optimal Prefix Free Codes with Partial Sorting. Algorithms 13(1), 12 (2020), DOI: 10.3390/a13010012

21. Barr, K.C., Asanovic, K.: Energy-aware lossless data compression. ACM Trans. Comput. Syst. 24(3), 250–291 (2006), DOI: 10.1145/1151690.1151692

22. Baudat, G., Anouar, F.: Generalized Discriminant Analysis Using a Kernel Approach. Neural Computation 12(10), 2385–2404 (2000), DOI: 10.1162/089976600300014980

23. Bellman, R., Lee, E.: History and development of dynamic programming. IEEE Control Systems Magazine 4(4), 24–28 (1984), DOI: 10.1109/MCS.1984.1104824

24. Bogaardt, L., Goncalves, R., Zurita-Milla, R., et al.: Dataset Reduction Techniques to Speed Up SVD Analyses on Big Geo-Datasets. ISPRS Int. J. Geo-Information 8(2), 55 (2019), DOI: 10.3390/ijgi8020055

25. Bookstein, A., Klein, S.T.: Is Huffman coding dead? Computing 50(4), 279–296 (1993), DOI: 10.1007/BF02243872

26. Boyuka II, D.A., Lakshminarasimhan, S., Zou, X., et al.: Transparent in Situ Data Transformations in ADIOS. In: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, 26-29 May 2014, Chicago, IL, USA. pp. 256–266. IEEE Computer Society (2014), DOI: 10.1109/CCGrid.2014.73

27. Bricman, P.A., Ionescu, R.T.: CocoNet: A deep neural network for mapping pixel coordinates to color values. CoRR abs/1805.11357 (2018), http://arxiv.org/abs/1805.11357

28. Brinckman, A., Chard, K., Gaffney, N., et al.: Computing environments for reproducibility: Capturing the "Whole Tale". Future Generation Comp. Syst. 94, 854–867 (2019), DOI: 10.1016/j.future.2017.12.029

29. Canal, R., González, A., Smith, J.E.: Very low power pipelines using significance compression. In: Wolfe, A., Schlansker, M.S. (eds.) Proc. of the 33rd Annual IEEE/ACM Int. Symposium on Microarchitecture, MICRO 33, 10-13 Dec. 2000, Monterey, California, USA. pp. 181–190. ACM/IEEE Computer Society (2000), DOI: 10.1109/MICRO.2000.898069

30. Cappello, F., Di, S., Li, S., et al.: Use cases of lossy compression for floating-point data in scientific data sets. IJHPCA 33(6) (2019), DOI: 10.1177/1094342019853336

31. Chao, G., Luo, Y., Ding, W.: Recent Advances in Supervised Dimension Reduction: A Survey. Machine Learning and Knowledge Extraction 1(1), 341–358 (2019), DOI: 10.3390/make1010020

32. Chen, K., Ramabadran, T.V.: Near-lossless compression of medical images through entropy-coded DPCM. IEEE Trans. Med. Imaging 13(3), 538–548 (1994), DOI: 10.1109/42.310885

33. Chen, X., Yang, L., Dick, R.P., et al.: C-Pack: A High-Performance Microprocessor Cache Compression Algorithm. IEEE Trans. VLSI Syst. 18(8), 1196–1208 (2010), DOI: 10.1109/TVLSI.2009.2020989

34. Chen, X., Benson, J., Peterson, M., et al.: KeyBin2: Distributed Clustering for Scalable and In-Situ Analysis. In: Proceedings of the 47th International Conference on Parallel Processing, ICPP 2018, 13-16 Aug. 2018, Eugene, OR, USA. pp. 34:1–34:10. ACM (2018), DOI: 10.1145/3225058.3225149

35. Chen, Z., Son, S.W., Hendrix, W., et al.: NUMARCK: machine learning algorithm for resiliency and checkpointing. In: Damkroger, T., Dongarra, J.J. (eds.) International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, 16-21 Nov. 2014, New Orleans, LA, USA. pp. 733–744. IEEE Computer Society (2014), DOI: 10.1109/SC.2014.65

36. Childs, H., Brugger, E., Whitlock, B., et al.: Visit. In: Bethel, E.W., Childs, H., Hansen, C.D. (eds.) High Performance Visualization - Enabling Extreme-Scale Scientific Insight. Chapman and Hall / CRC computational science series, CRC Press (2012), DOI: 10.1201/b12985-21

37. Cliff, A., Romero, J., Kainer, D., et al.: A High-Performance Computing Implementation of Iterative Random Forest for the Creation of Predictive Expression Networks. Genes 10(12), 996 (2019), DOI: 10.3390/genes10120996

38. Critchlow, T., van Dam, K.K.: Data-Intensive Science. CRC Press (2013)

39. Cunningham, J.P., Ghahramani, Z.: Linear dimensionality reduction: survey, insights, and generalizations. J. Mach. Learn. Res. 16, 2859–2900 (2015), http://dl.acm.org/citation.cfm?id=2912091

40. Delaunay, X., Courtois, A., Gouillon, F.: Evaluation of lossless and lossy algorithms for the compression of scientific datasets in netCDF-4 or HDF5 files. Geoscientific Model Development 12(9), 4099–4113 (2019), DOI: 10.5194/gmd-12-4099-2019

41. Di, S., Cappello, F.: Fast Error-Bounded Lossy HPC Data Compression with SZ. In: 2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, 23-27 May 2016, Chicago, IL, USA. pp. 730–739. IEEE Computer Society (2016), DOI: 10.1109/IPDPS.2016.11

42. Diederich, M., Doerk, T., Muehge, T., et al.: Decision-based data compression by means of deep learning technologies (2018), `https://patentswarm.com/patents/US20190221192A1`, application US 20180277068 A1

43. Dorier, M., Sisneros, R., Peterka, T., et al.: Damaris/Viz: A nonintrusive, adaptable and user-friendly in situ visualization framework. In: Geveci, B., Pfister, H., Vishwanath, V. (eds.) IEEE Symposium on Large-Scale Data Analysis and Visualization, LDAV 2013, 13-14 Oct. 2013, Atlanta, Georgia, USA. pp. 67–75. IEEE Computer Society (2013), DOI: 10.1109/LDAV.2013.6675160

44. Duque, E.P., Hiepler, D.E., Haimes, R., et al.: EPIC - An Extract Plug-In Components Toolkit for In-Situ Data Extracts Architecture. DOI: 10.2514/6.2015-3410

45. Filgueira, R., Singh, D.E., Pichel, J.C., et al.: Exploiting data compression in collective I/O techniques. In: Proceedings of the 2008 IEEE International Conference on Cluster Computing, 29 Sept.-1 Oct. 2008, Tsukuba, Japan. pp. 479–485. IEEE Computer Society (2008), DOI: 10.1109/CLUSTR.2008.4663811

46. Fogal, T., Proch, F., Schiewe, A., et al.: Freeprocessing: Transparent in situ Visualization via Data Interception. In: Amor, M., Hadwiger, M. (eds.) Eurographics Symposium on Parallel Graphics and Visualization, Swansea, Wales, UK. pp. 49–56. Eurographics Association (2014), DOI: 10.2312/pgv.20141084

47. Fournier, Q., Aloise, D.: Empirical Comparison between Autoencoders and Traditional Dimensionality Reduction Methods. In: 2nd IEEE International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2019, 3-5 June 2019, Sardinia, Italy. pp. 211–214. IEEE (2019), DOI: 10.1109/AIKE.2019.00044

48. Fowers, J., Kim, J., Burger, D., et al.: A Scalable High-Bandwidth Architecture for Lossless Compression on FPGAs. In: 23rd IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2015, 2-6 May 2015, Vancouver, BC, Canada. pp. 52–59. IEEE Computer Society (2015), DOI: 10.1109/FCCM.2015.46

49. Fukunaga, K., Olsen, D.R.: An Algorithm for Finding Intrinsic Dimensionality of Data. IEEE Trans. Computers 20(2), 176–183 (1971), DOI: 10.1109/T-C.1971.223208

50. Gamblin, T., LeGendre, M.P., Collette, M.R., et al.: The Spack package manager: bringing order to HPC software chaos. In: Kern, J., Vetter, J.S. (eds.) Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, 15-20 Nov. 2015, Austin, TX, USA. pp. 40:1–40:12. ACM (2015), DOI: 10.1145/2807591.2807623

51. Geist, A., Reed, D.A.: A survey of high-performance computing scaling challenges. IJHPCA 31(1), 104–113 (2017), DOI: 10.1177/1094342015597083

52. Geist II, G.A., Kohl, J.A., Papadopoulos, P.M.: Cumulvs: Providing Fault Tolerance, Visualization, and Steering of Parallel Applications. IJHPCA 11(3), 224–235 (1997), DOI: 10.1177/109434209701100305

53. Gilchrist, J.: Parallel data compression with bzip2. In: Proc. of the 16th IASTED int. conf. on parallel and distributed computing and systems. vol. 16, pp. 559–564 (2004)

54. Godlove, D.: Singularity: Simple, secure containers for compute-driven workloads. In: Furlani, T.R. (ed.) Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning), PEARC 2019, 28 July-1 Aug. 2019, Chicago, IL, USA. pp. 24:1–24:4. ACM (2019), DOI: 10.1145/3332186.3332192

55. Goyal, M., Tatwawadi, K., Chandak, S., et al.: DeepZip: Lossless Data Compression using Recurrent Neural Networks. CoRR abs/1811.08162 (2018), http://arxiv.org/abs/1811.08162

56. Gupta, A., Günther, U., Incardona, P., et al.: A Proposed Framework for Interactive Virtual Reality In Situ Visualization of Parallel Numerical Simulations. CoRR abs/1909.02986 (2019), http://arxiv.org/abs/1909.02986

57. Guyon, I., Elisseeff, A.: An Introduction to Variable and Feature Selection. J. Mach. Learn. Res. 3, 1157–1182 (2003), http://jmlr.org/papers/v3/guyon03a.html

58. Hadjidoukas, P.E., Wermelinger, F.: A Parallel Data Compression Framework for Large Scale 3D Scientific Data. CoRR abs/1903.07761 (2019), http://arxiv.org/abs/1903.07761

59. Halevi, S., Harnik, D., Pinkas, B., et al.: Proofs of ownership in remote storage systems. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, 17-21 Oct. 2011, Chicago, Illinois, USA. pp. 491–500. ACM (2011), DOI: 10.1145/2046707.2046765

60. Halkiadakis, E.: Proceedings for TASI 2009 Summer School on "Physics of the Large and the Small": Introduction to the LHC experiments (2010)

61. Higgins, J., Holmes, V., Venters, C.C.: Orchestrating Docker Containers in the HPC Environment. In: Kunkel, J.M., Ludwig, T. (eds.) High Performance Computing - 30th Int. Conf., 12-16 July 2015, Frankfurt, Germany. Lecture Notes in Computer Science, vol. 9137, pp. 506–513. Springer (2015), DOI: 10.1007/978-3-319-20119-1_36

62. Hu, X., Wang, F., Li, W., et al.: QZFS: QAT Accelerated Compression in File System for Application Agnostic and Cost Efficient Data Storage. In: Malkhi, D., Tsafrir, D. (eds.) 2019 USENIX Annual Technical Conference, USENIX ATC 2019, 10-12 July 2019, Renton, WA, USA. pp. 163–176. USENIX Association (2019), https://www.usenix.org/conference/atc19/presentation/hu-xiaokang

63. Huang, C., Harris, R.W.: A comparison of several vector quantization codebook generation approaches. IEEE Trans. Image Processing 2(1), 108–112 (1993), DOI: 10.1109/83.210871

64. Ibarria, L., Lindstrom, P., Rossignac, J., et al.: Out-of-core Compression and Decompression of Large n-dimensional Scalar Fields. Comput. Graph. Forum 22(3), 343–348 (2003), DOI: 10.1111/1467-8659.00681

65. Iturbide, M., Bedia, J., Garcia, S.H., et al.: The R-based climate4R open framework for reproducible climate data access and post-processing. Environmental Modelling and Software 111, 42–54 (2019), DOI: 10.1016/j.envsoft.2018.09.009

66. Jimenez, I., Sevilla, M., Watkins, N., et al.: The Popper Convention: Making Reproducible Systems Evaluation Practical. In: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2017, 29 May-2 June 2017, Orlando / Buena Vista, FL, USA. pp. 1561–1570. IEEE Computer Society (2017), DOI: 10.1109/IPDPSW.2017.157

67. Jin, S., Di, S., Liang, X., et al.: DeepSZ: A Novel Framework to Compress Deep Neural Networks by Using Error-Bounded Lossy Compression. CoRR abs/1901.09124 (2019), http://arxiv.org/abs/1901.09124

68. Kaiser, J., Gad, R., Süß, T., et al.: Deduplication Potential of HPC Applications' Checkpoints. In: 2016 IEEE International Conference on Cluster Computing, CLUSTER 2016, 12-16 Sept. 2016, Taipei, Taiwan. pp. 413–422. IEEE Computer Society, DOI: 10.1109/CLUSTER.2016.32

69. Kane, J., Yang, Q.: Compression Speed Enhancements to LZO for Multi-core Systems. In: Panetta, J., Moreira, J.E., Padua, D.A., et al. (eds.) IEEE 24th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2012, 24-26 Oct. 2012, New York, NY, USA. pp. 108–115. IEEE Computer Society (2012), DOI: 10.1109/SBAC-PAD.2012.29

70. Kraska, T., Beutel, A., Chi, E.H., et al.: The Case for Learned Index Structures. In: Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, 10-15 June 2018, Houston, TX, USA. pp. 489–504 (2018), DOI: 10.1145/3183713.3196909

71. Kress, J.: In Situ Visualization Techniques for High Performance Computing. http://www.cs.uoregon.edu/Reports/AREA-201703-Kress.pdf (2017), accessed: 2020-01-23

72. Kuhn, M., Kunkel, J., Ludwig, T.: Data Compression for Climate Data. Supercomputing Frontiers and Innovations 3(1), 75–94 (2016), DOI: 10.14529/jsfi160105

73. Kumar, A., Zhu, X., Tu, Y., et al.: Compression in Molecular Simulation Datasets. In: Sun, C., Fang, F., Zhou, Z., et al. (eds.) Intelligence Science and Big Data Engineering - 4th International Conference, IScIDE 2013, 31 July-2 Aug. 2013, Beijing, China, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8261, pp. 22–29. Springer (2013), DOI: 10.1007/978-3-642-42057-3_4

74. Kunkel, J., Novikova, A., Betke, E.: Towards Decoupling the Selection of Compression Algorithms from Quality Constraints An Investigation of Lossy Compression Efficiency. Supercomputing Frontiers and Innovations 4(4) (2017), DOI: 10.14529/jsfi170402

75. Kurtzer, G.M., Sochat, V., Bauer, M.W.: Singularity: Scientific containers for mobility of compute. PLOS ONE 12(5), 1–20 (2017), DOI: 10.1371/journal.pone.0177459

76. Lakhani, G.: Reducing coding redundancy in LZW. Inf. Sci. 176(10), 1417–1434 (2006), DOI: 10.1016/j.ins.2005.03.007

77. Lakshminarasimhan, S., Shah, N., Ethier, S., et al.: Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-temporal Data. In: Jeannot, E., Namyst, R., Roman, J. (eds.) Euro-Par 2011 Parallel Processing - 17th International Conference, Euro-Par 2011, 29 Aug.-2 Sept. 2011, Bordeaux, France, Proceedings, Part I. Lecture Notes in Computer Science, vol. 6852, pp. 366–379. Springer (2011), DOI: 10.1007/978-3-642-23400-2_34

78. Lakshminarasimhan, S., Shah, N., Ethier, S., et al.: ISABELA for effective in situ compression of scientific data. Concurrency and Computation: Practice and Experience 25(4), 524–540 (2013), DOI: 10.1002/cpe.2887

79. Larsen, M., Brugger, E., Childs, H., et al.: Strawman: A Batch In Situ Visualization and Analysis Infrastructure for Multi-Physics Simulation Codes. In: Weber, G.H. (ed.) Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV 2015, 15-20 Nov. 2015, Austin, TX, USA. pp. 30–35. ACM (2015), DOI: 10.1145/2828612.2828625

80. Lee, S.M., Jang, J.H., Oh, J., et al.: Design of hardware accelerator for Lempel-Ziv 4 (LZ4) compression. IEICE Electronic Express 14(11), 20170399 (2017), DOI: 10.1587/elex.14.20170399

81. Li, B., Zhang, L., Shang, Z., et al.: Implementation of LZMA compression algorithm on FPGA. Electronics Letters 50(21), 1522–1524 (2014), DOI: 10.1049/el.2014.1734

82. Li, S., Marsaglia, N., Garth, C., et al.: Data Reduction Techniques for Simulation, Visualization and Data Analysis. Comput. Graph. Forum 37(6), 422–447 (2018), DOI: 10.1111/cgf.13336

83. Li, W., Yao, Y.: Accelerate Data Compression in File System. In: Bilgin, A., Marcellin, M.W., Serra-Sagristà, J., et al. (eds.) 2016 Data Compression Conference, DCC 2016, 30 March-1 April 2016, Snowbird, UT, USA. p. 615. IEEE (2016), DOI: 10.1109/DCC.2016.24

84. Liang, X., Di, S., Tao, D., et al.: Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets. In: Abe, N., Liu, H., Pu, C., et al. (eds.) IEEE International Conference on Big Data, Big Data 2018, 10-13 Dec. 2018, Seattle, WA, USA. pp. 438–447. IEEE (2018), DOI: 10.1109/BigData.2018.8622520

85. Lin, J., Hu, Y., Liu, D.: Deep Learning-Based Video Coding (DLVC). `http://dlvc.bitahub.com/` (2020), accessed: 2020-02-20

86. Lindstrom, P.: Fixed-Rate Compressed Floating-Point Arrays. IEEE Trans. Vis. Comput. Graph. 20(12), 2674–2683 (2014), DOI: 10.1109/TVCG.2014.2346458

87. Lindstrom, P., Isenburg, M.: Fast and Efficient Compression of Floating-Point Data. IEEE Trans. Vis. Comput. Graph. 12(5), 1245–1250 (2006), DOI: 10.1109/TVCG.2006.143

88. Liu, D., Li, Y., Lin, J., et al.: Deep Learning-Based Video Coding: A Review and A Case Study. CoRR abs/1904.12462 (2019), http://arxiv.org/abs/1904.12462

89. Liu, Q., Hazarika, S., Patchett, J.M., et al.: Deep Learning-Based Feature-Aware Data Modeling for Complex Physics Simulations. CoRR abs/1912.03587 (2019), http://arxiv.org/abs/1912.03587

90. Liu, W., Mei, F., Wang, C., et al.: Data Compression Device Based on Modified LZ4 Algorithm. IEEE Trans. Consumer Electronics 64(1), 110–117 (2018), DOI: 10.1109/TCE.2018.2810480

91. Liu, Y., Wang, Y., Deng, L., et al.: A novel in situ compression method for CFD data based on generative adversarial network. J. Visualization 22(1), 95–108 (2019), DOI: 10.1007/s12650-018-0519-x

92. Lofstead, J.F., Baker, J., Younge, A.: Data Pallets: Containerizing Storage for Reproducibility and Traceability. In: Weiland, M., Juckeland, G., Alam, S.R., et al. (eds.) High Performance Computing - ISC High Performance 2019 International Workshops, 16-20 June 2019, Frankfurt, Germany, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11887, pp. 36–45. Springer (2019), DOI: 10.1007/978-3-030-34356-9_4

93. Lu, T., Liu, Q., He, X., et al.: Understanding and Modeling Lossy Compression Schemes on HPC Scientific Data. In: 2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, 21-25 May 2018, Vancouver, BC, Canada. pp. 348–357. IEEE Computer Society (2018), DOI: 10.1109/IPDPS.2018.00044

94. Lu, Z.M., Guo, S.Z.: Chapter 1 - Introduction. In: Lu, Z.M., Guo, S.Z. (eds.) Lossless Information Hiding in Images, pp. 1–68. Syngress (2017), DOI: 10.1016/B978-0-12-812006-4.00001-2

95. Lundborg, M., Apostolov, R., Spångberg, D., et al.: An efficient and extensible format, library, and API for binary trajectory data from molecular simulations. Journal of Computational Chemistry 35(3), 260–269 (2014), DOI: 10.1002/jcc.23495

96. Ma, C., Jung, J., Kim, S., et al.: Random projection-based partial feature extraction for robust face recognition. Neurocomputing 149, 1232–1244 (2015), DOI: 10.1016/j.neucom.2014.09.004

97. van der Maaten, L., Hinton, G.: Visualizing data using t-SNE. Journal of Machine Learning Research 9, 2579–2605 (2008)

98. van der Maaten, L., Postma, E., van den Herik, J.: Dimensionality reduction: a comparative review. Journal of Machine Learning Research 10(66-71), 13 (2009)

99. Magenheimer, D.: In-kernel memory compression. https://lwn.net/Articles/545244/ (2013), accessed: 2020-02-20

100. Mahoney, M.: Data Compression Explained. http://mattmahoney.net/dc/dce.html#Section_524 (2013), accessed: 2020-02-20

101. Marsaglia, N., Li, S., Belcher, K., et al.: Dynamic I/O Budget Reallocation For In Situ Wavelet Compression. In: Childs, H., Frey, S. (eds.) Eurographics Symposium on Parallel Graphics and Visualization, EGPGV 2019, 3-4 June 2019, Porto, Portugal. pp. 1–5. Eurographics Association (2019), DOI: 10.2312/pgv.20191104

102. Martel, E., Lazcano, R., López, J.F., et al.: Implementation of the Principal Component Analysis onto High-Performance Computer Facilities for Hyperspectral Dimensionality Reduction: Results and Comparisons. Remote Sensing 10(6), 864 (2018), DOI: 10.3390/rs10060864

103. Martínez, A.M., Kak, A.C.: PCA versus LDA. IEEE Trans. Pattern Anal. Mach. Intell. 23(2), 228–233 (2001), DOI: 10.1109/34.908974

104. Masek, P., Stusek, M., Krejci, J., et al.: Unleashing Full Potential of Ansible Framework: University Labs Administration. In: 22nd Conference of Open Innovations Association, FRUCT 2018, 15-18 May 2018, Jyväskylä, Finland. pp. 144–150. IEEE (2018), DOI: 10.23919/FRUCT.2018.8468270

105. Matthes, A., Huebl, A., Widera, R., et al.: In situ, steerable, hardware-independent and data-structure agnostic visualization with ISAAC. Supercomputing Frontiers and Innovations 3(4), 30–48 (2016), DOI: 10.14529/jsfi160403

106. McInnes, L., Healy, J., Melville, J.: Umap: Uniform manifold approximation and projection for dimension reduction. CoRR abs/1802.03426 (2018), https://arxiv.org/abs/1802.03426

107. Mecum, B.D., Jones, M.B., Vieglais, D., et al.: Preserving Reproducibility: Provenance and Executable Containers in DataONE Data Packages. In: 14th IEEE International Conference on e-Science, e-Science 2018, 29 Oct.-1 Nov. 2018, Amsterdam, The Netherlands. pp. 45–49. IEEE Computer Society (2018), DOI: 10.1109/eScience.2018.00019

108. Meister, D., Kaiser, J., Brinkmann, A., et al.: A study on data deduplication in HPC storage systems. In: Hollingsworth, J.K. (ed.) SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, 11-15 Nov. 2012, Salt Lake City, UT, USA. p. 7. IEEE/ACM (2012), DOI: 10.1109/SC.2012.14

109. Menegidio, F.B., Jabes, D.L., de Oliveira, R.C., et al.: Dugong: a Docker image, based on Ubuntu Linux, focused on reproducibility and replicability for bioinformatics analyses. Bioinformatics 34(3), 514–515 (2018), DOI: 10.1093/bioinformatics/btx554

110. Mentzer, F., Agustsson, E., Tschannen, M., et al.: Practical Full Resolution Learned Lossless Image Compression. CoRR abs/1811.12817 (2018), http://arxiv.org/abs/1811.12817

111. Moffat, A.: Huffman Coding. ACM Comput. Surv. 52(4), 85:1–85:35 (2019), DOI: 10.1145/3342555

112. Muthitacharoen, A., Chen, B., Mazières, D.: A Low-Bandwidth Network File System. In: Marzullo, K., Satyanarayanan, M. (eds.) Proceedings of the 18th ACM Symposium on Operating System Principles, SOSP 2001, 21-24 Oct. 2001, Chateau Lake Louise, Banff, Alberta, Canada. pp. 174–187. ACM (2001), DOI: 10.1145/502034.502052

113. Norton, A., Clyne, J.P.: The VAPOR Visualization Application. In: Bethel, E.W., Childs, H., Hansen, C.D. (eds.) High Performance Visualization - Enabling Extreme-Scale Scientific Insight. Chapman and Hall / CRC computational science series, CRC Press (2012), DOI: 10.1201/b12985-25

114. Ohtani, H., Hagita, K., Ito, A.M., et al.: Irreversible data compression concepts with polynomial fitting in time-order of particle trajectory for visualization of huge particle system. Journal of Physics: Conference Series 454, 012078 (2013), DOI: 10.1088/1742-6596/454/1/012078

115. Park, J., Park, H., Choi, Y.: Data compression and prediction using machine learning for industrial IoT. In: 2018 International Conference on Information Networking, ICOIN 2018, 10-12 Jan. 2018, Chiang Mai, Thailand. pp. 818–820 (2018), DOI: 10.1109/ICOIN.2018.8343232

116. Plugariu, O., Gegiu, A.D., Petrica, L.: FPGA systolic array GZIP compressor. In: 2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI). pp. 1–6. IEEE (2017), DOI: 10.1109/ECAI.2017.8166387

117. Pörtner, A., Hoffmann, M., Zug, S., et al.: SwarmRob: A Docker-Based Toolkit for Reproducibility and Sharing of Experimental Artifacts in Robotics Research. In: IEEE International Conference on Systems, Man, and Cybernetics, SMC 2018, 7-10 Oct. 2018, Miyazaki, Japan. pp. 325–332. IEEE (2018), DOI: 10.1109/SMC.2018.00065

118. Qiao, Y., Fang, J., Hofstee, H.P.: An FPGA-based Snappy Decompressor-Filter (2018), DOI: 10.13140/RG.2.2.30215.44962

119. Qin, Z., Wang, J., Liu, Q., et al.: Estimating Lossy Compressibility of Scientific Data Using Deep Neural Networks. IEEE Letters of the Computer Society 3(1), 5–8 (2020), DOI: 10.1109/LOCS.2020.2971940

120. Rattanaopas, K., Kaewkeeree, S.: Improving Hadoop MapReduce performance with data compression: A study using wordcount job. In: 2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, ECTI-CON, 27-30 June 2017, Phuket, Thailand. pp. 564–567 (2017), DOI: 10.1109/ECTICon.2017.8096300

121. Rippel, O., Bourdev, L.D.: Real-Time Adaptive Image Compression. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, 6-11 Aug. 2017, Sydney, NSW, Australia. pp. 2922–2930 (2017), http://proceedings.mlr.press/v70/rippel17a.html

122. Rivia, M., Caloria, L., Muscianisia, G., et al.: In-situ Visualization: State-of-the-art and Some Use Cases. http://www.prace-ri.eu/IMG/pdf/In-situ_Visualization_State-of-the-art_and_Some_Use_Cases-2.pdf (2012), accessed: 2020-02-20

123. Röber, N., Engels, J.F.: In-Situ Processing in Climate Science. In: Weiland, M., Juckeland, G., Alam, S.R., et al. (eds.) High Performance Computing - ISC High Performance 2019 International Workshops, 16-20 June 2019, Frankfurt, Germany, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11887, pp. 612–622. Springer (2019), DOI: 10.1007/978-3-030-34356-9_46

124. Rougier, N.P., Hinsen, K., Alexandre, F., et al.: Sustainable computational science: the ReScience initiative. PeerJ Computer Science 3, e142 (2017), DOI: 10.7717/peerj-cs.142

125. Sahinalp, S.C., Rajpoot, N.M.: Chapter 6 - Dictionary-Based Data Compression: An Algorithmic Perspective. In: Sayood, K. (ed.) Lossless Compression Handbook, pp. 153–167. Communications, Networking and Multimedia, Academic Press, San Diego (2003), DOI: 10.1016/B978-012620861-0/50007-3

126. Salomon, D.: Data compression - The Complete Reference, 4th Edition. Springer (2007)

127. Samanta, R., Mahapatra, R.: An Enhanced CAM Architecture to Accelerate LZW Compression Algorithm. In: 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems, VLSID'07, 6-10 Jan. 2007, Bangalore, India. pp. 824–829. IEEE (2007), DOI: 10.1109/VLSID.2007.34

128. Sasaki, N., Sato, K., Endo, T., et al.: Exploration of Lossy Compression for Application-Level Checkpoint/Restart. In: 2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, 25-29 May 2015, Hyderabad, India. pp. 914–922. IEEE Computer Society (2015), DOI: 10.1109/IPDPS.2015.67

129. Schendel, E.R., Jin, Y., Shah, N., et al.: ISOBAR Preconditioner for Effective and High-throughput Lossless Data Compression. In: Kementsietsidis, A., Salles, M.A.V. (eds.) IEEE 28th International Conference on Data Engineering, ICDE 2012, 1-5 April 2012, Washington, DC, USA. pp. 138–149. IEEE Computer Society (2012), DOI: 10.1109/ICDE.2012.114

130. Setia, A., Ahlawat, P.: Enhanced LZW Algorithm with Less Compression Ratio. In: Proceedings of Int. Conf. on Advances in Computing. pp. 347–351. Springer India, New Delhi (2012), DOI: 10.1007/978-81-322-0740-5_41

131. Shadura, O., Bockelman, B.P.: ROOT I/O compression algorithms and their performance impact within run 3. CoRR abs/1906.04624 (2019), `http://arxiv.org/abs/1906.04624`

132. Shanmugasundaram, S., Lourdusamy, R.: A Comparative Study Of Text Compression Algorithms. ICTACT Journal on Communication Technology 1(3), 68–76 (2011), DOI: 10.21917/ijct.2011.0062

133. Shehabi, A., Smith, S., Sartor, D., et al.: United States Data Center Energy Usage Report (2016), DOI: 10.2172/1372902

134. Shibata, Y., Kida, T., Fukamachi, S., et al.: Byte Pair Encoding: A Text Compression Scheme That Accelerates Pattern Matching (1999), `https://pdfs.semanticscholar.org/1e94/41bbad598e181896349757b82af42b6a6902.pdf`

135. Shudler, S., Ferrier, N.J., Insley, J.A., et al.: Spack meets singularity: creating movable in-situ analysis stacks with ease. In: Moreland, K., Garth, C., Bethel, E.W., et al. (eds.) Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV@SC 2019, 18 Nov. 2019, Denver, Colorado, USA. pp. 34–38. ACM (2019), DOI: 10.1145/3364228.3364682

136. Silver, J., Zender, C.: The compressionerror trade-off for large gridded data sets. Geoscientific Model Development 10, 413–423 (2017), DOI: 10.5194/gmd-10-413-2017

137. Simone, S.D.: Apple Open-Sources its New Compression Algorithm LZFSE (2016), https://www.infoq.com/news/2016/07/apple-lzfse-lossless-opensource/, accessed: 2020-02-20

138. Singhal, S., Sussman, A.: Adaptive Compression to Improve I/O Performance for Climate Simulations. https://web.njit.edu/~qliu/assets/adaptive-compression-scheme(acomps).pdf (2017), accessed: 2020-02-17

139. Srinivasan, R., Rao, K.R.: Predictive Coding Based on Efficient Motion Estimation. IEEE Trans. Communications 33(8), 888–896 (1985), DOI: 10.1109/TCOM.1985.1096398

140. Szorc, G.: Better Compression with Zstandard. https://gregoryszorc.com/blog/2017/03/07/better-compression-with-zstandard (2017), accessed: 2020-02-17

141. Tahghighi, M., Mousavi, M., Khadivi, P.: Hardware implementation of a novel adaptive version of Deflate compression algorithm. In: 2010 18th Iranian Conference on Electrical Engineering, 11-13 May 2010, Isfahan, Iran. pp. 566–569. IEEE (2010), DOI: 10.1109/IRANIANCEE.2010.5507007

142. Tajul, T.K., Bhuiyan, S.R., Habib, A.: Enhancement of LZAP (Lempel Ziv All Prefixes) Compression Algorithm. In: 2018 4th International Conference on Electrical Engineering and Information Communication Technology, iCEEiCT. pp. 69–73 (2018), DOI: 10.1109/CEEICT.2018.8628148

143. Tao, D., Di, S., Chen, Z., et al.: Significantly Improving Lossy Compression for Scientific Data Sets Based on Multidimensional Prediction and Error-Controlled Quantization. CoRR abs/1706.03791 (2017), http://arxiv.org/abs/1706.03791

144. Tao, D., Di, S., Guo, H., et al.: Z-checker: A framework for assessing lossy compression of scientific data. IJHPCA 33(2) (2019), DOI: 10.1177/1094342017737147

145. Tao, D., Di, S., Liang, X., et al.: Optimizing Lossy Compression Rate-Distortion from Automatic Online Selection between SZ and ZFP. IEEE Trans. Parallel Distrib. Syst. 30(8), 1857–1871 (2019), DOI: 10.1109/TPDS.2019.2894404

146. Tenenbaum, J.B., Silva, V.D., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science 290(5500), 2319–2323 (2000), https://science.sciencemag.org/content/sci/290/5500/2319.full.pdf

147. Toderici, G., Vincent, D., Johnston, N., et al.: Full Resolution Image Compression with Recurrent Neural Networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 21-26 July 2017, Honolulu, HI, USA. pp. 5435–5443 (2017), DOI: 10.1109/CVPR.2017.577

148. Underwood, R., Di, S., Calhoun, J.C., et al.: FRaZ: A Generic High-Fidelity Fixed-Ratio Lossy Compression Framework for Scientific Floating-point Data. CoRR abs/2001.06139 (2020), https://arxiv.org/abs/2001.06139

149. Vetterli, M., Kovacevic, J.: Wavelets and Subband Coding. Prentice Hall Signal Processing Series, Prentice Hall (1995)

150. Visualization and Analysis Software Team: VAPOR product roadmap. Tech. rep., NCAR (2017), https://ncar.github.io/vapor2website/sites/default/files/VAPORRoadmap.pdf

151. Welch, T.A.: A Technique for High-Performance Data Compression. IEEE Computer 17(6), 8–19 (1984), DOI: 10.1109/MC.1984.1659158

152. Welton, B., Kimpe, D., Cope, J., et al.: Improving I/O Forwarding Throughput with Data Compression. In: 2011 IEEE International Conference on Cluster Computing, CLUSTER, 26-30 Sept. 2011, Austin, TX, USA. pp. 438–445. IEEE Computer Society (2011), DOI: 10.1109/CLUSTER.2011.80

153. Whitlock, B., Favre, J.M., Meredith, J.S.: Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In: Kuhlen, T.W., Pajarola, R., Zhou, K. (eds.) Eurographics Symposium on Parallel Graphics and Visualization, EGPGV 2011, Llandudno, Wales, UK. Proceedings. pp. 101–109. Eurographics Association (2011), DOI: 10.2312/EGPGV/EGPGV11/101-109

154. Widianto, E.D., Prasetijo, A.B., Ghufroni, A.: On the implementation of ZFS (Zettabyte File System) storage system. In: 2016 3rd International Conference on Information Technology, Computer, and Electrical Engineering, ICITACEE. pp. 408–413 (2016), DOI: 10.1109/ICITACEE.2016.7892481

155. Williams, R.N.: An Extremely Fast Ziv-Lempel Data Compression Algorithm. In: Storer, J.A., Reif, J.H. (eds.) Proceedings of the IEEE Data Compression Conference, DCC 1991, 8-11 April 1991, Snowbird, Utah, USA. pp. 362–371. IEEE Computer Society (1991), DOI: 10.1109/DCC.1991.213344

156. Xia, W., Jiang, H., Feng, D., et al.: A Comprehensive Study of the Past, Present, and Future of Data Deduplication. Proceedings of the IEEE 104(9), 1681–1710 (2016), DOI: 10.1109/JPROC.2016.2571298

157. Xie, H., Li, J., Xue, H.: A survey of dimensionality reduction techniques based on random projection. CoRR abs/1706.04371 (2017), http://arxiv.org/abs/1706.04371

158. Yamada, M., Jitkrittum, W., Sigal, L., et al.: High-Dimensional Feature Selection by Feature-Wise Kernelized Lasso. Neural Computation 26(1), 185–207 (2014), DOI: 10.1162/NECO_a_00537

159. Zender, C.S.: Bit Grooming: statistically accurate precision-preserving quantization with compression, evaluated in the netCDF Operators (NCO, v4.4.8+). Geoscientific Model Development 9(9), 3199–3211 (2016), DOI: 10.5194/gmd-9-3199-2016

160. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Information Theory 23(3), 337–343 (1977), DOI: 10.1109/TIT.1977.1055714

DOI: 10.14529/jsfi200102

# Development of Computational Pipeline Software for Genome/Exome Analysis on the K computer

*Kento Aoyama*[1,2], *Masanori Kakuta*[1], *Yuri Matsuzaki*[1], *Takashi Ishida*[1], *Masahito Ohue*[1], *Yutaka Akiyama*[1]



Pipeline software that comprise tool and application chains for specific data processing have found extensive utilization in the analysis of several data types, such as genome, in bioinformatics research. Recent trends in genome analysis require use of pipeline software for optimum utilization of computational resources, thereby facilitating efficient handling of large-scale biological data accumulated on a daily basis. However, use of pipeline software in bioinformatics tends to be problematic owing to their large memory and storage capacity requirements, increasing number of job submissions, and a wide range of software dependencies. This paper presents a massive parallel genome/exome analysis pipeline software that addresses these difficulties. Additionally, it can be executed on a large number of K computer nodes. The proposed pipeline incorporates workflow management functionality that performs effectively when considering the task-dependency graph of internal executions via extension of the dynamic task distribution framework. Performance results pertaining to the core pipeline functionality, obtained via evaluation experiments performed using an actual exome dataset, demonstrate good scalability when using over a thousand nodes. Additionally, this study proposes several approaches to resolve performance bottlenecks of a pipeline by considering the domain knowledge pertaining to internal pipeline executions as a major challenge facing pipeline parallelization.

*Keywords: pipeline software, software development, K computer, message passing interface, genome analysis, exome analysis.*

## Introduction

The cost of analyzing the enormous amounts of life science data that accumulate on a daily basis has become a major research bottleneck because the appearance of next-generation DNA sequencers (NGS) has drastically improved the efficiency of sequencing. Today, pipeline software that automates a series of analytic tasks on a computer and provides functions that utilize computational resources effectively is indispensable for research work. In fact, data analysis using pipeline software on parallel computers is being performed daily at various research sites [1–3].

The "K computer" [4], located at RIKEN Advanced Institute for Computational Science, is representative of large-scale parallel computers in Japan and has a track record of winning the 9-th consecutive first place in the Graph500.org contest. Likewise, the K computer is expected to achieve exceptional results when performing bioinformatics analysis. However, to date, results obtained in this regard using the K computer are limited by several challenges that must be overcome.

First, several large-scale genome analysis requires large memory and storage capacity; however, the specification of the K computer is that each computing node has a small memory capacity. In addition, pipeline software tend to assume that a large number of jobs can be run during internal execution. However, such an operation is not conducive for efficient functioning of the K computer. The above-mentioned problems can be attributed to the architectural con-

[1]Department of Computer Science, School of Computing, Tokyo Institute of Technology, 152-8550, Tokyo, Japan
[2]AIST-Tokyo Tech Real World Big-Data Computation Open Innovative Laboratory (RWBC-OIL), National Institute of Advanced Industrial Science and Technology (AIST), Japan

cept of the K computer aimed at realization of massive parallel processing through use of a large number of lightweight nodes connected by a high-speed network.

Secondly, implementation of a general bioinformatics pipeline software on the K computer requires different software that are coded in different programming languages and different libraries to be modified to suit the SPARC architecture of the K computer. For instance, programming language environments, such as Java, are necessary to execute the software required in certain pipelines, but the same may not be officially provided within the K computer system. Therefore, the cost of developing an environment for bioinformatics analysis on the K computer can be considered a major challenge.

Exome analysis, the focus of this study, is one of the bioinformatics analysis methods that require a large-scale computing environment. In particular, exome analysis is a technique based on genome sequence analysis and is designed to efficiently detect functionally important mutations by analyzing only the "exon regions" in the whole human genome [5]. In exome sequencing, because the exon sequencing cost per sample is lower than general whole genome sequencing, a large number of samples is expected to be processed; thus, the computational resources are required to be more powerful. The acceleration of exome analysis to enable us to process hundreds of samples urgently requires the development of an efficient analysis pipeline on a large-scale computer system.

This motivated us to develop pipeline software that performs exome analysis on the K computer and an extension of the task management framework for this research. We consider the affinity of our task management system and job management system of the K computer to enhance the pipeline performance.

To summarize, this study makes the following key contributions:

- We developed pipeline software that performs exome analysis on the K computer to satisfy the demands of the vast amount of exome sequencing data.
- We developed an extension of the task management framework to enable effective execution by considering the task dependencies, and to run on the K computer.
- We propose approaches to resolve the bottleneck of parallel performance of the pipeline by re-arranging the internal process.

The remainder of this manuscript has been compiled as follows.

Section 1 discusses core implementations of the proposed pipeline software to be used on the K computer. The section provides an overview of the K computer, design and implementation of the proposed pipeline software. Additionally, performance results obtained for a core function – i.e., mapping for reference genome – of the said software have been discussed.

Section 2 describes pipeline parallelization based on an execution profile, thereby demonstrating means to improve pipelining performance. In this section, the authors propose several approaches to resolve performance bottlenecks and discuss the effects of proposed approaches through evaluation experiments.

Limitations of the pipeline and proposed approaches are presented in the Discussion section. Lastly, the Conclusions section summarizes major learnings from this study as well as discusses the scope for further research endeavors in this regard.

# 1. Development of a Genome/Exome Pipeline by Extending a Task Management Framework on the K computer

## 1.1. Overview of the K Computer

The K computer [4] is a large-scale parallel computer consisting of 88,128 nodes and 864 racks in total. Each node is equipped with a Fujitsu general-purpose scalar processor SPARC64 VIIIfx, in which eight processor cores, cache memory, and a memory controller are integrated into one chip. These nodes constitute the six-dimensional mesh/torus network connected by the "Tofu" interconnect [6] to ensure that the network achieves high throughput and high availability. Table 1 shows the system specification of the K computer[3, 4].

**Table 1.** System specification of K computer

| | |
|---|---|
| # compute nodes | 82,944 |
| CPU | SPARC64 VIIIfx [2.0 GHz] (8 cores) |
| Memory | DDR3 SDRAM 16 [GB] |
| Interconnect | Tofu interconnect [6] |
| Topology | 6D mesh/torus |
| Operating System | Linux (customized) |
| File system | Lustre (Fujitsu Exabyte File System) [8, 9] |

### *1.1.1. Two-layer file system model and staging process*

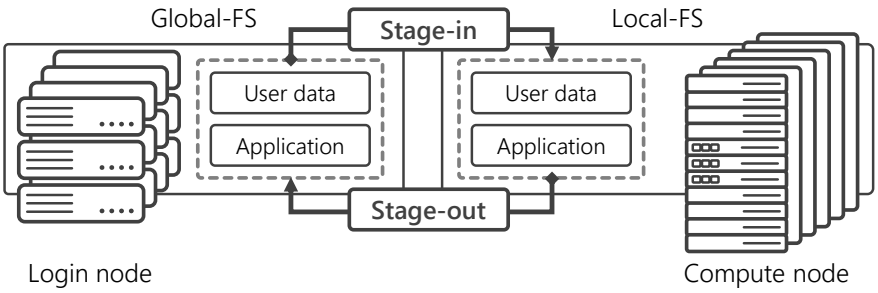



**Figure 1.** Two-layer file system model on K computer

The K computer supports large-scale file access from tens of thousands of nodes by adopting a two-layer file system model (Fig. 1). The first is a Local File System (Local-FS) that is used as a high-speed temporary storage area dedicated to the jobs running on the compute nodes. The second is a Global File System (Global-FS) that is used as a large-capacity shared storage area widely accessed from login node and other related systems. Both of them use the Fujitsu Exabyte File System [9] based on Lustre [8].

Users can access data stored in the large-capacity Global-FS from the login node, whereas when executing a job, the files necessary for running jobs must be transferred to the Local-FS. Specifically, by describing options such as the files path to the script at job submission, the following "staging" process will be started: input target files are copied from Global-FS to

[3] The K computer was operated by RIKEN in Japan between June 2012 and September 2019.
[4] "Fugaku" – a successor system of the K computer equipped with ARM-based CPU (A64FX) – is currently under development, and its operation is scheduled to begin in 2021 [7].

Local-FS before the job execution (stage-in), and after the job completion, the output target files are copied from Local-FS to Global-FS (stage-out).

All users must keep the staging process in mind when submitting a job to the K computer; consequently, the pipeline we developed on this research is also designed to be compatible with the two-layer file system model and to minimize data transfers in the staging process.

## 1.2. Genomon-exome: an Exome Analysis Pipeline Software

We selected "Genomon-exome" as a target for the proposed pipeline software developed for the K computer. Genomon-exome is exome analysis pipeline software designed to run in a general PC cluster environment. It provides various functions composed by using well-known open source software (OSS) in the bioinformatics field, and its entire source code is available on the web [3]. It also supports parallel execution of its pipeline internal processes by submitting jobs into the PC cluster to utilize the computational resources comprehensively. Genomon-exome has contributed to the exome sequencing data analysis in the biological field, and was extensively used to process and analyze data in various studies [1, 2].

### *1.2.1. General genome/exome analysis pipeline software*

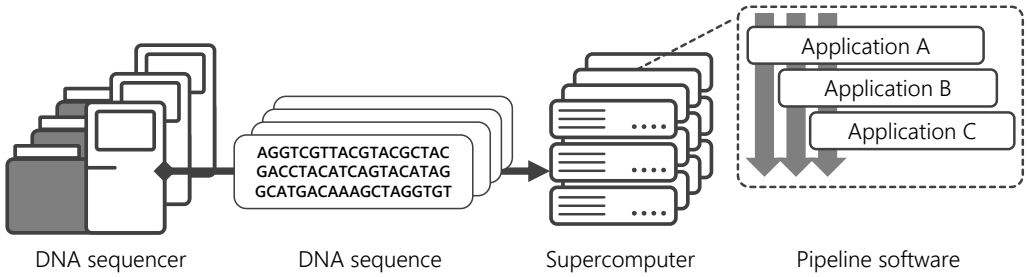



**Figure 2.** Computational experiments in general genome analysis workflow

The computational processes of exome analysis and general genome analysis are almost the same. Pipeline software is used to analyze the DNA sequencing data generated by the DNA sequencers or obtained from public databases to achieve its various purposes. The software is designed to perform in computing environments of various scales ranging from a small-scale PC cluster to a large-scale supercomputer (Fig. 2).

In the case of exome analysis, DNA fragments in a sample corresponding to "exonic regions" are specially hybridized and selected at the previous biological experiments, such that exome data performs effectively in specific analysis processes with metadata of exonic regions and can help to reduce the computational cost.

The following are general functions of a genome/exome analysis pipeline software.

**Mapping (Alignment)** Align the base sequences of several short DNA fragments (so-called "reads") with the positions of a similar sequence in a reference genome.

**Mutation call** Detect mutations by comparing the differences among the aligned sequences from several types of samples.

**Annotation** Annotate each of the mutations to each of the known gene databases to examine the relationship between mutations and genes in samples.

### *1.2.2. Overview of mapping in Genomon-exome*

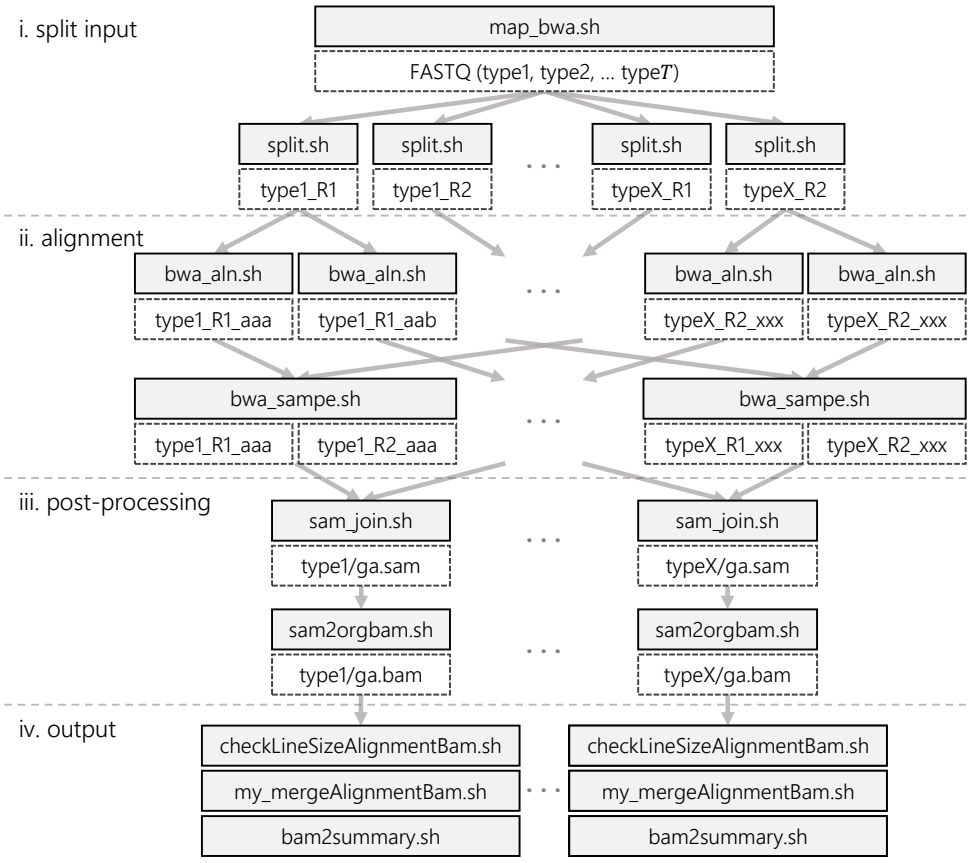



**Figure 3.** Task-dependency graph in mapping

Here, we present an overview of the mapping function of the pipeline in Genomon-exome (Fig. 3). Each task is defined as a shell-script that executes necessary binaries (or commands) to process input data for each purpose. All tasks in the mapping function are executed in parallel, to the extent possible, by submitting jobs to the scheduler.

**i. split input** Split input FASTQ format files based on parameters (`split.sh`), and submit jobs in parallel (`map_bwa.sh`).

**ii. alignment** Do alignment of all reads in files to reference genome using the Burrows-Wheeler Aligner (BWA) [10] (`bwa_aln.sh`), output the Sequence Alignment/Map (SAM) format [11] files as mapping results using the pair-end sequence (`bwa_sampe.sh`).

**iii. post-processing** Merge separated SAM files into one file by each sample (`sam_join.sh`). Convert into a compressed format (BAM), sort the reads based on aligned position, remove duplicated reads, and create an index of reads (`sam2orgbam.sh`) using SAMtools [12].

**iv. output** Check the consistency of output results (`checkLineSizeAlignmentBam.sh`). Merge and create index again if other BAM files from the same sample also exist (`my_mergeAlignmentBam.sh`). Output a statistical summary of the mapping (`bam2summary.sh`).

#### *1.2.3. Other functions within Genomon-exome*

The original Genomon-exome includes other functions such as mutation calling and gene annotation. However, these functions do not necessarily perform efficiently in large-scale parallel processing because their computational cost is much smaller than that of the mapping function. Furthermore, on the K computer, the supported programming language environments in the computing nodes do not meet the full requirements of the pipeline (Tab. 2); thus, it was difficult to develop full functions of the pipeline.

Therefore, we assumed that the remaining functions in the pipeline are executed on the "pre/post-processing node" – a general-purpose computing environment accessible from the login node with support for 'R' and 'Java' languages. We focus on the mapping function and its large-scale parallel performance on the computing node of the K computer in the following sections. Moreover, we first focus on the alignment task on the mapping function (`i. split` and `ii. alignment` in Fig. 3), because that task tends to be a dominant part of the entire mapping function.

**Table 2.** Software requirements of original Genomon-exome and language support in K computer

| Software | Purpose | Language | The K computer |
|---|---|---|---|
| BWA [10] | Alignment for reference sequence | C | ✓ |
| GATK [13] | Realignment for mapping result | Java | |
| SAMtools [12] | Utilities for SAM/BAM format | C | ✓ |
| ANNOVAR [14] | Annotation of result | Perl | ✓ |
| Picard [15] | Statistical output | Java | |
| cutadapt [16] | Removing adapter sequences | Python | ✓ |
| maq [17] | Conversion for Solexa/Sanger format | C/C++ | ✓ |
| bedtools [18] | Utilities for BED format | C/C++ | ✓ |
| bioconductor [19] | Copy Number Variation (CNV) analysis | R | |

## 1.3. Design and Implementation of Proposed Pipeline on K Computer System

Two major difficulties need to be solved to develop a pipeline on the K computer: those related to the pipeline task management and those related to the job management system of the K computer. The first problem is the increasing staging time due to an increase in the number of input jobs. The second problem is the increasing waiting time by unnecessary synchronization.

*Problem 1: Increasing the staging time in job queuing*

In the original Genomon-exome, one job was responsible for managing the entire workflow and progress of the other jobs (`map_bwa.sh` in Fig. 3). Once submitted by a user, a "management job" submits each pipeline task as a child job to the scheduler. It additionally waits for the synchronization of jobs of the same task, submits the next task as a job, and repeats it until the workflow reaches completion.

However, every job submission on the K computer incurs file transfer (staging) cost because of the differences of the file systems between Local-FS and Global-FS. Thus, an increase in the staging time of each job in a pipeline was a matter of pipeline development.

Here we present an example of the number of tasks used in the original Genomon-exome to show the maximum concurrent tasks in the mapping function (Tab. 3). In Tab. 3, $T$ is the number of samples, $t_i$ is the $i$-th sample, $S_{t_i}$ is the number of split files of the $i$-th sample.

During the mapping function, the alignment task (`bwa_aln.sh`) can be performed effectively in parallel by increasing the number of split files ($S_{t_i}$) because it directly increases the number of running tasks simultaneously. However, increasing the number of job submissions also incurs increased waiting time for resource allocation and the staging process, thus requiring the problem to be resolved by appropriate task management on the K computer.

**Table 3.** Example of number of tasks in mapping function for pair-end sequence

| Task name | Description | Concurrent tasks |
|---|---|---|
| `map_bwa.sh` | Management job | 1 |
| `split.sh` | Split input | $2T$ |
| `bwa_aln.sh` | Alignment | $2\sum_T S_{t_i}$ |
| `bwa_sampe.sh` | | $\sum_T S_{t_i}$ |
| `sam_join.sh` | Post-processing | $T$ |
| `sam2orgbam.sh` | | $T$ |
| `checkLineSizeAlignmentBam.sh` | Output results | $T$ |
| `my_mergeAlignmentBam.sh` | and statistics | $T$ |
| `bam2summary.sh` | | $T$ |

*Problem 2: Increasing the waiting time by unnecessary synchronization*

In the original pipeline, the "management job", which controls the workflow, always synchronized all jobs within a given task before proceeding to the next task. This required all tasks to wait for completion of the task with the longest execution time, albeit target data were independent of those involved the other tasks. This suboptimum task balance led to performance degradation. Therefore, it is important to resolve inter-task dependencies as well as determine the necessity for task synchronization.

### *1.3.1. Introducing master–worker framework for task management*

We introduce the MPIDP framework [20], a dynamic task-processing framework developed by the authors, as an alternative to the "management job" role in the original pipeline. MPIDP is based on the master-worker model using the MPI library, which dynamically distributes the tasks in the given list with the point-to-point MPI communications. While it is processing tasks, the master sends a new task to an available free worker, and workers execute the assigned task or request the next task, until all tasks have been completed. The data exchanges between processes are performed via the file system.

Introducing MPIDP enables one job to manage the entire task workflow and reduce the number of jobs to resolve the problem of staging time related to the increasing number of jobs.

Since then, the MapReduce framework [21], which performs well by utilizing the affinity of file system access on the K computer, became available. However, it was not available at the time of the beginning of this research, and its implementation design was developed with reference to the source code of MPIDP. Therefore, in this research, we adopted MPIDP as a task-processing framework for the pipeline on the K computer.

### *1.3.2. Developing extension to resolve task dependencies*

We also developed a new extension of MPIDP to solve another problem of the original pipeline related to unnecessary synchronization. The original MPIDP does not have the function to resolve task dependencies because it was developed for the embarrassingly parallel data-processing model. However, because each task in the pipeline depends on each target data that are not independent, the original MPIDP cannot be applied straightforwardly.

To resolve this issue, we developed an extension that enables tasks to be executed by considering data dependencies. The extension provides a function that allows a task to wait only for the task it directly depends on, and awards them executable status when the dependent tasks are completed.

The extension is implemented by the simple wait queuing program to resolve the directed acyclic graph (DAG) of the task-dependency graph. Users can simply use the extension by adding the `id` field of dependent tasks to the line in the task list; consequently, the next task is executed after waiting for the completion of dependent tasks without unnecessary synchronization.

### *1.3.3. Building a pipeline with task management on the K computer*

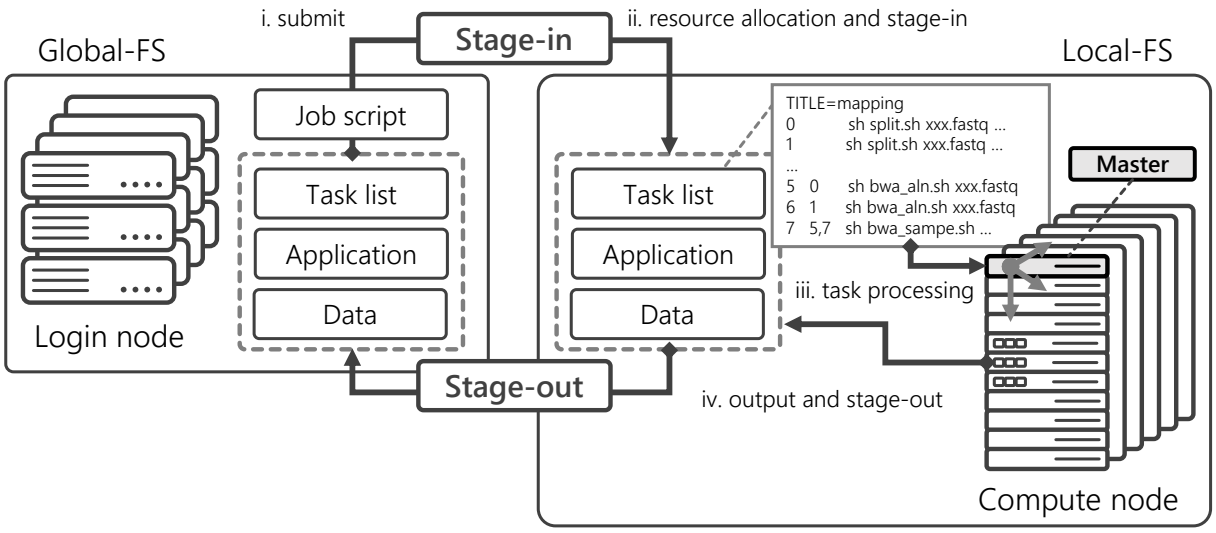



**Figure 4.** Diagram of exome pipeline on K computer using extended MPIDP

Figure 4 diagrammatically presents the exome pipeline that uses extended MPIDP on the K computer. The applications (e.g., BWA and SAMtools) and data (e.g., FASTQ files) required in the pipeline are stored in the data partition of the Global-FS on the login node.

**i. Submitting a job to the system** The user submits a job script that includes computing resources, staging information, a task list for the MPIDP framework, etc. to the job scheduler. That script is automatically generated after the user inputs the data and parameters.

**ii. Resource allocation and stage-in** The job scheduler allocates computing resources, and starts data transfer of the target files from the Global-FS to the Local-FS.

**iii. Task processing using extended MPIDP** The master process of MPIDP assigns the tasks to workers, and workers start to process the tasks. Tasks are dynamically distributed by MPI communications and performed by considering the task dependencies.

**iv. Output results and stage-out** After the job is completed, the results are transferred from the Local-FS to the Global-FS.

## 1.4. Performance Evaluation

We present a performance experiment to evaluate the exome pipeline on the K computer. The experiment shows the execution time and the scalability of the alignment task of the mapping function using our pipeline.

### *1.4.1. Experimental setup*

The input data we used were data from a real exome project obtained from ERP001575 [22], the lung cancer exome dataset (Tab. 4). We selected samples of normal data and samples containing tumors, such that the number of samples in all experiments is two ($T = 2$).

System and software specifications used in the experiments are provided in Tab. 5. The measured data were from `elapsed time` provided by the system stats, and internal task execution of MPIDP by the `gettimeofday` function. For configuration of the internal execution, the number of threads was set as 8 in the alignment task using BWA [10], and the others were run by using a single thread.

**Table 4.** Description of lung-cancer data (exome)

| Run | ERR160121 | ERR166339 |
|---|---|---|
| Type | normal | tumor |
| Length | 100 [bp] | 100 [bp] |
| Paired | yes | yes |
| Platform | Illumina HiSeq 2000 | Illumina HiSeq 2000 |
| Total size | 13.324 [GB]×2 (pair-end) | 23.726 [GB]×2 (pair-end) |

### *1.4.2. Experiment 1: alignment performance of mapping function*

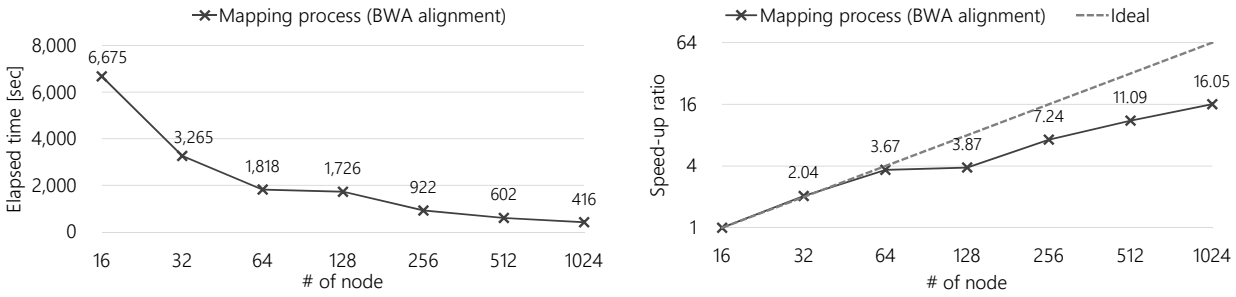
The execution time of the alignment task in the mapping function (`map_bwa.sh`) is shown in Fig. 5a, the scalability in strong scaling is shown in Fig. 5b.

First, it was confirmed that the output of the proposed pipeline was equivalent to the result obtained using the original Genomon-exome environment on a regular PC cluster. This, in turn, implies successful development of a genome/exome analysis pipeline software on the K computer system.

In view of the acceleration of the alignment task in the mapping function, our pipeline provided excellent scalability until the number of nodes was increased to 64 nodes, and also

**Table 5.** System and software settings pertaining to the K computer (Dec. 2014)

| | |
|---|---|
| # compute nodes | 82,944 |
| CPU | SPARC64 VIIIfx [2.0 GHz] (8 cores) |
| Memory | DDR3 SDRAM 16 [GB] |
| Operating System | Linux ver. 2.6.32 |
| C/C++ | Fujitsu C Compiler ver. 1.2.0 |
| Python | Python ver. 2.6.2 |
| Java | N/A |
| Perl | Perl ver. 5.10 |
| MPI | FUJITSU MPI Library (OpenMPI ver. 1.4.3) |




(a) Execution time (b) Scalability in strong scaling

**Figure 5.** Alignment performance of mapping function (`map_bwa.sh`)

performed well but that gradually slowed down. Finally, its speed-up was ×16.05 at 1024 nodes (based on 16 nodes) on the K computer.

On the other hand, because of the issue of total memory capacity of the compute node, our pipeline could not run with less than 16 nodes. It is because the BWA, which is software mainly used in the alignment task, consumes a large amount of memory, thereby causing an internal error in the program.

## 2. Further Challenges Facing Whole-Pipeline Parallelization on K Computer

In this study, we successfully developed an exome pipeline on the K computer. The said pipeline can dynamically distribute tasks within a given job whilst efficiently eliminating unnecessary synchronizations (Sec. 1.3). In this section, to improve the performance of the entire pipeline, we extend the proposed pipeline parallelization to latter parts of the mapping function with the objective of improving performance based on an execution profile.

To this end, this section first demonstrates the execution profile of the proposed pipeline followed by revelation of performance bottlenecks (Sec. 2.1). Subsequently, the handling of each task has been described (Sec. 2.2). Next, the authors propose several approaches to resolve identified pipeline bottlenecks (Sec. 2.3). Lastly, effects of proposed approaches on pipeline performance have been examined (Sec. 2.4).

## 2.1. Performance Analysis of the Mapping Function

To clearly demonstrate the performance bottleneck of the pipeline, we obtained an execution profile of the pipeline for real exome data and estimated the total amount of execution time for each task. Table 6 lists the total execution time of each task of the mapping function. The result in each line is the summation of the execution time of 16 computing nodes. The table data include an amount to correct the measured overhead to compensate for the duplicated file I/O overhead.

**Table 6.** Execution time of each task and its ratio

| Task name | Description | Elapsed time [sec] | ratio |
|---|---|---|---|
| `map_bwa.sh` | Management Job | - | - |
| `split.sh` | Split input | 683 | 0.005 |
| `bwa_aln.sh` | Alignment | 59,876 | 0.448 |
| `bwa_sampe.sh` | | 29,555 | 0.221 |
| `sam_join.sh` | Post-processing | 1,920 | 0.014 |
| `sam2orgbam.sh` | | 39,419 | 0.295 |
| `checkLineSizeAlignmentBam.sh` | Output results | 3,081 | 0.023 |
| `my_mergeAlignmentBam.sh` | and statistics | 125 | 0.001 |
| `bam2summary.sh` | | - | - |
| TOTAL | | 127,094 | 1.00 |

The results show, as we expected, the most dominant task of the mapping function was `bwa_aln.sh` (44.8%) as the main task of alignment. However, the second most time-consuming task was `sam2orgbam.sh` (29.5%) which is the post-processing of alignment. The task `sam2orgbam.sh`, called to perform post-process alignment, shared approximately 30% of all execution time but the degree of concurrency is not as high as that of other tasks. This suggests that only a number of samples are running simultaneously, i.e., two ($T = 2$) in this case.

Consequently, attempts to improve the performance and increase the number of nodes would be affected by this parallel bottleneck, which would detrimentally affect the parallel performance of the pipeline. Thus, we propose several approaches to solve this problem and improve the parallel performance of the pipeline.

## 2.2. Overview of the Alignment-post-process

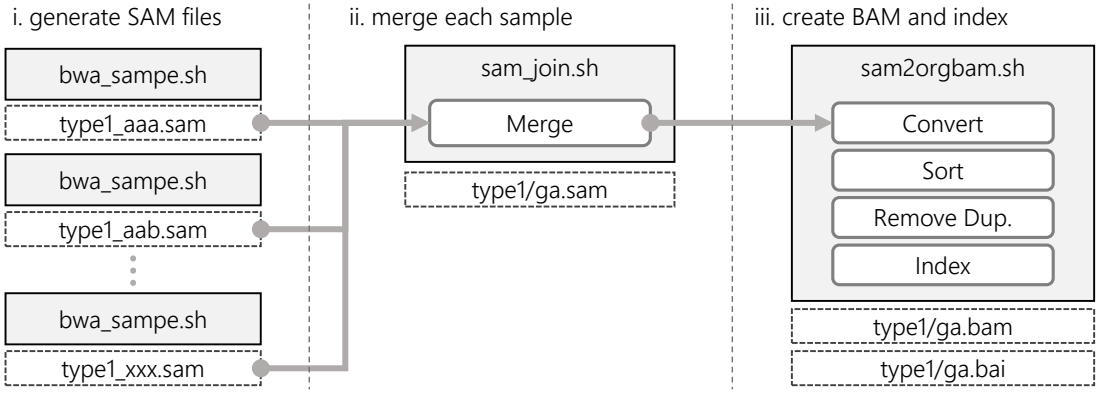



**Figure 6.** Diagram of the alignment-post-process

The alignment-post-process is composed of several tasks such as that of merging the alignment result files for each sample, and converting them into a compressed format (Fig. 6).

First, split SAM format [11] files are generated as the alignment result using "pair-end" information. Next, all SAM files are merged into a single SAM file for each sample. After that, each merged SAM file is converted into compressed BAM format [11], sorted based on the result of the alignment position, the duplicated reads in a file removed, and finally a BAM index created.

The Sequence Alignment/Map (SAM) format stores the result of mapping, including alignment information of reads described in human-readable text. The BAM (binary SAM) format is a compressed SAM file intended to reduce the file size, and is compressed by BGZF (Blocked GNU Zip Format) of GZIP. These two formats are compatible and they can be converted to each other by various tools without information loss.

## 2.3. Proposed Approaches to Improve the Performance of the Alignment-post-process

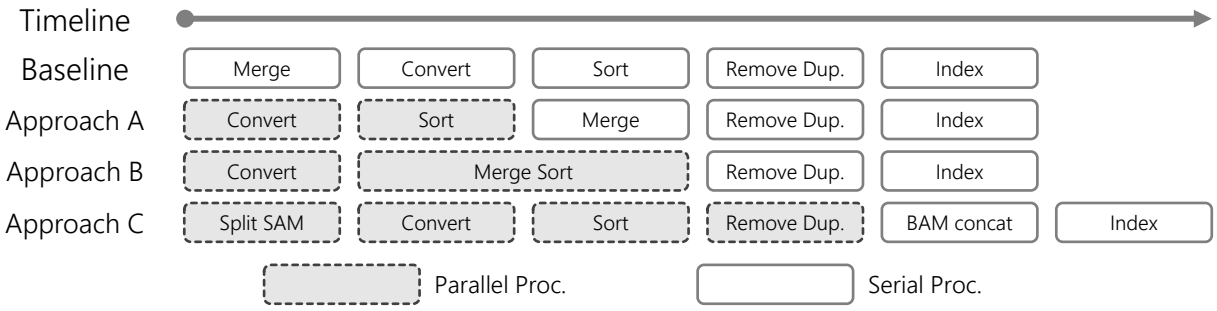



**Figure 7.** Overview of the proposed approaches

We propose 3+1 approaches to improve the parallel performance of the alignment-post-process. The first three approaches are mutually exclusive, but the last approach can be used with other approaches. No approach results in information loss, and results obtained were found to be compatible to those obtained using the baseline approach. Figure 7 shows an overview of the first three approaches.

*Approach A) Rearrangement of processing order*

In the alignment-post-process, the conversion of separated SAM format to BAM format can be executed independently from each other, and apart from this, the separated BAM files can be merged into one BAM file if they are sorted. Accordingly, it is possible to execute both the conversion and sorting processes in parallel prior to merging files. Subsequently, all sorted BAM files can be merged into a single BAM file (Approach A in Fig. 7).

By contrast, the "remove duplicates" process cannot be performed properly in parallel by using this approach. This is because the possibility of the existence of "duplicated reads" over other files would not allow the process to exclude the duplicates between files.

*Approach B) Introduction of parallel merge sort*

Next, we propose the introduction of a parallel sorting algorithm for the sorting and merging process of approach A. We can obtain one sorted and merged BAM file by $O(\log_2 N)$ steps by

using two-way merge sort (Approach B in Fig. 7), where $N$ is the number of separated files. If the sorting process dominates the execution time of alignment-post-process, it is expected to be accelerated using parallel merge sort.

*Approach C) Processing individual chromosomes in parallel*

Here we propose a method to parallelize the "remove duplicate" process. Once a read operation is aligned to a chromosome it cannot be aligned to another chromosome; thus, by dividing the input SAM files among each of the chromosomes, the process of removing the duplicates from each file can be executed independently from each other.

First, the script parses the read operation of the aligned chromosome and splits it into tiny SAM files. Second, the script gathers and merges these tiny SAM files with each of the SAM files aligned to the same chromosome. Then, the processes of converting, sorting, and removing duplicates can be performed in parallel (Approach C in Fig. 7).

Moreover, if all regions are sorted by each chromosome, it is possible to simplify the concatenation in the last merge process without internal sorting.

*Additional) Tuning of BAM compression level*

The file size of BAM format can be tuned by changing its "compression level" — similarly to gzip [23] compression level setting — and a trade-off exists between the file size and compression/decompression time. Note that, if the compression level is configured as high, the execution time of processes using BAM format increases because its internal program includes converting processes from BAM to SAM, and inverse.

Therefore, we configured the compression level to zero, which implies no compression of the internal output of the pipeline, to optimize performance for all proposed approaches. Finally, it set to the default value at the terminal output, to reduce the size of output file. There are still internal conversion processes of SAM/BAM format but the performance is expected to be improved.

## 2.4. Performance Evaluation

We present a performance experiment to evaluate the effectiveness of our proposals from the viewpoint of resolving the bottleneck of parallel performance. We apply each proposed approach to the pipeline and measure the execution time of the entire mapping process for each approach.

### *2.4.1. Experimental setup*

The computational environment that was used is the same as in the previous experiment.

### *2.4.2. Experiment 2: performance of the entire mapping function by applying the proposed approaches*

The execution time of the entire mapping function when applying the proposed approaches (Sec. 2.3) is shown in Fig. 8. The label `baseline (vanilla)` shows the naive implementation of the pipeline we developed in the previous section (Sec. 1.3). The label `baseline (tuned)`

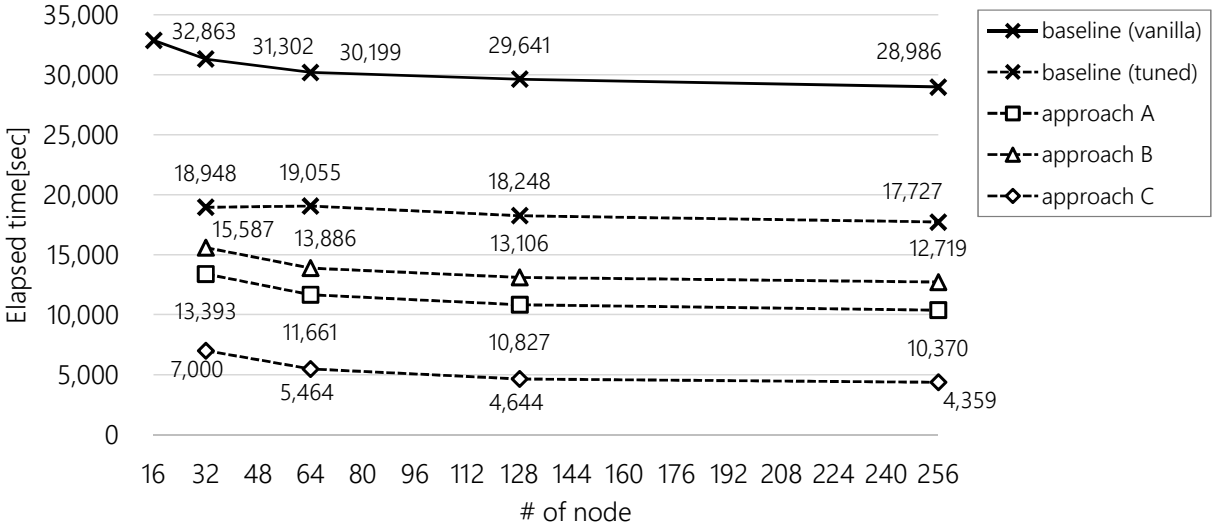



**Figure 8.** Execution time of the entire mapping function by applying the proposed approaches

shows the result of tuning the BAM compression level. The other labels of `approach A, B, C` show the result of applying each of the proposed approaches to the `baseline (tuned)`.

First, we confirmed that the result of tuning the BAM compression level is effective in all cases with an acceptable trade-off. Importantly, it reduces the bottleneck of internal format conversions in the alignment-post-process; therefore, we assumed that the tuning would also be effective in the other approaches.

Our experiments confirmed that `approach C`, with parallelizing for each chromosome, delivers the most effective performance improvement of all. `approach C`, in which almost all processes of the alignment-post-process are performed in parallel, yielded good results, and simple BAM concatenation also performed well because it reduces the time for internal BAM conversion. The other approaches also outperformed the `baseline (tuned)`. That is, `approach A, B` also showed good results but they were not comparable with the result obtained with `approach C`.

However, no results were obtained in cases of executions involving a small number of nodes ($N = 16$). This was attributed to occurrence of the segmentation-fault error during the alignment process when generating the result of alignment from using "pair-end" information. Investigation of what led to occurrence of the said error yielded no concrete results.

## 3. Discussion

In the last experiment, we confirmed our proposed approaches performed effective to reduce the execution time; however, even when we used the most effective of all the proposed approaches, the scalability was not as good as we expected.

*Limitation of the parallelization of chromosomes*

First, the problem directly related to the proposed method (approach C) is suggested to be the limitation by the number of types of human chromosomes. The number of types of human chromosomes is $22 + 2$ at maximum; hence, the maximum parallelism that can be achieved is limited by this number. Consequently, the use of more than $T \times 24$ nodes does not result in optimal performance during the alignment-post-process in approach C. To overcome the

problem, we consider splitting the SAM files into a larger number of small regions, rather than depending on the number of chromosomes to improve the performance.

*Task load balancing between worker nodes*

Additionally, to confirm the stats of load balancing, we calculated the average idle time in experiment 2 (Fig. 9). The average idle time increases with the number of nodes and approximately 56.5% of the execution time is also included in the idle time even for the most satisfactory case at 32 nodes on approach C. Therefore, our proposed approaches show certain improvements in parallel performance; however, they remain problematic with respect to concurrency.

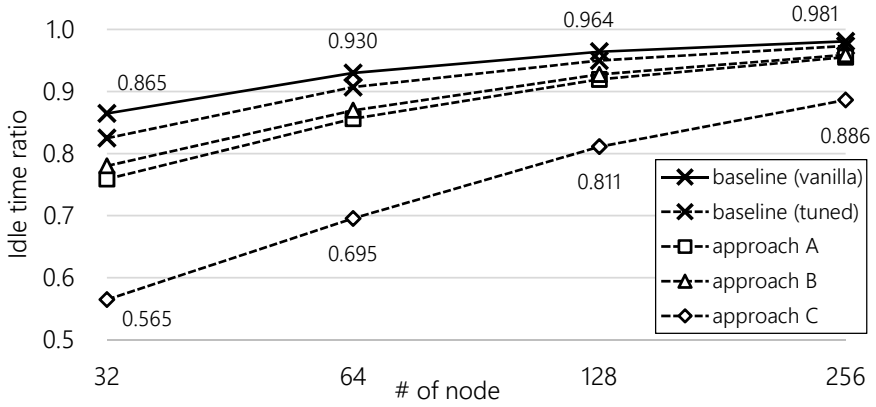



**Figure 9.** Average idle time ratio of each node during the mapping function

## 4. Related Work

Container-based virtualization that generates less performance overhead, such as the container systems Docker, Singularity [24], is spreading into the cloud and HPC environments. Benchmarks and system reports of container technology have been increasing in bioinformatics, especially in pipeline deployment, because the reproducibility and availability of a pipeline would be improved by container technology as application-level virtualization. Although container compatible pipelines (e.g., Nextflow [25]) are spreading in the bioinformatics field, the K computer does not provide such a container environment.

NGS analyzer [26] is a whole-genome analysis pipeline developed at RIKEN that can be implemented in HPC environments such as the K computer. We used its implementations as a reference for our pipeline development but the purpose of the pipeline is different from ours.

## Conclusions

We developed pipeline software for the massively parallel environment of the K computer and confirmed that the parallel performance in the alignment task of the mapping function shows good scalability when using the real exome dataset. Our pipeline was designed to distribute its internal tasks dynamically in parallel using MPI communications of the MPIDP framework; thereby, we succeeded in reducing the number of jobs in the pipeline, apart from ensuring compatibility with the system environment. We also developed an extension of MPIDP to minimize the task synchronizations by providing a function for considering the task dependencies.

Furthermore, we attempted to extend the pipeline parallelization to the alignment post-process, for which we proposed 3+1 approaches to resolve the performance bottleneck by focusing on the number of concurrent tasks in alignment-post-process. The experimental results showed that our approaches are able to effectively improve the performance of the pipeline, which achieved a maximum speed-up of ×6.6 on 256 nodes, compared with the baseline.

The proposed pipeline has been tailored to the system architecture of the K computer. However, the MPIDP framework extended in this study is based on the general MPI framework, and therefore, it is compatible with regular pipelines and PC clusters. In addition, the proposed pipeline would be expected to contribute to address the vast cost of genome analysis, and to the personalized drug development based on genomic information.

## Code Availability

The source code pertaining to the extended MPIDP framework is available in the author's GitHub repository[5]. The source code of the original Genomon-exome pipeline software [3] has been distributed under the Genomon License[6].

## Acknowledgements

We gratefully acknowledge contributions by the following people who provided us with the Genomon-exome and useful advice: Satoru Miyano, Seiya Imoto, Yuichi Shiraishi, Yoshinori Tamada, Satoshi Ito, and Kenichi Chiba. This research used computational resources of the K computer provided by the RIKEN Advanced Institute for Computational Science through the HPCI System Research project (Project ID: hp130017, hp140230). This research was supported by Core Research for Evolutional Science and Technology (CREST) "Extreme Big Data" (Grant No. JPMJCR1303) from the Japan Science and Technology Agency (JST).



## References

1. Yoshida, K., Yoshizato, T., Shiraishi, Y., et al.: Integrated molecular analysis of clear-cell renal cell carcinoma. Nature Genetics 45(8), 860–867 (2013), DOI: 10.1038/ng.2699

2. Yoshida, K., Sanada, M., Shiraishi, Y., et al.: Frequent pathway mutations of splicing machinery in myelodysplasia. Nature 478(7367), 64–69 (2011), DOI: 10.1038/nature10496

3. Genomon-exome. `http://genomon.hgc.jp/exome/en/`, accessed: 2019-02-20

4. Miyazaki, H., Kusano, Y., Shinjou, N., et al.: Overview of the K computer system. Fujitsu Scientific Technical Journal 48(3), 302–309 (2012)

5. Bamshad, M.J., Ng, S.B., Bigham, A.W., et al.: Exome sequencing as a tool for Mendelian disease gene discovery. Nature Reviews 12(11), 745–755 (2011), DOI: 10.1038/nrg3031

[5] `https://github.com/akiyamalab/mpidp`
[6] `http://genomon.hgc.jp/exome/en/`

6. Ajima, Y., Inoue, T., Hiramoto, S., et al.: The Tofu Interconnect. IEEE Micro 32(1), 21–31 (2012), DOI: 10.1109/MM.2011.98

7. Shimizu T.: Supercomputer "Fugaku". ISC High Performance 2019, 16-20 June 2019, Frankfurt, Germany. (2019)

8. Braam, P.J.: The Lustre Storage Architecture. CoRR abs/1903.01955v1 (2019), http://arxiv.org/abs/1903.01955

9. Sakai, K., Sumimoto, S., Kurokawa, M.: High-performance and highly reliable file system for the K computer. Fujitsu Scientific and Technical Journal 48(3), 302–309 (2012)

10. Li, H., Durbin, R.: Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics 25(14), 1754–1760 (2009), DOI: 10.1093/bioinformatics/btp324

11. Sequence Alignment/Map Format Specification - The SAM/BAM Format Specification Working Group. http://samtools.github.io/hts-specs/, accessed: 2019-02-20

12. Li, H., Handsaker, B., Wysocker, A., et al.: The Sequence Alignment/Map format and SAMtools. Bioinformatics 25(16), 2078–2079 (2009), DOI: 10.1093/bioinformatics/btp352

13. McKenna, A., Hanna M., Banks E., et al.: The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. Genome Research 20(9), 1297–1303 (2010), DOI: 10.1101/gr.107524.110

14. Wang, K., Li, M., Hakonarson, H.: ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. Nucleic Acids Research 38(16), e164 (2010), DOI: 10.1093/nar/gkq603

15. Picard. http://broadinstitute.github.io/picard, accessed: 2020-01-20

16. Marcel, M.: Cutadapt removes adapter sequences from high-throughput sequencing reads. EMBnet.journal 17(1), 10–12 (2011), DOI: 10.14806/ej.17.1.200

17. Li, H., Ruan, J., Durbin, R.: Mapping short DNA sequencing reads and calling variants using mapping quality scores. Genome Research 18(11), 1851–1858 (2008), DOI: 10.1101/gr.078212.108

18. Quinlan, A.R., Hall, I.M.: BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics 26(6), 841–842 (2010), DOI: 10.1093/bioinformatics/btq033

19. Gentleman, R.C., Carey, V.J., Bates, D.M., et al.: Bioconductor: open software development for computational biology and bioinformatics. Genome Biology 5(10), R80 (2004), DOI: 10.1186/gb-2004-5-10-r80

20. Ohue, M., Shimoda, T., Suzuki, S., et al.: MEGADOCK 4.0: an ultra-high-performance protein-protein docking software for heterogeneous supercomputers. Bioinformatics 30(22), 3281–3283 (2014), DOI: 10.1093/bioinformatics/btu532

21. Matsuda, M., Maruyama, N., Takizawa, S.: K MapReduce: A scalable tool for data-processing and search/ensemble applications on large-scale supercomputers. IEEE International Conference on Cluster Computing 2013, CLUSTER, 23-27 Sep. 2013, Douliu, Taiwan. pp. 1–8. IEEE (2013), DOI: 10.1109/CLUSTER.2013.6702663

22. Seo, J.S., Ju, Y.S., Lee, W.C., et al.: The transcriptional landscape and mutational profile of lung adenocarcinoma. Genome Research 22(11), 2109–2119 (2010), DOI: 10.1101/gr.145144.112

23. Deutsch, P.: "GZIP file format specification version 4.3", RFC Editor (1996), DOI: 10.17487/RFC1952

24. Kurtzer, G.M., Sochat, V., Bauer, M.W.: Singularity: Scientific containers for mobility of compute. PLOS ONE 12(5), 1–20 (2017), DOI: 10.1371/journal.pone.0177459

25. Tommaso, D.P., Chatzou, M., Floden, E.W., et al.: Nextflow enables reproducible computational workflows. Nature Biotechnology 35(4), 316–319 (2017), DOI: 10.1038/nbt.3820

26. NGS analyzer. `http://www.csrp.riken.jp/application_d_e.html`, accessed: 2019-02-20

DOI: 10.14529/jsfi200103

# Supercomputing Technologies as Drive for Development of Enterprise Information Systems and Digital Economy

*Oleg V. Loginovsky*[1], *Alexander L. Shestakov*[1], *Alexander A. Shinkarev*[1]



The article presents an analysis of approaches to the development of enterprise information systems that are in use today. One of the major trends that predetermines the agenda of information technology is the focus on parallel computing of large volumes of data using supercomputing technologies. The article considers the resulting ubiquitous move to distributed patterns of building enterprise information systems and avoiding monolithic architectures. The emphasis is placed on the importance of such fundamental characteristics of enterprise information systems as reliability, scalability, and maintainability. The article justifies the importance of machine learning in the context of effective big data analysis and competitive gain for business, vital for both maintaining a leading position in the market and surviving in conditions of global instability and digitalization of economy. Transition from storing the current state of a enterprise information system to storing a full log and history of all changes in the event stream is proposed as an instrument of achieving linearization of the data stream for subsequent parallel computing. There is a new view that is being shaped of specialists at the intersection of engineering and analytical disciplines, who would be able to effectively develop scalable systems and algorithms for data processing and integration of its results into company business processes.

*Keywords: enterprise information systems, parallel computing, supercomputing technologies, big data, machine learning, scalability, event stream, analysis, digital economy.*

## Introduction

Today, enterprise information systems experience what can be called a rebirth. Only yesterday monolithic applications were considered the basis of software development, while today unscalable development patterns from the "past" are falling into oblivion and giving way to microservice architectures, which in turn support the further efficient digitalization of the economy.

However, scalability, fault tolerance and possibility to carry out parallel computing across a cluster of computers come at a price of a significant increase in the systems complexity. Deployment, trouble shooting and support of such complex systems are the tasks far from trivial, requiring specialists of higher technical level compared to the development of applications that do not require fault tolerance, process small volumes of data and run on a single machine, namely, on the end user's computer.

For many, microservice patterns for building information systems today is primarily a matter of fashion. However, the need to move to flexible, loosely coupled and scalable systems did not come out of nowhere. The volume and speed of data generation in recent years, as well as the predicted growth of this trend cast doubt on the relevance of the entire data processing on one device. Nowadays, it is impossible to count on the successful processing of the so-called big data without a cluster of computing nodes, parallel algorithms, or involving supercomputer processing power. An important driver for the spread of parallel computing in enterprise applications, in addition to accumulating huge volumes of raw data, is the popularization of data science and machine learning. Business is much more aware of the potential of this field than it was 10 years ago. It allows one to see achievable goals for data analysis, invest money in computing resources and specialists of a new school – data engineers.

[1]South Ural State University, Chelyabinsk, Russian Federation

When developing enterprise applications based on parallel computing, it is more important than ever to pay special attention to such fundamental characteristics as reliability, scalability, and maintainability. There is a growth in complexity with moving to scalable development patterns, which means that one needs to be more scrupulous about the quality of the code, testing, fault tolerance, breaking tasks into subtasks, and linearizing command streams.

Besides developing approaches to parallel enterprise computing, databases also gained much impetus. There is also a move to scalable data warehouses and shift away from distributed transactions for the sake of achieving scalability of computing. Thus, distribution and scalability are fundamental characteristics of modern technologies of enterprise software development.

In machine learning, like in other spheres, there is a move from data processing on a scientist's personal computer to calculations and receiving results from several computing nodes. This is again explained by the volume of data that needs to be processed, as well as by the extent to which a granular computational problem can be broken down into unrelated subtasks.

For a more comprehensive description of the current situation and emerging trends in the development of modern enterprise information systems based on parallel computing, let us consider big data in more detail, as the main reason of a quantum leap in the field of distributed computing and related disciplines.

# 1. Big Data

## 1.1. General Characteristics

The concept of big data has in a sense become too common lately, and its meaning is now quite indistinct.

Here is one of the existing definitions of big data, which is quite generalized, like the term that it describes.

Big data is a scientific and practical field associated with the development and application of methods and tools for operating large volumes of unstructured data [4].

Such an understanding of big data as a phenomenon caused by the development of modern information technologies raises a number of global issues. The first issue is defining the criteria for categorizing data as big. The second issue is the assignment of only unstructured data (schemaless data) to the category in question at this stage of technology, when NoSQL and relational databases, as well as approaches to the storage and indexing, are gradually overlapping [30]. We list the well-known characteristics of big data [7]:

- Volume – How much data is there?
- Variety – How diverse are different types of data?
- Velocity – At what speed is new data generated?
- Veracity – How accurate is the data?

There are several additional characteristics, such as viability, value, variability, and visualization [22].

Here is the list of typical sources of big data [35]:

- Internet of things.
- Social networks.
- Telecommunication satellites.
- Internet stores.
- Internet encyclopedias.

- Various debug logs.

Surely, in addition to the listed above, there are many other sources of big data, and their number will continue to grow.

We list the main types of generated big data [7]:

- Structured – data possessing a specific schema and a fixed set of attributes.
- Unstructured – data without a permanent structure, for example various text documents.
- Natural language – a special kind of unstructured data that includes all texts in all languages of the world.
- Machine-generated – created by a computer, application, or other machine without human intervention.
- Graph-based – the natural representation of which is a graph modeling pairwise relations between objects.
- Streaming – data create in response to the occurrence of an event.
- Audio, video, and images.

Having identified the sources of the big data generators and the types of information they contain, let us consider the data science working process. Here is a description of the process systematizing the work of the analyst [7]:

- Setting the research goal – what you are going to research, how the company benefits from that, what data and resources you need.
- Retrieving data – checking the existence of, quality, and access to the data.
- Data preparation – enhancing the quality and consistency of the data, its normalization and removal of invalid entries.
- Data exploration – analyzing how variables interact with each other, a deeper understanding of the nature of phenomena under study.
- Data modeling – an iterative process of building a model to answer the research question using the insights about the data you found in the previous steps.
- Presentation and automation – using a suitable way to present the work results, automating the execution of the process to update results in case of new source data.

The presented typical process helps to systematically address problems with data that can fit in the memory of a single computer, and answer questions posed to a much larger amount of data, the processing of which may require a whole group of servers (cluster). Scientists and programmers spend considerable amount of time solving the problems of distributed processing of large data sets and teaching models on them.

Bearing in mind the volume of data generated in the world, the importance of solving the problem of efficient distributed data processing cannot be underestimated. And this volume is truly impressive. Google handles 5.5 billion searches per day [23], the number of Internet-connected devices according to various estimates could reach 22–50 billion by 2020 [35] and 75 billion by 2025 [39], Instagram users create 95 million posts per day [8].

Yet, data volume alone cannot give a business a competitive advantage. Only correctly putting questions to this data, as well as quickly making the most of the answers and restructuring your business, one can justify the overhead costs and the ever-increasing complexity of information systems that big data brings. In order to effectively work with data, analysts and stakeholders need tools, and it is vital that they fit for solving existing problems.

## 1.2. Big Data Tools

Numerous libraries that hide the complexity of the models behind their software interface is one of the factors making the methods of working with big data more accessible to a wide range of specialists.

On the one hand, this has a positive influence on the speed of the introduction and dissemination of new approaches to business and their integration into a greater number of key business processes. On the other hand, the specialists who only yesterday developed document management systems, by way of example, have a gap between the theory and practice of using machine learning models. This may adversely affect the final result of their work when standard models that work "out of the box" are not enough, and they have to be adjusted or even cascaded with other models.

Therefore, to develop corporate IT infrastructure in the direction of data mining and automation of management decision-making support, the involved specialists need to study the root technologies and the mathematical apparatus that underlie any library of machine learning models. It will not be possible to solve problems on big data without mastering the patterns of parallel data processing, distributed storage, and scalable multi-threaded algorithms. MapReduce, proposed by Google and already considered a classic pattern, can be a good starting point, although it is not so common in Google anymore [37].

The above mentioned "fundamentals" should become the focus of attention when training new personnel in the future, because only this way makes possible the necessary breakthrough growth of automation and informatization of Russian companies at the level of Western competitors and higher.

Today, business is pushing technology development towards big data methods and systems. There are various driving forces for such development [26]:

- Businesses need to be agile and respond to new market insights by quick and cheap hypotheses testing and short development cycles and TTM (Time to Market).
- Companies need to be able to modify their own software and systems, which fully fits into the concept of open source software, which has become very successful.
- CPU clocks are barely increasing, but multi-core processors are standard, and networks are getting faster. This means parallelism is only going to increase.
- Companies often benefit from outsourcing server capacities. Amazon Web Services and Microsoft Azure offer highly demanded cloud infrastructures, such as IaaS (Infrastructure as a Service).

Creating software systems is challenging. And the transition to a parallel pattern of working with data requires special skills and attention to the software being developed. From this point of view the following systems characteristics are especially important:

- Reliability.
- Scalability.
- Maintainability.

Let us dwell on each of these aspects in more detail, give their definitions and justify their importance for the success of corporate information systems in general.

# 2. Characteristics of Enterprise Systems

## 2.1. Reliability

The concept of reliability aggregates in itself such characteristics as the ability to recover from failures, resistance to hacker attacks and the consistent level of performance with user errors, software and hardware faults.

Thus, the system is reliable if it works correctly. There is no sense in creating an application that will survive the destruction of all servers on which it is deployed. However, one must strive to handle known types of failures in the early stages of system development. It is much more difficult to comprehend and rewrite a large system than display a bit of healthy paranoia at the start of work.

In terms of hardware faults, reliability is closely related to scalability, which is discussed further. Scaling data storage, in addition to increasing bandwidth, also protects against data loss due to hard drive faults. Apart from data duplication between data processing centers (DPCs), RAID is also used on a server scale. The problem of hard disk failure seems not so important given the probability of a 1% failure during the first 3 years of operation [41]. But on a storage cluster with 10,000 disks, we should expect on average one disk to die per day.

Not all hardware faults are related to the problem of data storage failure, but loss of information can have huge costs in terms of lost revenue and damage to reputation [13].

One can find software errors just looking at the application code, but the devil is in the details. Such factors as the size of the application code, qualification and perseverance of those who check it, integration with third-party services and the problem of race conditions [42] reduce to zero the theoretical possibility to detect all errors before running the code. Often, software errors lie dormant for a long time until they are triggered by an unusual set of circumstances.

Thus, to ensure the reliability of the system being developed, at a minimum, it is necessary to test the logic of the application and the source code itself, as well as back up data and store the complete log of its changes. Both of these requirements should be fulfilled starting from an early development time frame, and embedded in the overall process on an ongoing basis.

## 2.2. Scalability

There are two types of scalability of information systems: vertical and horizontal or shared nothing [32].

With vertical scalability, a possible increase in throughput and performance is achieved by increasing the capacity of an individual server, for example, by increasing RAM size or replacing the processor with a more powerful one. Horizontal scalability implies that adding a new server increases the load that the system can handle. The term shared nothing well reflects that the servers do not share common resources, such as CPU time, RAM, or hard drives, that is, they are independent.

On the one hand, improving the performance of a single server, instead of using multiple relatively low-power stations, has long been considered outdated, because the frequency of a single processor core no longer increases to a great extent as it did before [47].

On the other hand, there appeared new processors for desktop computers that have 16–18 cores and 32–36 threads of execution with an average frequency of one core equal to 3 GHz, which greatly exceeds the “classic” quad-core processors with a frequency of 3.2–3.6 GHz. Thus, even classical vertical scaling follows the path of horizontal scaling of the processor, increasing

the number of cores rather than the frequency of their work, limited with the current architecture [19].

Anyway, it is a technical debt to consider the option of vertical scaling right from the start, in the current realities of increasing requirements for fault tolerance, throughput, the ability to perform rolling upgrades. There is no doubt about it. Even Martin Fowler in 2002 in his book discussed the architecture of horizontally scalable systems as the most preferable [17].

Thus, it is safe to call developing monolithic applications, which initially do not imply the scaling possibility, an anti-pattern. Service architecture with independent deployment of blocks and their horizontal scalability, is becoming a common approach in the development of complex information systems. It should be noted that there is no need to build a service architecture for simple applications, where such "flexibility" will bring more problems than benefits.

### 2.3. Maintainability

In addition to reliability and scalability, how flexible the system architecture is, how easily it adapts to changing functional and technical requirements on the part of the business, determines whether a business can be quickly rebuilt and remain competitive where the life cycle of projects is measured in years and even decades.

When a project goes into the stage of support for 3, 5, 10 years and new functionality is added rarely, developers supporting such legacy or brownfield [2] project have a desire to rewrite it again. In some cases, it is justified. When a project uses a decade-old technology stack, it becomes harder to find people who can and would like to work with it. It happens that a project that has been living for a long time needs a new functionality or a change of the old-fashioned interface, which entails rewriting the scenarios of the application server part.

Rewriting everything from scratch is not always necessary. When it comes to projects with a monolithic architecture [24], small improvements using new technologies are complicated due to the fragility of the design and code, and the lack of modularity of the system parts. The complexly expandable system can be based on a relational database without replication, where the application logic relies on the ACID guarantees (Atomicity, Consistency, Isolation, Durability) [20]. Also the minimum number of simultaneous user sessions is ideal for functioning of such a system.

In this situation, you can try to carefully divide the monolith into separate services with a limited area of responsibility and the possibility of their independent deployment. It makes no sense to break all application use scenarios into services, but gradually, as the parts of the application are affected by new functionality, it needs to be done.

The modular system of services or micro-service architecture [33] has several advantages, such as independent deployment, loose coupling, using appropriate tools, programming languages, interaction methods and databases for the solution of different tasks [36].

However, the application should not be split into micro-services just for the sake of the Single Responsibility Principle or because the approach requires it. Excessive granularity leads to new problems in support not inherent in monolithic architecture. It becomes difficult to track connection between services, user query execution flows, interaction invariants between versions of deployed services, as well as backward compatibility of contracts.

DDD (Domain Driven Design) approach to software development and dividing the application into multiple bounded contexts makes possible a transition to a single database for a single service [43].

Various factors that influence the maintainability of an application are: the technology used, the culture of writing code, coverage by tests, and architectural decisions taken in the early stages of the project life cycle. There is no easy fix for making the system reliable, scalable and maintainable. But it is absolutely necessary to control the complexity of the project and its technical debt, to adequately assess future extension points, to conduct continuous testing, to have chances not to rewrite the code every two years.

# 3. Data Storage

## 3.1. NoSQL and SQL

Having chosen programming languages, platforms, and the overall technology stack of a project, one faces the question of choosing an appropriate data storage system. Eventually, there is a choice between relational and NoSQL databases.

The concept of a relational data model was first described in the article by Edgar Codd in 1969 [9]. Thus, research and development in the field of relational databases has been going on for 50 years. It is based on a well-developed theoretical apparatus and language for building queries to SQL data (Structured Query Language), which was first standardized at ANSI (American National Standards Institute) and ISO (International Organization for Standardization) in 1986 and 1987, respectively [6].

The term NoSQL in its current interpretation was formed in 2009 [38]. This direction of databases is focused on scalability, fault tolerance, ability to perform a large number of write operations, and the concept of Eventual Consistency (EC) [44] is crucial for them. EC implies that the system data, without being updated over time, will be consistent in all replicas, and data access services will return the same last recorded value from any replica. This guarantee is much weaker compared to ACID, but such behavior allows to achieve fast recording, scalability and fault tolerance.

One of the significant differences between relational and document-oriented databases is considered to be the presence of a strict data schema in the first case and the lack of such in the second [18]. However, this is not entirely true. Relational databases do have a mandatory schema, the so-called schema-on-write, while document-oriented databases allow you to store data in any form, including unstructured data. Though it does not imply that it is possible to work productively with such data. Anyway, there is a schema, but in the form of schema-on-read, when the client code expects to receive data in a specific format. Thus, schema-on-read is a softer limitation than schema-on-write [26]. This freedom on the one hand makes it possible to support multiple versions of data schemas, to perform hot-swapping of deployed services, while the other carries the risk of data inconsistency. A more detailed comparison of the database types in question is presented in the article [18].

However, in many areas considered in the article, and in the field of data storage and processing in particular, there are tendencies to take the best from several worlds, creating hybrid solutions. For example, the Polyglot Persistence approach [16] allows not to make a choice in favor of either a relational database or document-oriented storage, but gives an opportunity to select a variety of data storage options within a single project on grounds of expedience. This approach removes the strict framework and allows you to achieve flexibility by using tools that are best suited for the needs and tasks of the application parts. Alongside Polyglot Persistence, NoSQL techniques are now being introduced into classical relational databases, such as MS

SQL Server and PostgreSQL [46]. In particular, there are new data types that can be stored in columns, for example, JSON documents (JavaScript Object Notation) [45]. But there is a reverse trend, i.e., the creation of SQL-like data access languages in NoSQL solutions, for example, for MongoDB [5].

### 3.2. OLTP and OLAP

In order make a conscious choice between the types of data storage, you need to know the scenarios for its use. The number of write operations per second, the main types of read requests, the types of connections between entities, the cost of data loss, the criteria of fault tolerance – all of this crucially affects the choice between databases and the way data is organized, in particular the choice between OLTP and OLAP.

OLTP (Online Transaction Processing) is a way of organizing databases, in which the system works with small-sized transactions, but with a large stream, and at the same time the client expects a minimum response time from the system [11].

OLAP (Online Analytical Processing) is a data processing technology that consists in preparing summarized (aggregated) information based on large datasets, structured according to a multidimensional principle [10].

Based on these basic definitions, OLTP databases are used for prompt processing of user requests, while OLAP solutions are used for analyzing the system's snapshot at any point in time, such as OLAP cube [34], Data Warehouse [40], or Data Lake [25]. The use of OLTP solutions is typical of business transaction scenarios [17], where user input initiates writing to the database and executing read requests.

The need to create analytical reports scanning a large volume of data, possibly entire tables, led to the emergence of OLAP solutions that support a different interaction pattern than the OLTP solutions.

The comparison of access patterns for these two classes of solutions is given in Tab. 1 [26]. Initially, the same databases were used for both transaction scenarios and creation of analytics.

**Table 1.** Comparing Characteristics of Transaction Processing versus Analytical Systems

| Property | Transaction processing systems (OLTP) | Analytic systems (OLAP) |
|---|---|---|
| Main read pattern | Small number of records per query, fetched by the key | Aggregate over the large number of records |
| Main write pattern | Random-access, low-latency writes from user input | Bulk import or event stream |
| Primarily used by | End-user/customer, via web application | Internal analyst, for decision support |
| What data represents | Latest state of data (current point in time) | History of events that happened over time |
| Dataset size | Gigabytes to terabytes | Terabytes to petabytes |

In this regard, SQL proved a powerful and flexible tool. However, over time, analystics was removed into separate Data Warehouses [12]. Data Warehouse receives the data gathered from all available sources in the company, which is aggregated, cleaned, transformed into a convenient

format for creating analytics and loaded into the repository without editing options, i.e., for read-only. The described data loading process is called Extract-Transform-Load (ETL), that is, divided into stages: data extraction, transformation and loading [26].

While OLTP contains mostly normalized data without duplicating of information, OLAP solutions achieve their goals by denormalizing data. Thus, the number of table join operations is reduced.

The above systems are designed to solve different problems. Combining creation of analytics and reports with execution of users' business transactions may be easy at the start of the project, but give less flexibility in the long run and greatly affect the performance of both usage scenarios with increasing data. However, there are systems on the market that combine both solutions, for example, Microsoft SQL Server and SAP Hana, giving access through a common SQL interface [15, 29].

It may seem that the degradation of system performance with an increase in the amount of data when using one solution instead of two is acceptable within reasonable limits. However, Amazon's research suggests that increasing server response time by only 100 ms reduces revenue by 1% [31], other studies indicate that a 1 second slowdown reduces customer satisfaction by 16% [3, 14].

You need to cater to the needs of people who work with your systems every day throughout their working hours, increasing efficiency, reducing the time of response and report generation, not blocking the system operation with modal windows, and so on and so forth.

Storing the stream of events that occurred in the system is a fundamentally different storing data option, compared with storing only the system's current state. This option requires a different attitude and design, but the result makes it possible to receive the system status at any time, add analytics to make it as complex and deep as can be.

### 3.3. Event Stream Storage

Machine learning in general and its models in particular require source data, and the more extensive and high-quality it is, the more effective the work gets. This explains the importance of the source material, the lack of which may be the problem with the customary approach of storing only the last state of existing business entities.

The transition from storing only the current state without the history of changes, which would allow to restore data as of any moment of the systems existence, to storing the stream of events affecting the data, as the primary source of application data, seems to be a very logical step in the development of technology and world view of the community of programmers, architects and business in the broad sense of the word. Keeping a complete history of changes gives data tools "food for thought".

However, it is necessary to perform not only the transition to systems with events stream storing, but also the transition from a single relational data repository, or a synchronous replication repository – to which the community has become accustomed and even addicted – to a distributed repository with replication and partitioning, or sharding. The transition to asynchronous propagation of changes and abandoning distributed transactions is also a must as a guarantee of the integrity of operations, though it comes at a price in terms of overall system performance and its throughput for reading and writing.

Such shift in the mass paradigm in application development seems just as necessary and inevitable as the transition from single-core and single-thread systems and the absence of Race

Conditions [42] to a multi-thread model that imposes certain restrictions, requires greater care and accuracy, as well as the use of tools for the synchronization of threads and execution processes.

It is worth noting that despite the fact that multi-core systems and software models for working with them have been around for a long time by the IT standards, developers are still rather slow at mastering them. This may partially be due to the inertia of the process of higher education, which does not yet cover this aspect by default, as basic computer literacy, and partially due to the fact that many companies need a quick solution, designed for only a few users with certain risk of data loss and a high time of response that is considered non-critical for internal corporate users, and such approach prevails over more costly current approaches. Therefore, employees find it hard and unnecessary to learn new things, since there is no demand for such skills in the company.

Of course, the idea of storing the history of data change and obtaining its inherited representation cannot be called a new one. A similar approach is found in relational databases which store the transaction log and WAL (write ahead log) for indexes. They allow to restore the state of a database after failures and deadlocks, as well as conduct an audit, but are limited in size and are often cleared after a certain period of time, that is, not stored forever.

However, storing the entire stream of changes and events, in particular Event Sourcing, does not involve the removal of old events to save disk space. In contrast, the events that occur are considered immutable, and this has several advantages. Among such advantages we can single out the possibility of the in-depth analysis of the history of system events, creation of analytics of any complexity having a complete history, not just the latest actual state of the system, caching events, etc.

The transition to the accumulation of huge datasets has set the task of their smart analysis, identifying patterns and predicting the behavior of systems that are directly affected by feedback from end users. The practical application of the revived direction of machine learning has become the solution to these issues, without which it seems impossible to continue effective business operations.

# 4. Machine Learning

## 4.1. Relevance

The application of machine learning has literally captured the minds of IT. Predicting product demand, personalized targeted advertising, fraud detection – these are just a few of the applications that everyone has heard of. To get the idea of the great demand for specialists in this field, it is enough to analyze data from hh.ru website. The current labor market demonstrates a high demand for skills and technologies directly related to big data, analysis and machine learning [21]. Python, Big Data, Machine Learning, Hadoop, Spark, Data Mining, Deep Learning, Scala are at the top of the list.

The hype around this new and exciting – by IT standards – field, attracts more and more young professionals. But besides yesterday's students, developers with experience are also eager to try their hands at the field of intelligent systems. Interest in another "breakthrough" framework for building server or client-side of applications fades as time goes by, some new technologies appear, and there is no bottom to this turmoil. However, when it comes to big data and machine learning, this new, previously unavailable business tool gives a bonus unattainable

before – new knowledge. Therefore, the relevance of this direction will continue to increase with the development of models and methods.

The analysis of historical data allows us to understand where the business is losing money, to identify hidden trends and relationships, to avoid unprofitable decisions using existing experience.

Stream processing of new data allows you to make tactically more balanced decisions, increase profits in the short term, identify and prevent malicious activity, thereby reducing losses.

Any new field requires workers of a new type, and now this is the case with Data Science (DS) specialists. The thing is, data science specialists are not software engineers [1]. The key skills of engineers are programming and creating software systems, whereas the core competencies of a DS specialist are mathematical statistics, mathematics in general, machine learning and analysis. Most of all, these two profiles overlap in the area of big data.

Attempts to impose responsibilities of engineers on DS specialists lead to a loss of efficiency. The speed of solving problems associated with data analysis can reduce by 70–80% [1]. Therefore, companies need to divide these areas, distribute responsibilities and competencies of employees. But there must be a point of contact, and thus a machine learning engineer becomes such point. Usually software engineers with a set towards mathematics and 3–6 years of experience in software development and data flow design become specialists of this kind. They feel cramped in the framework of their routine tasks and see an opportunity to do what they have always wanted, but found difficult to start.

The toolkit of today allows a gradual transition from software engineering to data science, step by step, deepening the knowledge of underlying theory. Such training can take years, but it is possible that in 5–10 years we will see a legion of this kind of programmers.

In many ways, the explosive growth of the popularity of various machine learning models and distributed systems that handle large volumes of often unstructured data is caused by the advances in computer technology performance, increasing the number of processor cores, reducing the cost of data storage, including SSD technology, and most importantly the ever-growing use of cloud services as basics of available distributed computing.

## 4.2. Applicability

The problem of many companies is that there is not enough understanding of where and how to appropriately apply methods and models of machine learning to the available data on business processes. It is also a non-trivial task to find the sources of information from which to write the history of changes in the long-term storage for its further analysis, to identify trends, to ask the right questions, the answers to which will help reduce costs and increase the efficiency of the company as a whole.

Existing machine learning models, such as regression, classification, clustering, and neural networks, have proven themselves effective, but, as in many areas of science and technology, hybrid models are of considerable interest, those that are created at the intersection, absorbing all the best from several families of models and bringing something new, increasing the accuracy and speed of forecasting.

In Western and European countries, in particular in the Netherlands, there have long existed degree courses in data science. In Russia, however, the practical aspect of applying university knowledge in mathematical statistics, theory of probability, econometrics, and mathematical analysis is not yet so well emphasized. Such a gap between training and rapidly changing needs of the labor market adversely affects the pace of automation of business processes. In addition,

not all IT specialists at this stage feel an urgent need for retraining, mastering skills related to machine learning, big data, distributed computing, algorithms used in cryptocurrencies.

Such rigidity from the part of developers is largely determined by the inertia of processes and repeated tasks, which programmers in IT departments of large companies have faced for the most part over the past 10 years. Growing companies, by contrast, can afford to adapt more quickly and try to introduce new models. It is worth pointing out that yesterday's students learn more quickly, pursue career path faster and sometimes, as early as at the age of 21, work in the position of Senior Developer.

Large corporations, which have established complex and highly inert business processes and are not largely involved into information technology and consulting in the field of data analysis, can experience considerable difficulties in the transition to new business models. Despite this, such complete rethink of information expertise seems vital.

Timely upgrade will allow to remain competitive in the market, actualize the technologies and knowledge which among other things can be used to reduce costs associated with support of outdated approaches and tools.

## Conclusion

Whatever direction the enterprise information systems development may evolve in, – be it a variety of data storage solutions, evolution of processors, introduction of new software architecture – the major ubiquitous tendency is moving away from the vertical scalability to the horizontal one. With the growth of data volume and the complexity of its analysis, it is the scalability of computing and data warehousing that is becoming the cornerstone of further software development evolving. Supercomputing systems are becoming the foundation of the future digital economy of Russia and the whole world. There is a clear need for decentralized development and extensive use of supercomputers at universities and enterprises. A supercomputing complex at SUSU in Chelyabinsk is an example of such successful implementation and application [27, 28].

Despite the abundance of terms and cliched phrases, such as big data, machine learning, the Internet of things, asynchronous behavior, parallelism, distribution, they are united by common basic concepts, ideas, and problems, already known in the twentieth century. For example, abstractions like Acknowledgment or Race Condition Automaton have existed in circuit engineering for a long time, but modern IT continues to rediscover them again and again with the advent of new fashionable development technologies and paradigms.

From this perspective, modern "breakthrough" and "hype" trends seem to be somewhat a rethinking of the existing bundle of knowledge, showcasing it in a new sparkling edition. Hence, there is no "silver bullet", which would solve all problems in a certain area at the snap of your fingers. Indeed, the development of such a universal tool may require a significant rethinking of approaches to building information systems.

The inertia of introducing new approaches to development is explained by the complexity of the mental shift towards the event stream as the main option for storing data, though it is also well-grounded on the past. For instance, take an account book, where entries are made, but not corrected after being made, and errors in previous entries are only compensated by new lines.

Due to ideological, methodological and technical difficulties, enterprise systems in the vast majority of companies are built using approaches that have been approbated over the past decade.

We can say that there is a number of common tasks, such as document management, authentication, authorization, CRUD operations for various business entities (Create, Read, Update, Delete), building reports and analytics, etc. Their solutions have also been largely established, though there are various options both from the point of view of the technological stack and the quality of the implementation.

Despite the evolution of hardware and IT infrastructure, the mathematical interpretation of the domain processes has not changed so much compared to the changes in technosphere.

For example, today much is said about sensor networks and what can be achieved with their help to describe various objects of control. But at the same time, it does not matter how many sensors there are on an object that we want to control more efficiently, these sensors only record the dynamics of a control object's state, and nothing more. The controlling mechanisms themselves have not experienced any breakthrough. In essence, this process of reflecting the characteristics of a control object in time is nothing more than a digital shadow. And globally, decision-makers are still responsible for making decisions based on experience and intuition, rather than on adequate models that would allow to justify these decisions.

Modern academic research is out of touch with the practice of industry. Theoretical studies are important, but, unfortunately, are very far from practical needs.

The field of data science, entering upon the stage of IT specialists training, will very likely change, or rather, is already changing, the vector of information systems development, training specialists for their building and maintenance, and what is most important, the tasks, solving which business becomes more competitive.

It can be unequivocally asserted that big data, data science, and evolution of scalability and fault tolerance of information systems have predetermined the development of information technologies in the long term. Companies which do not take advantage of the full range of the offered opportunities can stay afloat, but only those that invest resources and adapt to rapidly changing realities will ensure maximum benefit and gain the upper hand.



## References

1. Anderson, J.: Data Engineers vs. Data Scientist. `https://www.oreilly.com/ideas/data-engineers-vs-data-scientists` (2018), accessed: 2019-10-13

2. Belcham, D., Baley, K.: Brownfield application development in. NET. Manning Publications Co. (2010)

3. Brutlag, J.: Speed Matters for Google Web Search. `http://googleresearch.blogspot.com/2009/06/speed-matters.html` (2009), accessed: 2019-10-13

4. Buxton, B.: Big Data: the Next Google. `https://www.nature.com/news/2008/080903/full/455008a.html` (2008), accessed: 2019-10-13

5. Celesti, A., Fazio, M., Villari, M.: A study on join operations in MongoDB preserving collections data models for future internet applications. Future Internet 11(4), 83 (2019), DOI: 10.3390/fi11040083

6. Chamberlin, D.D.: Early history of SQL. IEEE Annals of the History of Computing 34(4), 78–82 (2012), DOI: 10.1109/MAHC.2012.61

7. Cielen, D., Meysman, A.D.B., Ali, M. (eds.): Introducing Data Science. Manning (2016)

8. Clarke, T.: Twenty Two Plus Instagram Stats That Marketers Can't Ignore This Year. https://blog.hootsuite.com/instagram-statistics (2019), accessed: 2019-10-13

9. Codd, E.F.: Derivability, redundancy and consistency of relations stored in large data banks. IBM Research Report, San Jose, California RJ599 (1969)

10. Codd, E.F., Codd, S.B., Salley, C.T.: Providing OLAP (on-line analytical processing) to user-analysts: an it mandate (1992)

11. Conn, S.S.: OLTP and OLAP data integration: a review of feasible implementation methods and architectures for real time data analysis. In: Proceedings of the IEEE SoutheastCon 2005, 8-10 April 2005, Ft. Lauderdale, FL, USA. pp. 515–520 (2005), DOI: 10.1109/SECON.2005.1423297

12. Dedi, N., Stanier, C.: An evaluation of the challenges of multilingualism in data warehouse development. In: Proceedings of the 18th International Conference on Enterprise Information Systems, 25-28 April 2016, Rome, Italy. SCITEPRESS - Science and and Technology Publications (2016), DOI: 10.5220/0005858401960206

13. Drinkwater, D.: Does a Data Breach Really Affect Your Firm's Reputation. https://www.csoonline.com/article/3019283/does-a-data-breach-really-affect-your-firm-s-reputation.html (2016), accessed: 2019-10-13

14. Everts, T.: The Real Cost of Slow Time vs Downtime. http://www.webperformancetoday.com/2014/11/12/real-cost-slow-time-vs-downtime-slides, accessed: 2019-10-13

15. Färber, F., May, N., Lehner, W., et al.: The SAP HANA database – an architecture overview. IEEE Data Eng. Bull. 35(1), 28–33 (2012), http://sites.computer.org/debull/A12mar/hana.pdf

16. Fowler, M., Sadalage, P.: The Future is Polyglot Persistence. https://martinfowler.com/articles/nosql-intro-original.pdf (2012), accessed: 2019-10-13

17. Fowler, M.: Patterns of enterprise application architecture. Addison-Wesley Longman Publishing Co., Inc. (2002)

18. Gaikwad, R.G., Goje, A.: SQL and NoSQL: Which is better. International Journal of Emerging Technologies and Innovative Research 2(8), 3277–3284 (2015), http://www.jetir.org/papers/JETIR1508005.pdf

19. Gepner, P., Kowalik, M.F.: Multi-core processors: New way to achieve high system performance. In: International Symposium on Parallel Computing in Electrical Engineering, 13-17 Sept. 2006, Bialystok, Poland. pp. 9–13 (2006), DOI: 10.1109/PARELEC.2006.54

20. Gray, J.: The transaction concept: Virtues and limitations. In: Proc. of the 7th Int. Conf. on Very Large Databases, 13-17 Sept. 1981, Cannes, France. pp. 144–154 (1981)

21. Grishina, A.: Big Data and Data Science Labor Market Survey. https://habr.com/company/newprolab/blog/320336 (2017), accessed: 2019-10-13

22. Hilbert, M., López, P.: The world's technological capacity to store, communicate, and compute information. Science 332(6025), 60–65 (2011), DOI: 10.1126/science.1200970

23. Jun, S.P., Yoo, H.S., Choi, S.: Ten years of research change using Google Trends: From the perspective of big data utilizations and applications. Technological Forecasting and Social Change 130, 69–87 (2018), DOI: 10.1016/j.techfore.2017.11.009

24. Kalske, M., Mäkitalo, N., Mikkonen, T.: Challenges when moving from monolith to microservice architecture. In: Current Trends in Web Engineering - ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine, 5-8 June 2017, Rome, Italy, Revised Selected Papers. pp. 32–47 (2017), DOI: 10.1007/978-3-319-74433-9_3

25. Khine, P.P., Wang, Z.S.: Data lake: a new ideology in big data era. In: ITM Web of Conferences. vol. 17, p. 03025. EDP Sciences (2018), DOI: 10.1051/itmconf/20181703025

26. Kleppmann, M.: Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly (2016), http://shop.oreilly.com/product/0636920032175.do

27. Kostenetskiy, P., Safonov, A.: SUSU supercomputer resources. In: Proc. of the 10th Annual Int. Scientific Conf. on Parallel Computing Technologies, PCT 2016, 29-31 March 2016, Arkhangelsk, Russia. CEUR Workshop Proceedings. vol. 1576, pp. 561–573 (2016)

28. Kostenetskiy, P., Semenikhina, P.: SUSU supercomputer resources for industry and fundamental science. In: 2018 Global Smart Industry Conference, GloSIC, 13-15 Nov. 2018, Chelyabinsk, Russia. pp. 1–7. IEEE (2018), DOI: 10.1109/GloSIC.2018.8570068

29. Larson, P., Clinciu, C., Fraser, C., et al.: Enhancements to SQL server column stores. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, 22-27 June 2013, New York, NY, USA. pp. 1159–1168 (2013), DOI: 10.1145/2463676.2463708

30. Li, C., Gu, J.: An integration approach of hybrid databases based on SQL in cloud computing environment. Software: Practice and Experience 49(3), 401–422 (2019), DOI: 10.1002/spe.2666

31. Linden, G.: Make data useful. https://www.scribd.com/doc/4970486/Make-Data-Useful-by-Greg-Linden-Amazon-com (2006), accessed: 2019-10-13

32. Liu, C.Y., Shie, M.R., Lee, Y.F., et al.: Vertical/horizontal resource scaling mechanism for federated clouds. In: 2014 International Conference on Information Science Applications, 6-9 May 2014, Seoul, South Korea

33. Namiot, D., Sneps-Sneppe, M.: On micro-services architecture. International Journal of Open Information Technologies 2(9), 24–27 (2014)

34. Naouali, S., Salem, S.B.: Towards reducing the multidimensionality of OLAP cubes using the evolutionary algorithms and factor analysis methods. CoRR abs/1602.04613 (2016), http://arxiv.org/abs/1602.04613

35. Novikov, D.A.: Big Data and Big Management. https://mipt.ipu.ru/sites/default/files/page_file/BigDataBigControl.pdf, accessed: 2019-10-13

36. Richter, J.: Architecting Distributed Cloud Applications. https://www.youtube.com/watch?v=xJMbkZvuVO0 (2017), accessed: 2019-10-13

37. Robinson, H.: The Elephant Was a Trojan Horse: On the Death of Map-Reduce at Google. https://www.datacenterknowledge.com/archives/2014/06/25/google-dumps-mapreduce-favor-new-hyper-scale-analytics-system (2014), accessed: 2019-10-13

38. Sadalage, P.: NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Addison-Wesley, Upper Saddle River, NJ (2013)

39. Safaei, B., Monazzah, A.M., Bafroei, M., et al.: Reliability side-effects in Internet of Things application layer protocols. In: 2017 2nd International Conference on System Reliability and Safety, 20-22 Dec. 2017, Milan, Italy. pp. 207–212. IEEE (2017), DOI: 10.1109/ICSRS.2017.8272822

40. Singh, S.: Data warehouse and its methods. Journal of Global Research in Computer Science 2(5), 113–115 (2011)

41. Su, C.J., Huang, S.F.: Real-time big data analytics for hard disk drive predictive maintenance. Computers & Electrical Engineering 71, 93–101 (2018), DOI: 10.1016/j.compeleceng.2018.07.025

42. Unger, S.H.: Hazards, critical races, and metastability. IEEE Transactions on Computers 44(6), 754–768 (1995), DOI: 10.1109/12.391185

43. Viggiato, M., Terra, R., Rocha, H., et al.: Microservices in practice: A survey study (2018)

44. Vogels, W.: Eventually consistent. Communications of the ACM 52(1), 40 (2009), DOI: 10.1145/1435417.1435432

45. Šimec, A., Maglii: Comparison of JSON and XML data formats. In: Central European Conference on Information and Intelligent Systems (2014)

46. Vyawahare, H., Karde, P., Thakare, V.M.: A hybrid database approach using graph and relational database. In: 2018 International Conference on Research in Intelligent and Computing in Engineering, 22-24 Aug. 2018, San Salvador, El Salvador. pp. 1–4 (2018), DOI: 10.1109/RICE.2018.8509057

47. Zhislina, V.: Why the frequency does not increase. https://software.intel.com/en-us/blogs/2014/02/19/why-has-cpu-frequency-ceased-to-grow (2014), accessed: 2019-10-13

DOI: 10.14529/jsfi200104

# Online MPI Process Mapping for Coordinating Locality and Memory Congestion on NUMA Systems

*Mulya Agung*[1], *Muhammad Alfian Amrizal*[2], *Ryusuke Egawa*[3,1], *Hiroyuki Takizawa*[3,1]



Mapping MPI processes to processor cores, called process mapping, is crucial to achieving the scalable performance on multi-core processors. By analyzing the communication behavior among MPI processes, process mapping can improve the communication locality, and thus reduce the overall communication cost. However, on modern non-uniform memory access (NUMA) systems, the memory congestion problem could degrade performance more severely than the locality problem because heavy congestion on shared caches and memory controllers could cause long latencies. Most of the existing work focus only on improving the locality or rely on offline profiling to analyze the communication behavior.

We propose a process mapping method that dynamically performs the process mapping for adapting to communication behaviors while coordinating the locality and memory congestion. Our method works online during the execution of an MPI application. It does not require modifications to the application, previous knowledge of the communication behavior, or changes to the hardware and operating system. Experimental results show that our method can achieve performance and energy efficiency close to the best static mapping method with low overhead to the application execution. In experiments with the NAS parallel benchmarks on a NUMA system, the performance and total energy improvements are up to 34% (18.5% on average) and 28.9% (13.6% on average), respectively. In experiments with two GROMACS applications on a larger NUMA system, the average improvements in performance and total energy consumption are 21.6% and 12.6%, respectively.

*Keywords: communication, congestion, locality, MPI, multi-core, NUMA, process mapping.*

## Introduction

In the fields of high-performance computing (HPC) and enterprise computing, a large-scale Symmetric Multi-Processing (SMP) system is typically built by using multiple processors to have more processor cores within a system. These processors have shared memories and on-chip memory controllers that form the base for NUMA (Non-Uniform Memory Access) multi-processors. Each processor consists of one or several groups of processor cores, each of which is physically associated with one or more memory controllers and memory devices. Such a group of processor cores is referred to as a NUMA node [13, 16, 25]. Although the NUMA nodes are generally connected by high-speed interconnect links such as QuickPath Interconnect (QPI) [32] and HyperTransport [25], accessing a remote NUMA node still needs a longer latency than accessing the data of the local memory device. Thus, the cost of remote memory access is higher than that of local memory access.

Currently, modern multi-core processors are widely used not only for shared-memory parallel processing but also for distributed-memory parallel processing, such as Message Passing Interface (MPI) [24]. In MPI, communication among MPI processes is explicit and is performed by sending and receiving messages. Each MPI process has a unique identifier called process ID or process rank. The process that sends the message is called a sender, while the process that receives the message is called a receiver. Thus, a communication event can be defined as one message with

[1]Graduate School of Information Sciences, Tohoku University, Japan
[2]Research Institute of Electrical Communication, Tohoku University, Japan
[3]Cyberscience Center, Tohoku University, Japan

its corresponding processes of a sender and a receiver. This pair is also referred to as a process pair [12, 21].

In NUMA systems, a communication event will access the local memory device if it is performed by a process pair whose sender and receiver are executed by different cores of the same NUMA node. On the other hand, it will access the memory device of the remote NUMA node if it is performed by a process pair whose sender and receiver are executed by different cores of different NUMA nodes. MPI provides extensions that enable faster intra-node communication through the use of shared memory, such as Nemesis for MPICH2 [8] and KNEM [17] for Open MPI [15]. However, as the cost of communication significantly affects the performance on NUMA systems, exploiting the communication behavior to optimize the mapping between MPI processes and processor cores is necessary to improve performance. Such process mapping methods are called *communication-aware process mapping* [12, 26].

In modern NUMA systems, process mapping becomes more challenging because a large number of processor cores in a system induce a large number of accesses to memory devices. As the number of processor cores increases, the number of communication that can simultaneously happen will also increase, causing congestion on shared caches and memory controllers. We refer to this congestion as *memory congestion*. Conventional work on MPI process mapping mostly focuses on improving the locality of communication by mapping processes frequently communicate with each other, to processor cores that are closer to each other in the memory hierarchy. Improving the locality of communications is important because it will reduce the cost of communication and congestion on interconnect links. However, only considering the locality is not sufficient to improve performance on modern NUMA systems. Furthermore, maximizing the locality can degrade performance because it potentially increases the memory congestion [2, 16].

To optimize the process mapping, it is necessary to analyze the communication behavior of the MPI applications. The communication behavior is determined by the communication among MPI processes of the application. A process does not necessarily need to communicate with all the other processes, and the time and amount of data exchanged among processes may vary. Conventional process mapping methods rely on offline profiling to trace the communication events and analyze the communication behavior. However, the offline profiling and analysis impose a high overhead and is not applicable if the application changes its communication behavior between executions. Furthermore, the data generated during profiling might be very large, requiring a time-consuming analysis [30].

In this paper, we present a process mapping method, called *Online Decongested Locality for MPI* (OnDeLoc-MPI), to tackle the locality and the memory congestion problems on modern NUMA systems. It consists of a mechanism that dynamically performs the process mapping for adapting to changes in the communication behavior, and a mapping algorithm to calculate the process mapping that can simultaneously reduce the amount of remote memory accesses and the memory congestion. OnDeLoc-MPI works at runtime during the execution of an MPI application. It does not require prior knowledge of the communication behavior, modifications to the application, or changes to the hardware and operating system.

The rest of the paper is organized as follows. In Section 1, we discuss related work and compare it to OnDeLoc-MPI. Then, we describe the procedure and implementation of OnDeLoc-MPI in Section 2. Experimental setup and results are presented in Section 3. This section also discusses the overhead of our method. Finally, conclusions and future work are summarized in Section 3.5.

## 1. Related Work

Various MPI process mapping methods have been proposed in the related studies. Most of the methods rely on offline profiling to trace communication between processes and to analyze the communication behaviors of the applications [2, 9, 21, 31]. The main drawback of these methods is the requirement of offline profiling, which has a high overhead and is potentially time-consuming. On the other hand, the proposed OnDeLoc-MPI does not have these disadvantages because it performs the process mapping dynamically at runtime during the execution of the application.

An online communication detection method, called CDSM, has been proposed in a related study [13]. It works on the operating system level during the execution of the parallel application. It detects the communication behavior from page faults, and uses this information to dynamically perform the process and thread mapping. CDSM employs a locality-based mapping algorithm to minimize the communication cost. The evaluation results of the related work have shown that CDSM can improve the performance of the MPI benchmarks. There are two key differences between CDSM and this work. First, OnDeLoc-MPI does not need to employ a communication detection mechanism because communication events can be traced directly from MPI events. Thus, it does not suffer from detection inaccuracy nor overhead caused by the communication detection mechanism. Second, OnDeLoc-MPI employs a mapping algorithm that aims to reduce both the communication cost and the memory congestion. In Section 3, we compare the performance results of OnDeLoc-MPI and CDSM, and discuss the benefits of our method.

A memory placement method, called Carrefour, has been proposed in [10]. It improves performance on modern NUMA systems by reconciling the data locality and the memory congestion problems. Carrefour works as a data mapping policy of the Linux kernel to dynamically place memory pages on NUMA nodes to avoid the congestion. Since the method works at system runtime, it suffers from the overheads caused by the memory access sampling and memory page replication. Lepers et al. [23] proposed a thread and memory placement method, called AsymSched, that considers the bandwidth asymmetry of asymmetric NUMA systems to minimize congestion on interconnect links and memory controllers on modern NUMA systems. It relies on continuous sampling of the memory accesses to analyze the communication among threads.

We cannot compare OnDeLoc-MPI with Carrefour and AsymSched because both methods require a sampling mechanism that is available only in AMD processors. However, in contrast to these methods, OnDeLoc-MPI analyzes the communication behavior directly from MPI communication events, and it does not require the communication detection and memory sampling mechanisms. Moreover, OnDeLoc-MPI works on the runtime system level, and it does not rely on a specific operating system or hardware. Compared with AsymSched, OnDeLoc-MPI focuses on reducing not only memory congestion but also the amount of remote accesses. Our evaluation results in Sections 3.2 and 3.3 show that the reduction of the amount of remote accesses can substantially improve performance and energy efficiency.

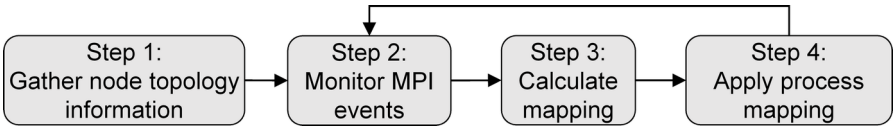
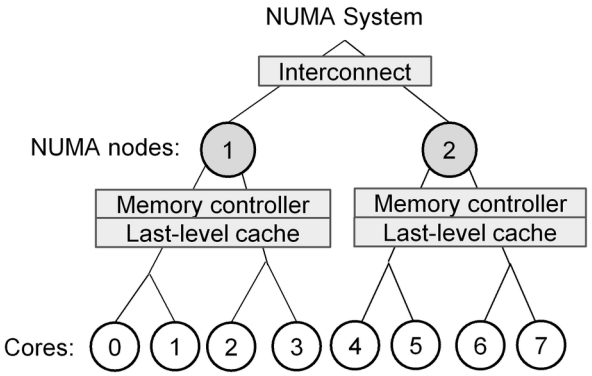



**Figure 1.** The procedure of OnDeLoc-MPI




(a) A node topology with eight cores

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 10 | 500 | 500 | 0 | 1 | 0 | 0 | 0 |
| 2 | 10 | 100 | 0 | 500 | 0 | 0 | 0 | 0 |
| 1 | 100 | 0 | 100 | 500 | 0 | 0 | 0 | 0 |
| 0 | 0 | 100 | 10 | 10 | 0 | 0 | 0 | 0 |
| Process ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

(b) A communication matrix with eight processes

**Figure 2.** The examples of NUMA node topology and communication matrix

## 2. OnDeLoc-MPI: An Online Process Mapping Method for Coordinating Locality and Memory Congestion

In this section, we describe how OnDeLoc-MPI works during the execution of an MPI application. We first explain the procedure of the method, and then describe the implementation of the method in the MPI runtime system.

### 2.1. Procedure of OnDeLoc-MPI

Figure 1 shows the procedure of OnDeLoc-MPI, which consists of four steps:

1. Gather the node topology information of the NUMA system.
2. Monitor MPI communication events during the execution.
3. Calculate the MPI process mapping.
4. Apply the process mapping.

First, when the target application is launched, OnDeLoc-MPI obtains the information about the NUMA node topology of the target system. The topology is modeled as a tree to express the information about the locations of cache memories, memory controllers, and interconnect links. In NUMA systems considered in this work, each NUMA node is physically associated with a shared last-level cache (LLC) and an integrated memory controller, such as Intel-based and AMD-based NUMA systems [25]. Thus, the location of the NUMA node represents both the location of memory controllers and LLCs. This information is required because OnDeLoc-MPI focuses on reducing the amount of remote accesses through interconnects and reducing the congestion on the shared caches and memory controllers. Figure 2(a) shows an example of the model of a two-node NUMA system that consists of eight processor cores. The model also contains information about the physical identities of the NUMA nodes and the processor cores. This information is used later by the mapping algorithm to calculate the mapping.

Second, during the execution of the application, the communication behavior of the application is analyzed by monitoring the communication events among MPI processes. A commu-

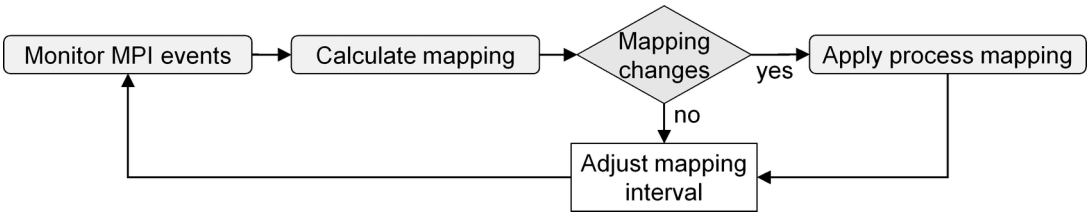



**Figure 3.** The mechanism of mapping interval adjustment

nication matrix is used to model the communication behavior, and it consists of identifiers of MPI processes and amount of communication among the processes. The communication matrix is a square matrix of order $N_p$, where $N_p$ is the number of MPI processes executed by the application. It has the same number of rows and columns because each process can communicate with all the other processes. Figure 2(b) shows an example of the communication matrix for an application that consists of eight processes. Each cell $(x, y)$ of the matrix contains the amount of communication between a pair of processes $x$ and $y$, which is obtained by aggregating the volume and number of communication events between the pair. In the communication behavior shown by the matrix, processes 0 to 3 have a larger amount of communication than the other processes. Since the communication behavior of the application may change during the execution, we update the communication matrix periodically with a certain time interval, called *mapping interval*. At the beginning of the execution, the values of all cells are set equal to zero.

In the third and fourth steps, OnDeLoc-MPI uses the information about communication behavior to optimize the process mapping. In Step 3, the mapping is obtained by using an algorithm, called OnDeLocMap+ algorithm, which is executed every time the communication matrix is updated. The algorithm calculates the mapping that can improve the communication locality and the memory congestion, which is detailed in Section 2.2. Then, in Step 4, the calculated mapping is applied to the execution by assigning processor cores to processes according to the mapping. Since the mapping result of Step 3 can be different for each calculation, OnDeloc-MPI will migrate a process if the mapping of the process changes from the previous mapping.

As shown in Fig. 1, Steps 2 to 4 are performed iteratively during the execution of the application. We note that the mapping interval can significantly affect the performance results and overhead of our method. If the interval is too long, OnDeLoc-MPI may not adapt quickly to the changes in the communication behavior. On the other hand, a shorter interval will increase the overhead because it increases the frequency of updating the communication matrix and calculating the process mapping. We discuss the overhead of OnDeLoc-MPI in more detail in Section 3.4.

To reduce the overhead, OnDeLoc-MPI dynamically adjusts the mapping interval, which is shown in Fig. 3. We use a slope parameter, $v$, to automatically increase and decrease the mapping interval, where $v > 1$. If the calculated mapping does not differ from the previous mapping, the mapping interval is multiplied by $v$ as in Equation 1. The mapping interval is increased because the current mapping is assumed to be stable. On the other hand, if the mapping differs from the previous mapping, the mapping interval is divided by $v$ as in Equation 2. We shorten the mapping interval so that the mapping can adapt more quickly to changes in the communication behavior.

$$Interval_{curr} = Interval_{prev} \times v. \tag{1}$$

$$Interval_{curr} = \frac{Interval_{prev}}{v}. \tag{2}$$

## 2.2. Implementation of OnDeLoc-MPI

OnDeLoc-MPI has been implemented as a module for Open MPI runtime system [15]. The implementation consists of four parts:

1. A modification to the monitoring layer of the runtime system.
2. Data consolidation among MPI processes.
3. OnDeLocMap+ algorithm.
4. Data structures to store the communication matrices and process mapping.

### *2.2.1. Modification of the runtime system*

For the implementation, we added a module in the monitoring layer of the Open MPI runtime system, and the module is started when an MPI application is launched by the runtime system. To perform Steps 1 and 2 of the OnDeLoc-MPI procedure, we have implemented two functions in the module. The first function obtains the node topology information of the system by using Hwloc library [7]. We use this library because it can provide both logical and physical indexes of the processor cores and the NUMA nodes. The second function monitors MPI communication events during the execution. During the monitoring, the amount of communication of each process pair is accumulated using a counter. This function uses a monitoring framework [6] that is built on top of the point-to-point management layer (PML) of the Open MPI stack [15]. We use PML because it can monitor point-to-point operations organizing a collective communication, and thus the communication events can be traced in both cases of point-to-point and collective communications.

For Steps 3 and 4, we create a thread, called *mapper thread*, that periodically calculates and applies the process mapping. This thread is a child thread of the process that has the local ID 0. In MPI, the local ID is the local rank of an MPI process within a system [15, 18]. It means that if an MPI application is executed with more than one NUMA system, OnDeLoc-MPI will create one mapper thread for each system, and each thread calculates and applies the process mapping separately for each system. To apply the process mapping, we assign the target processor cores to processes according to the mapping by using the `sched_setaffinity()` function call of the Linux system. However, the use of this function is not mandatory. Alternatively, some libraries such as Hwloc-bind [7] and Likwid-pin [29] can be used to assign processor cores to processes.

We are aware that, in an MPI application, two MPI processes running on different NUMA systems may communicate with each other. Thus, calculating the process mapping separately for each system may not result in the best mapping for the application. However, by performing the process mapping separately for each system, OnDeLoc-MPI does not suffer from the overhead of migrating processes from one system to another system. This overhead may surpass the benefit of our method because migrating MPI processes across systems potentially incurs a significant network overhead [5]. In the future, we will discuss the impacts of migrating MPI processes across NUMA systems to the performance results of our method.

### *2.2.2. Data consolidation*

In an MPI application, each MPI process is executed on a processor core as a single operating system process. However, the PML monitors MPI communication events separately for each process, and in Linux and other UNIX operating systems, a process cannot directly access

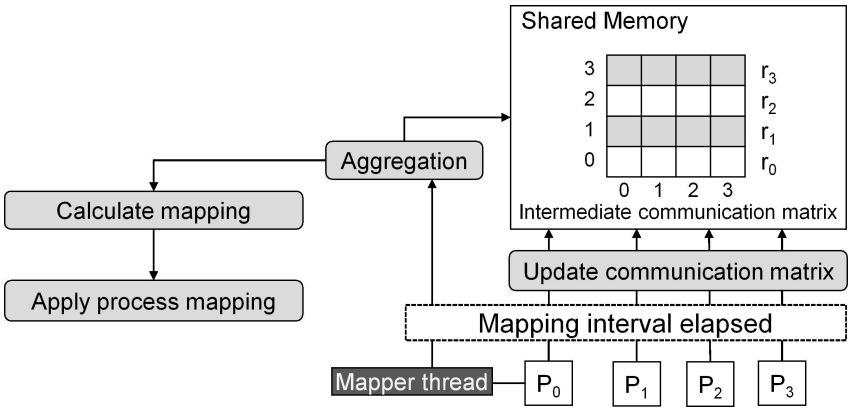



**Figure 4.** The consolidation mechanism with four MPI processes

the address space of another process. Thus, to determine the communication behavior of the application, it is necessary to consolidate the communication events among the processes.

The consolidation mechanism is shown in Fig. 4. For the consolidation purpose, we store an intermediate communication matrix in a shared memory region. Each row of this matrix has its own shared memory object, and thus the accesses to the matrix do not need to be locked since each row is updated only by one process. We use the POSIX shared memory API [22] to read and write the shared memory objects. However, the consolidation process may increase congestion on the shared region if a large number of processes frequently update the matrix. Thus, we also use the mapping interval to limit the frequency of updating the matrix.

During Step 2, each process updates its row in the intermediate matrix using the monitoring counters. The $i$-th row ($r_i$) is updated by the process with ID $i$ ($P_i$). However, the value of each counter is accumulated during the monitoring step. If the communication matrix is updated with the accumulated values, OnDeLoc-MPI may not be able to detect changes in the communication behavior because the previous communication behavior may significantly influence the current result of communication behavior. Thus, to reset the communication behavior, we update cell $(x, y)$ by subtracting the last value of the cell from the counter value for processes $x$ and $y$. The mapper thread generates a consolidated matrix by aggregating the data from all rows of the intermediate matrix. The generated communication matrix is then used as the input to the mapping algorithm.

### *2.2.3. OnDeLocMap+ algorithm*

The OnDeLocMap+ algorithm is depicted in Algorithm 1. We adopt the algorithm proposed in our previous work, called On-DeLoc [3], to implement OnDeLocMap+. A key difference between these two algorithms is that OnDeLocMap+ considers the previous mapping to calculate the current mapping. In the previous work, we have shown that the migration overhead has a significant impact on the performance results of On-DeLoc. To reduce this overhead, the OnDeLocMap+ algorithm prevents unnecessary process migrations by giving a higher priority to processes that have a higher amount of communication to be mapped to the same NUMA node of the previous mapping. We detail the difference between the two algorithms in the following description.

First, OnDeLocMap+ uses the topology model to construct the map between processor core IDs and process IDs (Line 1). The keys of the map represent the IDs of processor cores available

**Algorithm 1** The OnDeLocMap+ Algorithm.

**Input:** $T$ {The node topology tree}
**Input:** $A$ {The communication matrix}
**Input:** $PrevM$ {The previous mapping}
**Output:** $M$ {The map of processor core IDs and process IDs}

```
1:  M ← createMap(T)
2:  Pairs ← generatePairs(A)
3:  sortedPairs ← sortByAcomm(Pairs)
4:  i ← 0
5:  while i < num(Pairs) and numUnmappedCores(M) > 0 do
6:    current_pair ← sortedPairs[i]
7:    prev_nodes ← getPreviousNodes(current_pair, PrevM)
8:    if isNodesAvailable(prev_nodes) then
9:      mapPair(current_pair, prev_nodes)
10:   else
11:     mapPair(current_pair, nextNodeAvailable(T))
12:   end if
13:   i ← i + 1
14: end while
```

in the system, and each value represents the ID of the process mapped to the processor core of the key. At the beginning of the algorithm, all values are set to empty. Then, it generates pairs of processes from the communication matrix (Line 2). A pair of processes $x$ and $y$ is generated for each matrix cell $(x, y)$ of the matrix. The algorithm then selects a process pair that has not been mapped to processor cores sequentially from the pairs with the highest to the lowest amount of communication. This selection is achieved by the sorting step in the algorithm (Line 3). A NUMA node is available for the mapping if it has one or more unmapped cores. In On-DeLoc, the *mapPair*() function always maps the processes to the processor cores of the NUMA node that is currently available in a round-robin fashion (Line 11). We aim to improve the locality by mapping two processes of a pair to the same NUMA node, while also reducing the memory congestion by mapping different pairs to the different NUMA nodes. However, in contrast to On-DeLoc, when selecting the target NUMA nodes for a process pair, OnDeLocMap+ first evaluates the NUMA nodes that have been previously mapped for the same pair (Lines 7–8). The function *getPreviousNodes*() will return two NUMA nodes, where each node is associated with each process of the pair. If the previous NUMA nodes are available, it will map each process of the pair to the processor cores of the previous NUMA node associated with the process (Line 9) so that each process of the pair will not be migrated to a different NUMA node. Otherwise, OnDeLocMap+ will map the pair to the processor cores in a round-robin fashion, the same way as the previous algorithm.

### *2.2.4. Data structures*

For each MPI application, we allocate two arrays to store the two communication matrices. The first array is to store the consolidated communication matrix, and the other array is to store the intermediate matrix. Since the matrix is a square matrix of order $N_p$, the size of each

matrix scales quadratically with the number of MPI processes. The size of each matrix cell is 4 bytes, and thus the total size of the memory used for all the communication matrices is $(N_p^2 \times 2 \times 4)$ bytes. In addition to the communication matrices, we allocate a key-value map to store the previous mapping, where each element of the map consists of a processor core ID and its associated process ID. The size of the map scales linearly with $N_p$. The size of each element is 8 bytes, and thus the total size of the memory allocated for the key-value map is $(N_p \times 8)$ bytes.

# 3. Experimental Evaluation

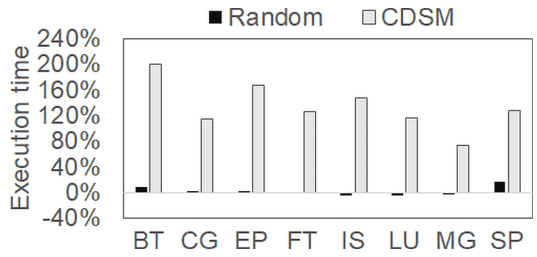
In this section, we present the experimental evaluation of OnDeLoc-MPI. Our main experimental results on a NUMA system consist of three parts: performance, energy consumption, and overhead. In addition, we provide and discuss our evaluation results with a larger NUMA system.

## 3.1. Experimental Setup

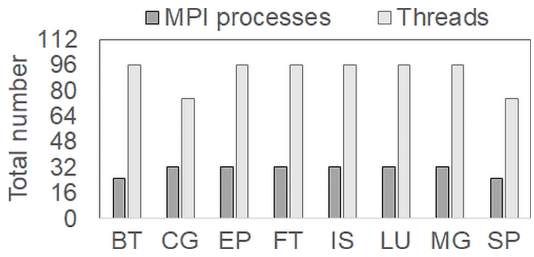
The main experiments have been conducted on a NUMA system, named Xeon2, that consists of two NUMA nodes and one Intel Xeon E5-2690 processor per NUMA node. The system has 32 logical cores in total, and it is running Linux OS kernel v3.2. The NUMA nodes are connected with QuickPath Interconnect (QPI) [32], and each NUMA node has 16 logical cores and an Integrated Memory Controller (IMC). Open MPI v3.1 is used as the MPI runtime system for the experiments. As workloads, we used eight MPI applications of the NAS Parallel Benchmarks (NPB) [4] v3.4 with the class C problem size. All the applications except BT and SP are executed with 32 processes using all cores in the system. BT and SP require a square number of processes, and thus we execute these two applications with 25 processes.

In all the experiments, we keep the mapping interval higher than or equal to 500 ms to limit the overhead. The parameter $v$ is set equal to 2, and this value is chosen empirically from experiments with the NPB applications. We are aware that in parallel applications that change their communication behavior during their execution, this parameter will affect the performance results of our method. However, to determine the optimal value of $v$ for a particular application, it is necessary to analyze its temporal communication behavior prior to the execution of the application. We avoid this analysis step because it will incur a high overhead [30]. In our future work, we will investigate the impacts of this parameter on the performance and overhead of our method.

We compare OnDeLoc-MPI with an online-based thread mapping method and two static mapping methods. The static mapping methods are Default and Static-best. Default is the original mapping of the MPI runtime system that maps the neighboring processes to processor cores of different NUMA nodes in a round-robin fashion. It represents the baseline of our experiments. Static-best mapping is obtained by using an offline-based method proposed in our previous work [2]. It first collects communication traces by preliminary running the target application. Then, it analyzes the communication behavior of the application and calculates the mapping using the CLB algorithm. We also evaluated the Treematch [21] algorithm to calculate the mapping. Since CLB shows lower execution times than Treematch, we show only the results for CLB. Note that Static-best mapping works offline prior to the execution of an application, and thus has a high overhead caused by the preliminary run and offline analysis.




(a) Performance results of CDSM, normalized to the baseline




(b) The total number of threads compared with the number of MPI processes

**Figure 5.** Evaluation results with CDSM

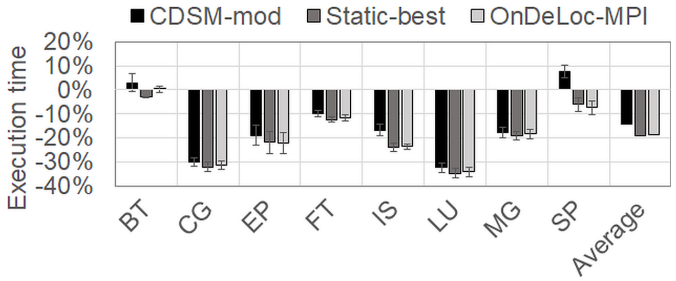


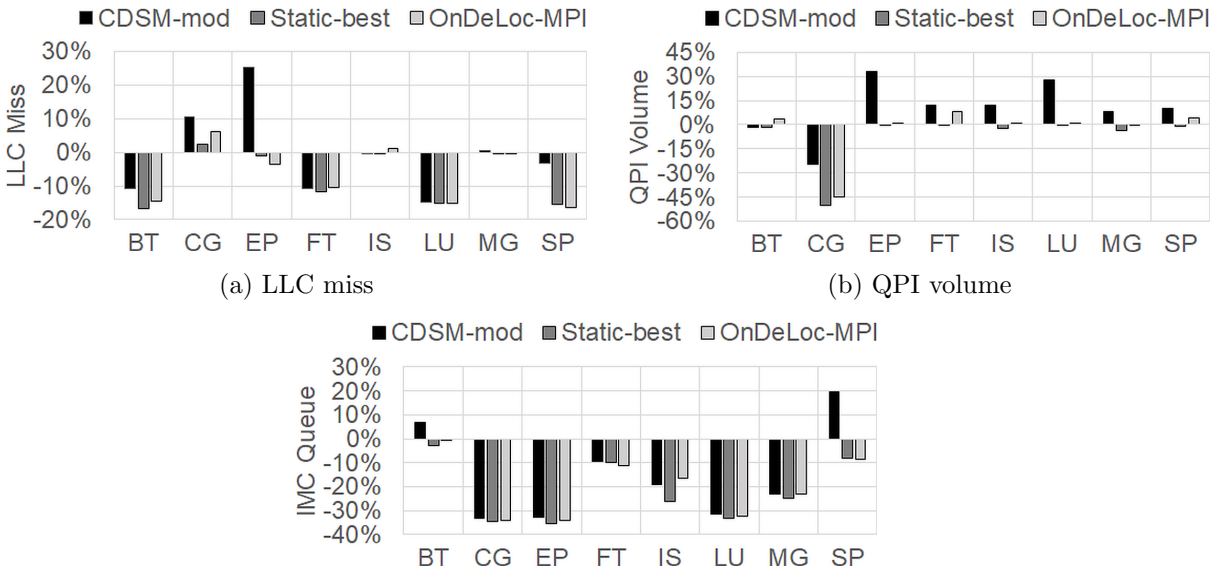

**Figure 6.** Performance results on Xeon2, normalized to the baseline

The online-based mapping method used for the evaluation is CDSM-mod, which is a modified version of CDSM. We modify CDSM because our experimental results show that it significantly degrades the performance of all the tested applications, which are contrast with the results shown in the related work [13]. Figure 5(a) shows the performance results of CDSM, compared with the baseline and a random mapping method. For the random mapping, we randomly generate a process mapping before each execution. As shown in the figure, CDSM shows higher execution times for all the applications. In BT, CDSM can increase the execution time by a factor of two compared with Default. We have observed that the performance degradation is caused by the inaccuracy of detecting the communication events among MPI processes. CDSM detects the communication events by analyzing the page faults of all threads of the application. However, in multithreaded MPI implementations, an MPI process can spawn multiple threads [14]. Thus, the total number of threads spawned by the runtime system can be higher than the number of MPI processes.

Figure 5(b) shows the total number of threads executed for the NPB applications with 32 number of MPI processes on Xeon2. Although the number of MPI processes is less than or equal to the number of processor cores of the system, the total number of threads executed during the execution is substantially higher than the number of cores and MPI processes. By detecting the communication among all threads of the application, CDSM cannot accurately detect the communication among MPI processes. To increase the accuracy, we modify the method to only include the parent threads of the MPI processes in the steps of detecting communication and calculating the thread mapping. We detect these parent threads from the first $n$ threads created at the beginning of the execution, where $n$ is equal to $N_p$.

## 3.2. Performance Evaluation

Figure 6 shows the performance results on the Xeon2 system. We measure the execution time of the applications with each mapping method. All the experimental results are the averages




(a) LLC miss

(b) QPI volume

(c) IMC queue

**Figure 7.** Performance monitoring results, normalized with the baseline

obtained from 10 sample executions, which are normalized to the results of the baseline method. We also provide the 95% confidence interval calculated with Student's t-distribution. The error line of the bar represents the confidence intervals of the samples.

On average, OnDeLoc-MPI shows higher performance improvements than those of Default and CDSM-mod. Compared with Default, the average performance improvement of OnDeLoc-MPI is 18.5%. The highest performance improvements are exhibited in CG and LU by 31.2% and 34%, respectively. CDSM-mod shows performance improvements from Default for most of the applications, indicating that our modification to CDSM effectively increases the communication detection accuracy, and thus it can increase the performance of the applications.

For all the applications except EP and SP, Static-best shows the highest improvements. However, the average improvement of OnDeLoc-MPI is only 0.5% lower than that of Static-best, which means that our method can achieve performance close to that of Static-best even without any kind of extensive profiling and analysis. Moreover, in SP, OnDeLoc-MPI achieves the highest performance improvement among the methods. These results indicate that in the case of SP, the static mapping is not sufficient to take into account the temporal changes of the communication behavior.

To investigate the sources of performance improvements, we evaluate the performance characteristics of the NPB applications. We use LLC misses, QPI volume and IMC queue metrics for the evaluation. These metrics are obtained by monitoring the Intel performance counters [20]. LLC misses represent the number of last-level cache misses across all NUMA nodes. IMC queue is the total queuing time of memory accesses in the memory controllers. A higher value of this metric indicates a longer queuing delay caused by the memory congestion. QPI volume is the total volume of data sent through interconnect links. A higher value of this metric indicates longer latencies from the remote memory accesses.

Figures 7(a), 7(b) and 7(c) show the results of last-level cache misses, QPI volume and IMC queue, respectively. These figures show that most of the applications gain a substantial performance improvement from reductions in the caches misses and IMC queuing delay. It means that the memory congestion has a significant impact on the performance of most of

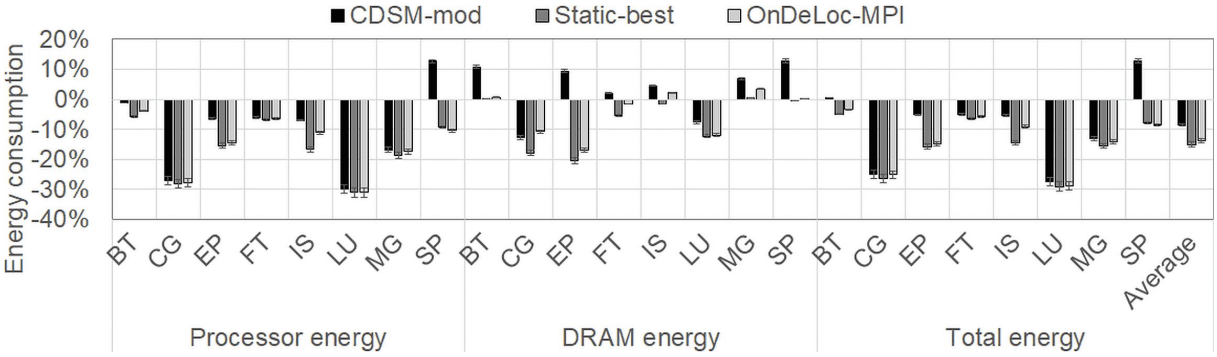



**Figure 8.** Energy consumption results on Xeon2, normalized with the baseline

the NPB applications. Moreover, in BT and SP, CDSM-mod increases the IMC queuing delay, and in most of the applications, it shows a higher IMC queue than those of OnDeLoc-MPI and Static-best. This fact suggests that only considering the locality is not sufficient to achieve the best performance for these applications. In the cases of BT and SP, the performance differences among the methods are smaller than that in the other applications. It is because, as shown in our previous work [3], these two applications have the communication behavior that can benefit from the Default mapping. In these two applications, most communication events are performed by the neighboring processes, and thus the Default mapping is sufficient to improve the performance of these applications.

In most of the applications, CDSM-mod shows a higher QPI volume (Fig. 7(b)) and a longer execution time (Fig. 6) compared with Static-best and OnDeLoc-MPI. By migrating the processes during the execution, CDSM-mod and OnDeloc-MPI potentially increase data traffic on interconnects because the migrated processes may need to access data that reside in a remote NUMA node. However, the performance improvements achieved by OnDeLoc-MPI are close to those of Static-best. Furthermore, even for the applications that cannot gain a significant performance improvement from Static-best mapping, such as BT, OnDeLoc-MPI does not reduce the performance of the applications. These results show that the migration overhead in online-based mapping methods can have a significant impact on the execution time. However, our method can effectively reduce this overhead.

### 3.3. Energy Consumption Evaluation

In this section, we discuss the energy consumption of the NPB applications on the Xeon2 system. We measure the processor energy and DRAM energy by using the Running Average Power Limit (RAPL) hardware counters [11]. As shown in the performance monitoring results, the process mapping methods have a significant impact on the cache misses, the interconnect traffic, and queuing delay in memory controllers. Therefore, the mapping methods will affect the energy consumption of not only the processor cores but also the interconnects and memory controllers. In the Intel processor used for the evaluation, each package of processor consists of core and Uncore components. Uncore refers to components that are apart from processor core, which include QPI and memory controllers [19]. To evaluate the energy consumption of processor core and Uncore components, we decompose processor energy consumption into core and Uncore energy consumptions. We measure core and Uncore energy also using the RAPL counters.

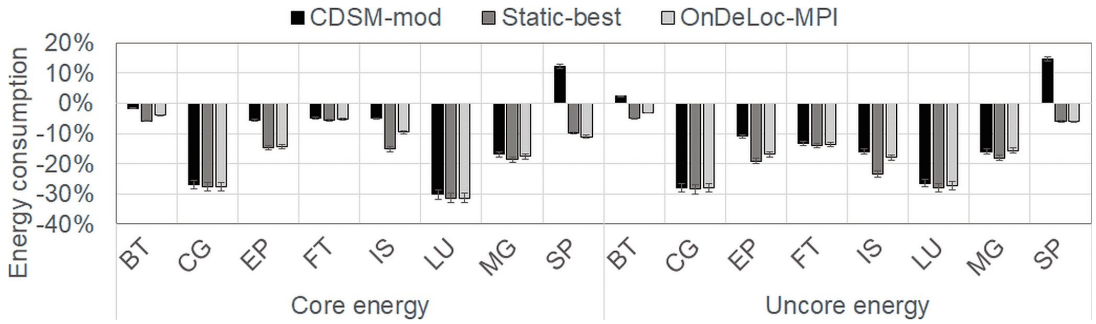


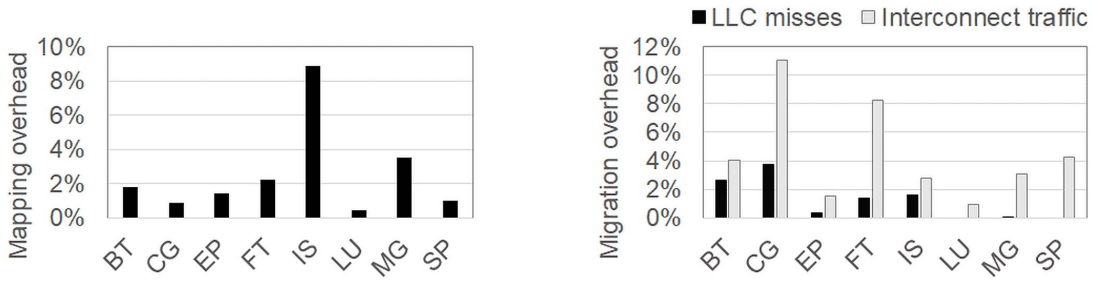

**Figure 9.** Core and Uncore energy consumptions on Xeon2, normalized with the baseline

Figure 8 shows the results of processor energy and DRAM energy for each mapping method on Xeon2. In all the applications, OnDeLoc-MPI shows a lower total energy consumption than those of Default and CDSM-mod. On average, the total energy is reduced by 13.6% compared with Default, and the highest reduction is 28.9% in the case of LU. In some applications, such as IS and MG, OnDeLoc-MPI and CDSM-mod increase DRAM energy. It is because these two online mapping methods use more DRAM to analyze the communication behavior and calculate the mapping. However, the increase in DRAM energy is relatively small compared with the decrease in processor energy. Since the total energy is mostly contributed by the processor energy, OnDeLoc-MPI achieves lower total energy consumption than the baseline in all the applications.

Figure 9 shows the results of core and Uncore energy consumptions. In all the NPB applications, OnDeLoc-MPI show lower core and Uncore energy consumptions than those of CDSM-mod and Default. The core energy consumption is reduced most in LU, with a reduction of 31.5%, and the Uncore energy consumption is reduced most in CG, with a reduction of 27.9%. On average, the core energy is reduced by 15% in OnDeLoc-MPI and 16.1% in Static-best, while Uncore energy is reduced by 16% and 17.7%, respectively.

In most of the applications, CDSM-mod shows higher core and Uncore energy consumptions than Static-best and OnDeLoc-MPI. The increases in core and Uncore energy is caused by the increases in execution time and interconnect traffic, respectively. Moreover, in BT and SP, CDSM-mod shows the highest Uncore energy because, as shown in Fig. 7(b) and 7(c), it also increases the queuing time in the memory controllers. These results show that compared to Default and CDSM-mod, OnDeLoc-MPI is more effective in reducing the energy consumption of interconnects and memory controllers.

In most of the applications, the lower execution time leads to the lower core and Uncore energy consumptions, indicating that the energy reductions are mainly contributed by the reduction in execution time. By reducing the execution time, OnDeLoc-MPI, CDSM-mod, and Static-best reduce the static energy consumption in most of the applications. However, as shown in BT and SP, OnDeLoc-MPI can achieve lower reductions in core and Uncore energy consumptions than those of Default and CDSM-mod with no significant differences in the execution time. This fact shows that OnDeLoc-MPI also reduces the dynamic energy consumption by reducing the number of cache misses and queuing time in the memory controllers.




(a) Mapping overhead

(b) Migration overhead

**Figure 10.** Overhead of OnDeLoc-MPI

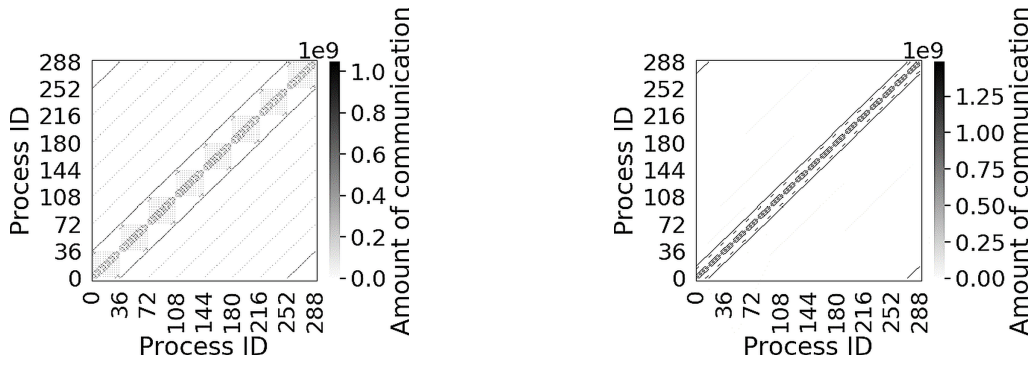
### 3.4. Overhead of OnDeLoc-MPI

OnDeLoc-MPI incurs overhead on the execution of an MPI application because it works during the application runtime. The overhead is caused by the computation of the mapping and the migration of processes. The mapping overhead consists of the repeated accesses to the intermediate and consolidated communication matrices and the execution of the mapping algorithm, while the migration overhead consists of increases in cache misses and interconnect traffic for the process after migration. In this section, we evaluate the overhead of OnDeLoc-MPI on the Xeon2 system.

The mapping overhead is shown in Fig. 10(a). It is evaluated by measuring the time in the function that accesses the communication matrices and to calculate the mapping. The values are the percentage of the execution time of each application. For all the applications, the mapping overhead is less than 9%, and the average overhead of the mapping is low, which is 2.5%. IS shows the highest mapping overhead because its execution time is much shorter than those of the other applications, and thus, the ratio of the mapping overhead to the execution time is higher than those of the other applications. However, the time used in IS for updating the communication matrices and calculating the mapping is less significant compared with the other applications. The results of mapping overhead show that the dynamic adjustment of the mapping interval can effectively reduce the overhead.

The migration overhead is evaluated by comparing the performance monitoring results of Static-best mapping and OnDeLoc-MPI for each application. However, in this evaluation, we disable the functions of OnDeLoc-MPI that repeatedly update the communication matrix and compute the mapping. The process mapping for each interval is provided offline prior to the execution. We obtain the mapping of each interval by preliminary running each application with OnDeLoc-MPI. Thus, in this evaluation, only the migration overhead affects the execution of each application.

Figure 10(b) shows the migration overhead on the cache misses and interconnect traffic. We obtain cache misses by aggregating the cache misses of all cache levels across the NUMA nodes. For all the applications, the migration overhead is less than 11%, and the highest overheads on the interconnect traffic are imposed in CG and FT. As shown in our previous work [3], CG has a wide variation of the amount of communication among processes. Moreover, these two applications have a high number of memory accesses. Thus, migrating a process potentially increases the amount of remote accesses because the process may need to access data that reside in the previous NUMA node. BT and CG show the highest overhead to last-level cache misses,




(a) Test Case A (b) Test Case B

**Figure 11.** The communication behaviors of the GROMACS applications

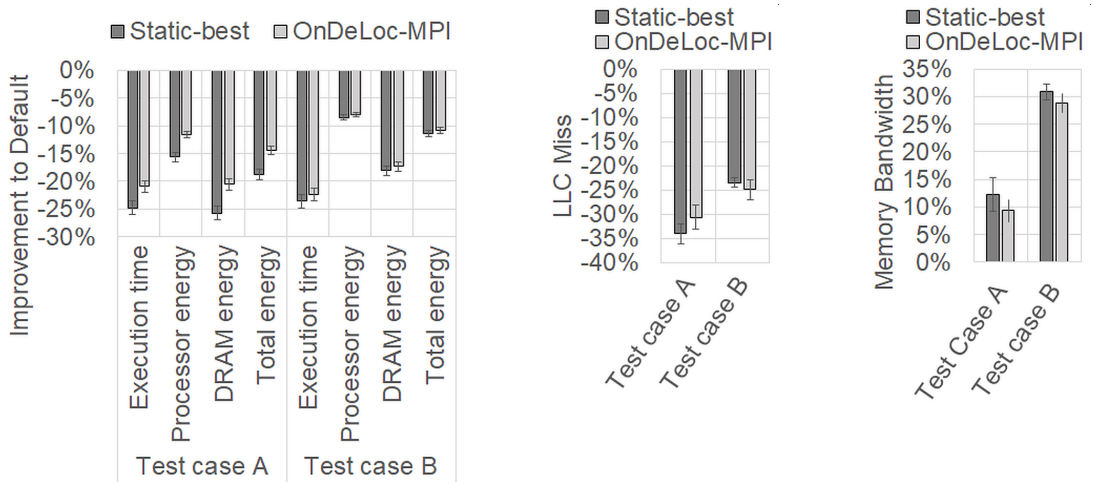
indicating that these two applications access cache memories more than the other applications. For the other applications, the migration overheads are small because, in these applications, the process mapping is more stable than those of CG and FT. OnDeLoc-MPI performs less migration during the execution of these applications.

The performance and overhead results show that the migration overhead mainly causes the performance differences between OnDeLoc-MPI and Static-best mapping. This overhead is affected by the numbers of memory accesses and changes in the communication behavior of the application. During the execution, some applications, such as BT, CG and FT access memory devices and change their communication behavior more frequently than the other applications. The migration overheads of OnDeLoc-MPI in these applications are the highest among the applications, and thus OnDeLoc-MPI shows a lower performance improvement compared with Static-best mapping. On the other hand, it shows lower migration overheads in the other applications, and thus OnDeLoc-MPI can achieve a comparable performance with Static-best in the other applications. Moreover, in SP, the benefit of online mapping surpasses the overhead of OnDeLoc-MPI, and thus it can achieve a higher performance than that of Static-best.

### 3.5. Evaluation on a Larger System

To evaluate our method with larger number of NUMA nodes and processor cores, we have conducted experiments on a NUMA system, named KNL4, which is based on Intel Xeon Phi Knights Landing (KNL) processor [28]. The system has 288 logical cores in total, and it is running Linux kernel v4.4. We configure the system as a four-node NUMA system by setting Sub-NUMA clustering (SNC) mode as the clustering mode of the KNL. In this cluster mode, the system is partitioned into four NUMA nodes, with 72 logical cores per NUMA node. We also use Open MPI v3.1 as the MPI runtime system for this evaluation.

In this evaluation, we use two biomolecular applications of the UAEBS workloads [27, 33]. The applications are Test Case A and Test Case B of the workloads that are based on GROMACS simulation [1]. The communication behaviors of these applications are shown by the communication matrices in Fig. 11. The darker cells indicate a higher amount of communication. We obtain these matrices from the last mapping result of OnDeLoc-MPI during the execution of these applications. These two applications are executed with 288 number of MPI processes to use all the processor cores of the system. We do not use the NPB applications because all the applications except EP cannot be executed with this number of processes. OnDeLoc-MPI is compared with




(a) Performance and energy (b) LLC misses (c) Memory bandwidth

**Figure 12.** Performance and energy consumption results on KNL4, normalized with the baseline

Default and Static-best mapping. We cannot compare our method with CDSM-mod because of its limitation to the previous version of Linux kernel. This fact highlights the advantage of implementing our method in the runtime system. which is that OnDeLoc-MPI does not depend on specific features of the hardware or operating system.

Figure 12(a) shows the results of performance and energy consumption on the KNL4 system. We cannot provide the Uncore energy in this evaluation because of the limitation of the RAPL counters on the KNL. However, we can still evaluate the impacts of the mapping methods on the total energy consumption by measuring processor and DRAM energy. As shown in the figure, OnDeLoc-MPI reduces the execution times and energy consumptions in both applications. Compared to Default, the execution time is reduced most in Test Case B, with a reduction of 22.3%, while the highest total energy reduction is shown in Test Case A, with a reduction of 14.3%. On average, the execution time and total energy consumption are reduced by 21.6% and 12.6%, respectively.

As shown in the figure, OnDeLoc-MPI shows lower improvements compared with Static-best. However, the performance and energy consumption results of these two methods are close to each other. The differences of execution time are 3.91% in Test Case A, and 1.24% in Test Case B, while the processor energy differences are 3.98% and 0.47%, respectively. The differences in Test Case A are higher than those in Test Case B because the communication behavior of Test Case A is more irregular than that of Test Case B. As shown in their communication matrices, the number of tasks that perform substantial amount of communication in Test Case A is higher than that in Test Case B. OnDeLoc-MPI updates the process mapping more frequently in Test Case A. The performance and energy consumption results of these two applications suggest that the overhead of OnDeLoc-MPI is still low even if the number of processes and NUMA nodes becomes larger.

For further evaluation, we measure the last-level cache misses and memory bandwidth metrics on the KNL4 system. Memory bandwidth is the bandwidth of memory accesses to the memory controllers. We cannot measure the QPI volume and IMC queue metrics because of the limitation of the monitoring counters on the KNL. However, the memory bandwidth metric can show the impacts of the mapping methods on the memory congestion because higher memory congestion leads to lower memory access bandwidth.

Figures 12(b) and 12(c) show the results of cache misses and memory bandwidth, respectively. These results suggest that both OnDeLoc-MPI and Static-best gain performance and energy improvements by reducing last-level cache misses and memory congestion. Compared with Default, OnDeLoc-MPI shows lower cache misses and higher memory bandwidth for both applications. The highest reduction of cache misses is 30% with Test Case A, while the highest improvement of memory bandwidth is 28.8% with Test Case B. Compared with Static-best, OnDeLoc-MPI shows higher cache misses in Test Case A, and shows lower memory bandwidth for both applications. However, the differences of the results between the two methods are small. The differences of cache misses are 3.37% in Test Case A, while the memory bandwidth differences are 2.9% and 2.1% in Test Case A and Test Case B, respectively.

## Conclusions and Future Work

In this paper, we have proposed a process mapping method, called OnDeLoc-MPI, to address the locality and the memory congestion problems on NUMA systems. Our method works online during the execution of an MPI application, and dynamically performs the process mapping to adapt to changes in the communication behavior of the application. In contrast to the related work, OnDeLoc-MPI does not need offline profiling to analyze the communication behavior of the application, and does not rely on communication detection mechanisms and specific features of the hardware or operating system. Alternatively, it analyzes the communication behavior by monitoring the MPI communication events during the execution of the application.

OnDeLoc-MPI has been evaluated on a real NUMA system with a set of NAS parallel benchmarks. On average, it can achieve performance and energy improvements close to the best static method with low overhead. Compared with the default mapping of the MPI runtime system, the performance and total energy improvements are up to 34% (18.5% on average), and 28.9% (13.6% on average), respectively. In addition, OnDeLoc-MPI has been evaluated on a larger NUMA system with two GROMACS applications. On the larger system, it achieves performance and energy improvements up to 22.3% and 14.3%, respectively. Our evaluation results have shown that the performance and energy improvements are obtained from reductions in cache misses, interconnect traffic, and queuing delay in memory controllers.

During the execution of an MPI application, OnDeLoc-MPI imposes overhead from the mapping calculation and process migration. To reduce the overhead, OnDeLoc-MPI employs mechanisms to prevent unnecessary process migrations and automatically adjust the mapping interval. The evaluation results show that the mechanisms can effectively reduce the overhead. The mapping overhead to the execution time is less than 9%, and the migration overhead to interconnect traffic and cache misses is less than 11%.

As future work, we intend to evaluate the impacts of our method on the performance and energy consumption of a large cluster of NUMA systems. For this future work, we will investigate the overhead of migrating processes between systems, and the impacts of parameter $v$ for different applications. We also intend to extend to our method for parallel applications with hybrid programming of MPI and multithreading, such as OpenMP and Pthreads.

## Acknowledgements

This work is partially supported by Japan's Ministry of Education, Culture, Sports, Science and Technology (MEXT) Next Generation High-Performance Computing Infrastructures and

Applications R&D Program "R&D of A Quantum-Annealing-Assisted Next Generation HPC Infrastructure and its Applications" and Grant-in-Aid for Scientific Research(B) #16H02822 and #17H01706.



## References

1. Abraham, M.J., Murtola, T., Schulz, R., et al.: GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX 1-2, 19–25 (2015), DOI: 10.1016/j.softx.2015.06.001

2. Agung, M., Amrizal, M.A., Komatsu, K., et al.: A memory congestion-aware MPI process placement for modern NUMA systems. In: 2017 IEEE 24th International Conference on High Performance Computing, HiPC, 18-21 Dec. 2017, Jaipur, India. pp. 152–161. IEEE (2017), DOI: 10.1109/HiPC.2017.00026

3. Agung, M., Amrizal, M.A., Egawa, R., et al.: An automatic MPI process mapping method considering locality and memory congestion on NUMA systems. In: 2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip, MCSoC, 1-4 Oct. 2019, Singapore. pp. 17–24. IEEE (2019), DOI: 10.1109/MCSoC.2019.00010

4. Bailey, D., Barszcz, E., Barton, J., et al.: The NAS Parallel Benchmarks. Int. J. High Perform. Comput. Appl. 5(3), 63–73 (1991), DOI: 10.1177/109434209100500306

5. Barak, A., Margolin, A., Shiloh, A.: Automatic resource-centric process migration for MPI. In: Träff, J.L., Benkner, S., Dongarra, J.J. (eds.) Recent Advances in the Message Passing Interface, 23-26 Sep. 2012, Vienna, Austria. pp. 163–172. Springer, Berlin, Heidelberg (2012), DOI: 10.1007/978-3-642-33518-1_21

6. Bosilca, G., Foyer, C., Jeannot, E., et al.: Online Dynamic Monitoring of MPI Communications. In: European Conference on Parallel Processing, 28 Aug-1 Sep. 2017, Santiago de Compostela, Spain. pp. 49–62. Springer, Cham (2017), DOI: 10.1007/978-3-319-64203-1_4

7. Broquedis, F., Clet-Ortega, J., Moreaud, S., et al.: Hwloc: A generic framework for managing hardware affinities in HPC applications. In: 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, 17-19 Feb. 2010, Pisa, Italy. pp. 180–186. IEEE (2010), DOI: 10.1109/PDP.2010.67

8. Buntinas, D., Mercier, G., Gropp, W.: Implementation and shared-memory evaluation of MPICH2 over the Nemesis communication subsystem. In: Mohr, B., Träff, J.L., Worringen, J., Dongarra, J. (eds.) Recent Advances in Parallel Virtual Machine and Message Passing Interface, 17-20 Sept. 2006, Germany. pp. 86–95. Springer, Berlin, Heidelberg (2006), DOI: 10.1007/11846802_19

9. Chen, H., Chen, W., Huang, J., et al.: MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters. In: Proceedings of the 20th Annual

International Conference on Supercomputing, 28 June-1 July, 2006, Cairns, Queensland, Australia. pp. 353–360. ACM (2006), DOI: 10.1145/1183401.1183451

10. Dashti, M., Fedorova, A., Funston, J., et al.: Traffic management: A holistic approach to memory placement on NUMA systems. In: Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems, Houston, Texas, USA. pp. 381–394. ACM, New York, NY, USA (2013), DOI: 10.1145/2451116.2451157

11. David, H., Gorbatov, E., Hanebutte, U.R., et al.: RAPL: Memory power estimation and capping. In: 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design, 18-20 Aug. 2010, Austin, TX, USA. pp. 189–194. IEEE (2010), DOI: 10.1145/1840845.1840883

12. Diener, M., Cruz, E.H., Alves, M.A., et al.: Affinity-based thread and data mapping in shared memory systems. ACM Computing Surveys 49(4), 64 (2017), DOI: 10.1145/3006385

13. Diener, M., Cruz, E.H., Navaux, P.O., et al.: Communication-aware process and thread mapping using online communication detection. Parallel Comput. 43(C), 43–63 (2015), DOI: 10.1016/j.parco.2015.01.005

14. Dózsa, G., Kumar, S., Balaji, P., et al.: Enabling concurrent multithreaded MPI communication on multicore petascale systems. In: Proceedings of the 17th European MPI Users' Group Meeting Conference on Recent Advances in the Message Passing Interface, 12-15 Sept. 2010, Stuttgart, Germany. pp. 11–20. Springer-Verlag, Berlin, Heidelberg (2010), DOI: 10.1007/978-3-642-15646-5_2

15. Gabriel, E., Fagg, G.E., Bosilca, G., et al.: Open MPI: Goals, concept, and design of a next generation MPI implementation. In: European Parallel Virtual Machine/Message Passing Interface Users Group Meeting, 19-22 Sept. 2004, Budapest, Hungary. pp. 97–104. Springer, Berlin, Heidelberg (2004), DOI: 10.1007/978-3-540-30218-6_19

16. Gaud, F., Lepers, B., Funston, J., et al.: Challenges of memory management on modern NUMA systems. Commun. ACM 58(12), 59–66 (2015), DOI: 10.1145/2814328

17. Goglin, B., Moreaud, S.: KNEM: A generic and scalable kernel-assisted intra-node MPI communication framework. Journal of Parallel and Distributed Computing 73(2), 176–188 (2013), DOI: 10.1016/j.jpdc.2012.09.016

18. Gropp, W.: MPICH2: A new start for MPI implementations. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface, 19-22 Sept. 2004, Budapest, Hungary. pp. 7–7. Springer, Berlin, Heidelberg (2002), DOI: 10.1007/3-540-45825-5_5

19. Hofmann, J., Fey, D., Eitzinger, J., et al.: Analysis of Intel's Haswell Microarchitecture Using the ECM Model and Microbenchmarks. In: Architecture of Computing Systems, ARCS 2016, 4-7 April 2016, Nuremberg, Germany. pp. 210–222. Springer, Cham (2016), DOI: 10.1007/978-3-319-30695-7_16

20. Intel: Intel Xeon Processor E5 and E7 v4 Product Families Uncore Performance Monitoring Reference Manual. `https://www.intel.com/content/www/us/en/products/docs/processors/xeon/xeon-e5-e7-v4-uncore-performance-monitoring.html` (2016)

21. Jeannot, E., Mercier, G., Tessier, F.: Process placement in multicore clusters:algorithmic issues and practical techniques. IEEE Transactions on Parallel and Distributed Systems 25(4), 993–1002 (2014), DOI: 10.1109/TPDS.2013.104

22. Kerrisk, M.: Linux/UNIX System Programming: POSIX Shared Memory. http://man7.org/training/download/posix_shm_slides.pdf (2015), accessed: 2019-05-14

23. Lepers, B., Quéma, V., Fedorova, A.: Thread and memory placement on NUMA systems: Asymmetry matters. In: Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference, 8-10 July 2015, Santa Clara, CA. pp. 277–289. Berkeley, CA, USA (2015), DOI: 10.5555/2813767.2813788

24. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. http://www.mpi-forum.org (2012)

25. Molka, D., Hackenberg, D., Schöne, R.: Main Memory and Cache Performance of Intel Sandy Bridge and AMD Bulldozer. In: Proceedings of the Workshop on Memory Systems Performance and Correctness, Edinburgh, United Kingdom. pp. 4:1–4:10. ACM, New York, NY, USA (2014), DOI: 10.1145/2618128.2618129

26. Orduña, J.M., Silla, F., Duato, J.: On the development of a communication-aware task mapping technique. J. Syst. Archit. 50(4), 207–220 (2004), DOI: 10.1016/j.sysarc.2003.09.002

27. PRACE: Unified European Applications Benchmark Suite. www.prace-ri.eu/ueabs (2013), accessed: 2019-10-01

28. Sodani, A.: Knights landing (KNL): 2nd Generation Intel Xeon Phi processor. In: 2015 IEEE Hot Chips 27 Symposium, 22-25 Aug. 2015, Cupertino, CA, USA. pp. 1–24. IEEE (2015), DOI: 10.1109/HOTCHIPS.2015.7477467

29. Treibig, J., Hager, G., Wellein, G.: Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In: Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures, 13-16 Sept. 2010, San Diego, CA, USA. pp. 207–216. IEEE (2010), DOI: 10.1109/ICPPW.2010.38

30. Zhai, J., Sheng, T., He, J., et al.: Efficiently acquiring communication traces for large-scale parallel applications. IEEE Transactions on Parallel and Distributed Systems 22(11), 1862–1870 (2011), DOI: 10.1109/TPDS.2011.49

31. Zhang, J., Zhai, J., Chen, W., et al.: Process mapping for MPI collective communications. In: Euro-Par 2009 Parallel Processing, 25-28 Aug. 2009, Delft, The Netherlands. pp. 81–92. Springer, Berlin, Heidelberg (2009), DOI: 10.1007/978-3-642-03869-3_11

32. Ziakas, D., Baum, A., Maddox, R.A., et al.: Intel QuickPath Interconnect architectural features supporting scalable system architectures. In: 2010 18th IEEE Symposium on High Performance Interconnects, 18-20 Aug. 2010, Mountain View, CA, USA. pp. 1–6. IEEE (2010), DOI: 10.1109/HOTI.2010.24

33. Zivanovic, D., Pavlovic, M., Radulovic, M., et al.: Main memory in HPC: Do we need more or could we live with less? ACM Trans. Archit. Code Optim. 14(1), 3:1–3:26 (2017), DOI: 10.1145/3023362

DOI: 10.14529/jsfi200105

# Tools for GPU Computing – Debugging and Performance Analysis of Heterogenous HPC Applications

*Michael Knobloch*[1], *Bernd Mohr*[1]



General purpose GPUs are now ubiquitous in high-end supercomputing. All but one (the Japanese Fugaku system, which is based on ARM processors) of the announced (pre-)exascale systems contain vast amounts of GPUs that deliver the majority of the performance of these systems. Thus, GPU programming will be a necessity for application developers using high-end HPC systems. However, programming GPUs efficiently is an even more daunting task than traditional HPC application development. This becomes even more apparent for large-scale systems containing thousands of GPUs. Orchestrating all the resources of such a system imposes a tremendous challenge to developers. Luckily a rich ecosystem of tools exist to assist developers in every development step of a GPU application at all scales.

In this paper we present an overview of these tools and discuss their capabilities. We start with an overview of different GPU programming models, from low-level with CUDA over pragma-based models like OpenACC to high-level approaches like Kokkos. We discuss their respective tool interfaces as the main method for tools to obtain information on the execution of a kernel on the GPU. The main focus of this paper is on two classes of tools, debuggers and performance analysis tools. Debuggers help the developer to identify problems both on the CPU and GPU side as well as in the interplay of both. Once the application runs correctly, performance analysis tools can be used to pinpoint bottlenecks in the execution of the code and help to increase the overall performance.

*Keywords: performance analysis, debugging, GPU computing.*

## Introduction

General purpose GPUs are now ubiquitous in high-end supercomputing. With the rise of deep learning and the convergence of simulation-based HPC and AI, GPU computing took a major leap forward. All but one (the Japanese Fugaku system, which is based solely on ARM processors) of the announced (pre-)exascale systems contain vast amounts of GPUs that deliver the majority of the performance of these systems. Thus, GPU programming will be a necessity for application developers using high-end HPC systems. However, programming GPUs efficiently is an even more daunting task than traditional HPC application development. This becomes even more apparent for large-scale systems containing thousands of GPUs. Orchestrating all the resources of such a system imposes a tremendous challenge to developers. Besides GPUs other accelerators have been tried, the most prominent being Intels Xeon Phi as a many-core architecture and FPGAs. However, the Xeon Phi has been discontinued and FPGAs are only a niche solution for very specific workloads or research projects, but not (yet) ready for production use in general HPC.

NVIDIA GPUs power most of todays GPU-enabled supercomputers. 136 systems in the TOP500 list of November 2019[2] are equipped with NVIDIA GPUs, including the number one and two systems, the U.S.-based Summit [33] and Sierra [25] supercomputers. Thus, we put a strong focus on NVIDIA GPUs in this paper.

Tools have always been an integral part of the HPC software stack. Debuggers and correctness checker help application developers to write bug-free and efficient code. Code efficiency can

[1]Forschungszentrum Jülich GmbH, Jülich Supercomputing Center, Jülich, Germany

[2]`https://www.top500.org/list/2019/11/`

be improved by pinpointing bottlenecks with performance analysis tools. The tools community is working hard to provide tools that master the complexity of modern HPC systems [30], facing the same challenges when scaling up as the application developers themselves. Today, a rich ecosystem of tools exist to assist developers in every development step of a GPU application at all scales, from a workstation to a supercomputer.

In this paper we present an overview of these tools and discuss their capabilities. We present the currently dominant programming models for GPU computing and discuss their tool interfaces as the main method for tools to obtain information on the execution of a kernel on the GPU in section 1. Then we look into debuggers in section 2, which help to develop correct heterogenous applications that scale to several hundred or thousand of GPUs. Performance analysis tools, which help to use these resources efficiently, are discussed in section 3. Finally we conclude the paper and give an outlook on future developments in heterogenous supercomputing.

# 1. GPU Programming Models

For decades two programming paradigms dominated the HPC landscape – distributed-memory programming (inter-node) and shared-memory programming (intra-node). The main programming model for distributed-memory programming is MPI, the Message Passing Interface [28], which is used in virtually all HPC applications. MPI is a rather low-level interface, i.e. the user has to explicitly express the communication pattern and data transfers. Shared memory programming is mostly done via OpenMP [36], a directive-based API. For both MPI and OpenMP alternatives exist, like the PGAS (Partitioned Global Address Space) model for distributed memory or pthreads and TBB (Threading Building Blocks) for shared memory, but none come close in popularity to MPI and OpenMP.

With the advent of general purpose GPUs, things changed significantly. Now a new very powerful but also very complex architecture was thrown into the mix, yet on the other hand the old programming paradigms are still valid in order to create scaling HPC applications. There exist several programming models for GPUs, some are low-level like MPI, others are pragma-based like OpenMP. Some support only certain languages or specific vendor architectures, others are more open. So it is a challenge for an application developer to choose the right programming model for his application, but also for tools developers to choose which models to support. In this section we present various GPU programming models that suits different needs, CUDA and OpenCL as high-performance low-level interfaces, OpenACC and OpenMP as easy-to-use yet efficient directive-based approaches and KOKKOS and RAJA that aim for performance portability on a wide range of architectures. Where applicable we also give an introduction to the respective tools interface as the main source for tools to get information on the kernels running on the accelerator and the data transfers to and from the device.

## 1.1. CUDA

CUDA [32] is a parallel computing platform and programming model developed by NVIDIA for general computing on NVIDIA GPUs. It is a very low-level interface, i.e. the programmer has to specify every data movement and kernel launch explicitly. Given access to all hardware features of modern GPUs like Unified Memory, CUDA can yield the highest performance achievable on GPUs. However, this comes at the cost of a rather high development effort and non-portability. A rich set of libraries, both from NVIDIA directly and from third parties, are available for

CUDA, enabling developers to harness the power of CUDA without the need to deal with all the low-level details of the architecture. So far CUDA is the most popular programming model for GPU programming, thus most tools support CUDA to some extend. While CUDA itself is C++, CUDA bindings exist for many programming languages like C, Fortran (currently only for PGI compilers), Python and MATLAB.

#### *1.1.1. CUPTI – The CUDA Performance Tools Interface*

The NVIDIA CUDA Profiling Tools Interface (CUPTI) provides performance analysis tools with detailed information about how applications are using the GPUs in a system. CUPTI provides two simple yet powerful mechanisms that allow performance analysis tools to understand the inner workings of an application and deliver valuable insights to developers. The first mechanism is a callback API that allows tools to inject analysis code into the entry and exit point of each CUDA C Runtime (CUDART) and CUDA Driver API function. Using this callback API, tools can monitor an applications interactions with the CUDA Runtime and driver. The second mechanism allows performance analysis tools to query and configure hardware event counters designed into the GPU and software event counters in the CUDA driver. These event counters record activity such as instruction counts, memory transactions, cache hits/misses, divergent branches, and more. This enables automated bottleneck identification based on metrics such as instruction throughput, memory throughput, and more.

### 1.2. OpenCL, SYCL and oneAPI

The aim of OpenCL, the *O*pen *C*omputing *L*anguage, is to provide a vendor independent programming interface for all kinds of computing devices, from CPUs over GPUs to FPGAs. OpenCL is developed by the Khronos Group, an open industry consortium of over 100 leading hardware and software companies. OpenCL, like CUDA, is a low-level API where the kernels are written in the OpenCL C++ kernel language, a static subset of C++14.

To ease the development of heterogenous applications, the Khronos group developed SYCL as an abstraction layer build on the concepts, portability and efficiency of OpenCL. SYCL allows the developer to program on a higher level than OpenCL, while still having access to lower-level code. A lot of the boilerplate code of OpenCL is removed by SYCL and a single-source programming, where host and device code are contained in the same source file, is enabled.

The newest member in the OpenCL language space is Intel'soneAPI with DPC++ (*D*ata *P*arallel C++), which in turn is built upon SYCL. Due to its recent Beta release and the – at the time of writing – limited availability of hardware, the support of tools for oneAPI could not be evaluated for this paper. However, it is clear that the well-known Intel tools VTune and Advisor will have rich support for oneAPI. The most interesting and unique feature of the Intel Advisor will be an analysis of the potential gain of offloading a sequential code path to an accelerator.

It will be interesting to see how oneAPI will be adopted by the HPC community and how the tools support for SYCL and oneAPI develops. Codeplay, a compiler vendor and active part in the SYCL community, recently announced SYCL support for NVIDIA GPUs [38], which could dramatically increase the interest in SYCL as a portable API as it significantly increases to potential user-base.

#### 1.2.1. The OpenCL Profiling Interface

OpenCL provides a very basic interface to get profiling information on memory operations and kernel launches. If profiling is enabled, the function `clGetEventProfilingInfo` returns timing information of OpenCL functions that are enqueued as commands to a command-queue. The most interesting for performance analysis are the begin and end timestamps of kernel launches. The SYCL specification defines a similar profiling interface. However, most tools with OpenCL support use some form of library wrapping to obtain information of the OpenCL execution.

### 1.3. OpenACC

The OpenACC (Open ACCelerator) API [34] describes a collection of compiler directives to specify loops and regions of code to be executed in parallel on a multicore CPU, or to be offloaded and executed in parallel on an attached accelerator device, providing portability across operating systems, CPUs, and accelerators. With directives for C/C++ and Fortran, OpenACC covers the most important programming languages for HPC.

OpenACC eases the development of heterogenous applications as it relieves the user from explicit accelerator and data management as well as data transfers to and from the device. Data management is handled with the *data* construct, where *enter data* and *exit data* directives can be used to control data transfers between host and device. Two fundamental compute constructs, *kernels* and *parallel* can be used to offload the execution of code blocks to an accelerator.

While OpenMP is a prescriptive programming model, i.e. the developer explicitly states *how* to split the execution of loops, code regions and tasks among well-defined teams of threads, OpenACC is more descriptive model, telling the compiler *where* it is safe to parallelize loops or offload kernels and *what* data has to be transferred. This enables the compiler to perform more optimizations and generate faster code [43].

#### 1.3.1. OpenACC Profiling Interface

OpenACC provides a profiling interface for both profile and trace data collection. This interface provides callbacks that are triggered during runtime if specific events occur. Three types of events are supported: data events, launch events and other events. Data events cover the allocation/deallocation of memory on the accelerator as well as data transfers. Launch events trigger before and after a kernel launch operation. Other events include device initialization and shutdown as well as wait operations [10]. However, these events only give host-side information. For information on the device the respective tools interface of the backend has to be used.

### 1.4. OpenMP

OpenMP is a directive-based API already well known for shared-memory parallelization on CPUs which is easy to learn. It also offers a path to more portable GPU-accelerated applications. Like OpenACC, one of the goals of the OpenMP standard is to minimize the need for applications to contain vendor-specific statements. Thus, codes are portable across all supported GPU architectures.

Pragmas to offload work on general purpose GPUs have been introduced in OpenMP 4 [35], the OpenMP device constructs. The *target* construct is required to specify a region to be launched

on the device. *Target data* maps the variables on the device. The *teams* pragma inside target spawns the set of teams with multiple OpenMP threads. The *distribute* construct partitions the iterations and maps it to each team.

#### *1.4.1. The OpenMP Tools Interfaces*

Unlike the other programming interfaces, OpenMP since version 5 [36] provides two tools interfaces, OMPT for performance analysis tools and OMPD for debuggers [12].

OMPT [13] is a portable interface for both sampling-based and instrumentation-based performance analysis tools. Like the other tool interfaces, OMPT provides callbacks for defined OpenMP events, like the begin of a parallel region or the start of a offloaded kernel. It also maintains the tools data for OpenMP scopes and it provides signal-safe inquiry functions to get OpenMP runtime information. OMPT is intended for first-party tools, i.e. tools that are linked into or loaded from the OpenMP application.

OMPD, the OpenMP debugging interface, on the other hand is an interface for third-party tools, i.e. tools that live in a different process from the OpenMP application. This interface allows external tools to inspect the OpenMP state of a running program via callbacks. The debugger has no direct access to the OpenMP runtime, it interacts with it through the OMPD architecture and the OMPD interface is transparent to the OpenMP application. The OMPD library can be used to debug a running program as well as core files generated when the application aborted due to an error.

### 1.5. Kokkos and RAJA

As stated above, HPC programming models didn't change for a long time, which gave application developers some confidence that their application will perform on the next generation of machines. With an increased variability in architectures and programming models that does not hold any more. An application tuned for a specific platform could perform badly on the next system, which could be completely different from the current one. Further, applications and libraries that are used universally need some kind of assurance to perform well on a wide range of architectures.

In the scope of the Exascale Computing Project [29], two projects emerged that strive for performance portability by providing an abstraction layer over the existing programming models. Both originate from US national laboratories, one is Kokkos [11], developed at Sandia, and the other one RAJA [2] from LLNL. The abstraction layers include memory and execution spaces, data layout (i.e. the data layout might change depending on the architecture the application is compiled for) and parallel execution.

Both Kokkos and RAJA currently provide only C++ interfaces and only have a CUDA backend for offloading work to a GPU, though support for other programming models is likely to follow.

#### *1.5.1. The Kokkos Profiling Interface*

Kokkos provides a set of hooks for profiling libraries to interface with the Kokkos runtime [19]. These hooks can be implemented in the form of callbacks within a shared library. Upon start of the application, the Kokkos runtime loads the library, checks for implemented callbacks, and invokes the performance monitor via corresponding callbacks. Currently Kokkos

supports callbacks for initialization and finalization of the runtime, deep data copies, and the three parallel execution models `parallel_for`, `parallel_reduce`, and `parallel_scan`. Similar to the OpenACC profiling interface only events on the host are triggered, though device events can be captured with CUPTI. RAJA unfortunately does not provide a profiling interface at this time.

## 2. Debuggers

Developing correct parallel programs is already a daunting task, adding the complexity of GPUs to the mix makes that endeavour even harder. This holds especially when using low-level programming paradigms, where the user is responsible for correct memory management and data movement. Luckily, several debugging solutions exist to assist the application developer in finding and fixing bugs, both at small and large scale.

**Table 1.** Debugger compatibility matrix showing the level of support of different debuggers for the most popular GPU programming models [Status Feb. 2020]

| Tool | CUDA | OpenACC | OMPD | OpenCL |
|---|---|---|---|---|
| CUDA-MEMCHECK | yes | partly (CUDA kernels) | no | no |
| CUDA-GDB | yes | partly (CUDA kernels) | no | no |
| TotalView | yes | yes | prototype | no |
| DDT | yes | yes | no | no |

Debuggers, especially those for large-scale HPC systems are very complex and sophisticated pieces of software and virtually no open source solution exist here. The main debugging solutions for GPU programming right now are the NVIDIA provided CUDA-MEMCHECK and CUDA-GDB and the commercially available TotalView and DDT. Table 1 shows the supported GPU programming models of each of these debuggers. There is very good support for CUDA and OpenACC (where the NVIDIA tools support the debugging of the generated CUDA kernels), but nearly no support for the other programming models. TotalView showed a prototype with support for OpenMP offloading using an experimental OMPD-enabled OpenMP runtime. There exist a couple of debugges for OpenCL, but none proved usable for complex HPC applications[3].

### 2.1. NVIDIA Debugging Solutions

NVIDIA realized the importance of debugging for novel programming paradigms right from the beginning and shipped debugging tools right from the beginning with the CUDA toolkit [17]. These tools can be used standalone from the command-line, but are also integrated in the Nsight IDE [20], NVIDIAs development platform for CUDA and OpenACC applications. An example debugging session is shown in Fig. 1.

#### *2.1.1. CUDA-MEMCHECK*

CUDA-MEMCHECK is like valgrind for GPUs, a very powerful memory tracker and analysis tool. Hundreds of thousands of threads running concurrently on each GPU can be monitored.

[3]Some promising OpenCL debuggers are usable only on Microsoft Windows, which is not the intended platform for HPC applications

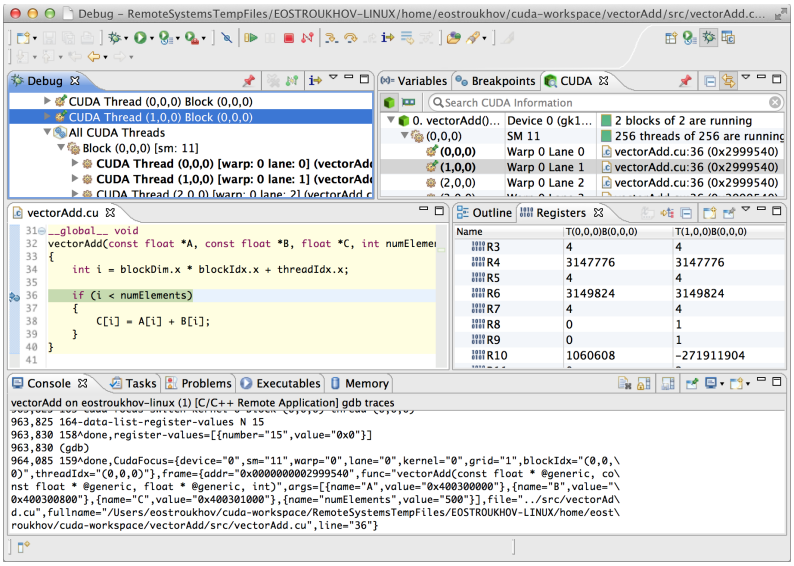



**Figure 1.** Debugging a CUDA application from within the Nsight IDE

It reports detailed information about global, local, and shared memory access errors (e.g. index out-of-bounds or misaligned memory accesses) and runtime executions errors (e.g. stack overflows and illegal instructions). Potential race conditions can also be detected with CUDA-MEMCHECK. In case of an error, CUDA-MEMCHECK displays stack back-traces on host and device.

### *2.1.2. CUDA-GDB*

CUDA-GDB is, as the name indicates, an extension to gdb, *the* Unix debugger. Simultaneous debugging on the CPU and multiple GPUs is possible. The user can set conditional breakpoints or break automatically on every kernel launch. It is possible to examine variables, read/write memory and registers and inspect the GPU state when the application is suspended. Memory access violations can be analyzed by running CUDA-MEMCHECK in an integrated mode to detect the precise causes.

## 2.2. TotalView

TotalView[4] is a symbolic debugger specifically designed for HPC applications written in C/C++, Fortran or Python. Noteworthy are the analysis capabilities for heavily templated C++ codes with complex data types. Advanced Memory Debugging allows to keep track of all memory accesses and memory allocations and deallocations to find memory leaks and corrupted memory. Another feature that sets TotalView apart from the competition is reverse debugging, i.e. the program execution is recorded and the user can step back from the point where the error occurred. This is especially helpful in fixing non-deterministic bugs. TotalView features full control over processes and threads with the ability to stop and debug an individual thread or groups of threads or processes. Debugging of CUDA [18] and OpenACC applications is supported with the possibility to debug multiple GPUs on a single node or multiple nodes across a cluster. Here it is possible to seamlessly set breakpoints in host and device code.

[4] `https://totalview.io/products/totalview`

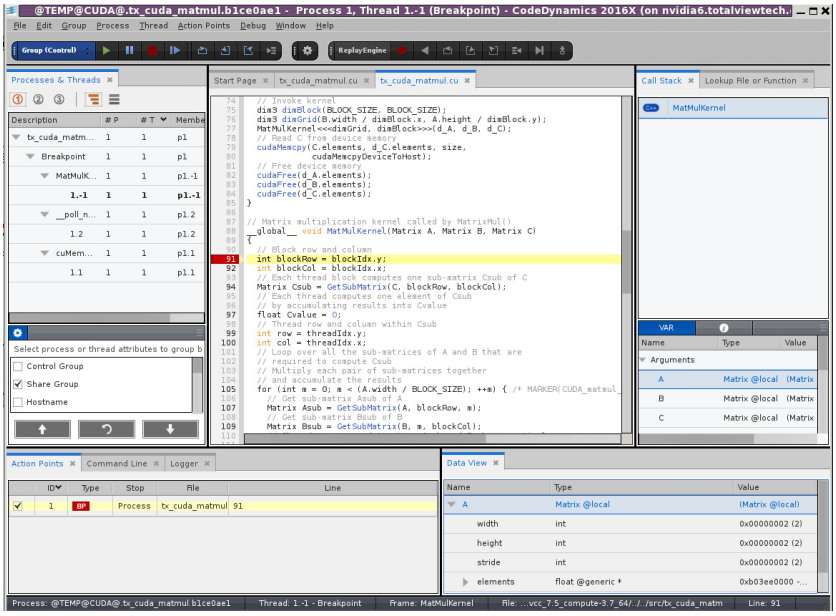


**Figure 2.** Debugging a matrix-multiplication kernel with TotalView

Figure 2 shows a screenshot of a CUDA debugging session using the new TotalView GUI, which greatly improves the usability.

## 2.3. Arm DDT

DDT[5] is another commercial debugger with a modern interface and very similar features to TotalView. It supports all major HPC programming languages with a special focus on complex C++ applications. Multi-process and multi-thread support is a matter of course. DDT also features advanced memory debugging and visualizations of huge data sets. Like TotalView, DDT supports debugging of CUDA and OpenACC applications with a fine-grained thread control, as shown in Fig. 3. DDT is available standalone or together with the Arm profiling tools in the Arm Forge suite[6].

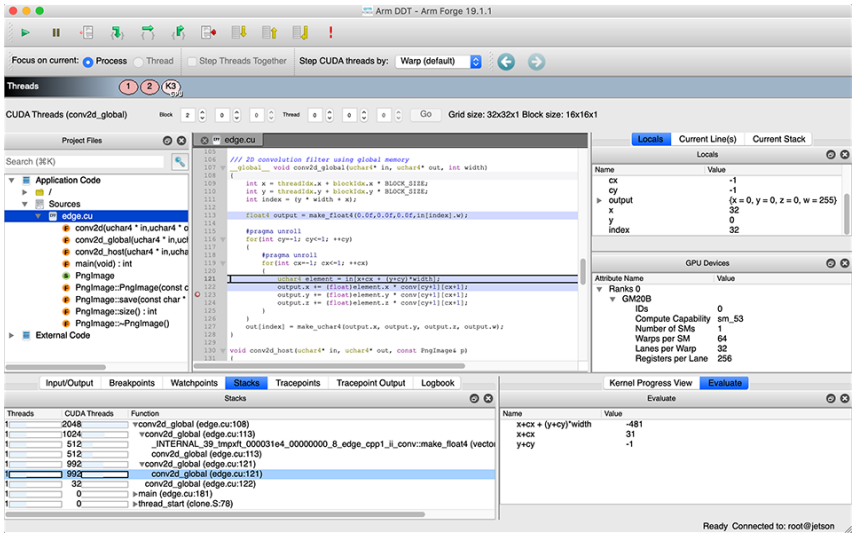


**Figure 3.** Arm DDT

[5] https://www.arm.com/products/development-tools/server-and-hpc/forge/ddt
[6] https://www.arm.com/products/development-tools/server-and-hpc/forge

# 3. Performance Analysis Tools

Performance analysis tools are an integral component in the HPC software stack for decades and many application developers were exposed to profilers to a certain degree. There are many tools for all kinds of analyzes, some are vendor provided and thus tied to a specific platform, some are commercial and several open source. The latter are usually developed at universities or national research laboratories with larger supercomputers. The tools community, which has a long history of collaboration, started adding GPU support relatively early [26], though the programming models and amount of features supported varies significantly between tools.

Though we commonly refer to performance analysis tools as profilers, we distinguish between trace-based tools, which store all events with timestamps and profile-based tools, which only store statistical information like the number of calls to a specific routine and the total time spend in that routine. Several tools can generate both profiles and traces and are thus universally applicable.

**Table 2.** Performance tool compatibility matrix showing the support for GPU programming models of several popular performance analysis tools [Status Feb. 2020]

| Tool | CUDA | OpenACC | OMPT | OpenCL |
|---|---|---|---|---|
| NVIDIA Tools | yes | yes | no | no |
| ARM Tools | yes | no | no | no |
| Score-P | yes | yes | prototype (no offload) | yes |
| TAU | yes | yes | prototype (no offload) | yes |
| HPCToolkit | yes | no | yes (experimental runtime) | no |
| Extrae/Paraver | yes | no | no | yes |

Tool support for the various GPU programming models varies significantly. The tool compatibility matrix for some of the most popular and wide-spread performance analysis tools is shown in Tab. 2. CUDA is supported by all the tools we consider. This is partly because CUDA was the first programming model for GPUs, but also because NVIDIA provides a very powerful and easy to use tools interface with CUPTI. Half of the tools support OpenACC or OpenCL, so there are options for all application developers. Several tools are working on supporting OpenMP offload to GPUs, but there is currently no public OpenMP runtime that implements OMPT for target directives. However, both Score-P and TAU already support OMPT on the host-side. HPCToolkit showed a prototype with OpenMP offload support using an internal experimental OpenMP runtime that implements OMPT for target directives.

## 3.1. NVIDIA Tools

NVIDIA realized early on that good tools (and a good documentation) are a necessity for a new platform to gain traction. So NVIDA began shipping their own profiler nvvp, the NVIDIA Visual Profiler, shortly after the release of CUDA. It is an integral feature of the CUDA tool-kit since then, so it is available on all CUDA-enabled platforms, without the need for a third-party tool. After several years, nvvp began to show scalability (and maintenance) issues and will be deprecated in a future CUDA release. Luckily, two new tools, Nsight Compute and Nsight System, are ready to fill that gap.

### *3.1.1. NVIDIA Visual Profiler*

For many years, nvvp [5] was the de-facto standard profiler for CUDA applications. It presents a unified CPU and GPU timeline including CUDA API calls, memory transfers and kernel launches. For a more detailed analysis of CPU activities, users can annotate the source code using the NVIDIA Tools Extension (NVTX) [24]. It supports all the advanced features of recent CUDA versions like Unified Memory, with CPU and GPU page faults and data migrations shown in the timeline. Upon selection of a specific kernel, nvvp shows a detailed low-level kernel analysis with performance metrics collected directly from GPU hardware counters and software instrumentation. Nvvp can compare results across multiple sessions to verify improvements from tuning actions. Another unique feature is an Automated or Guided Application Analysis with graphical visualizations to help identifying optimization opportunities. The Guided Analysis provides a step-by-step analysis and optimization guidance. The Visual Profiler is available as

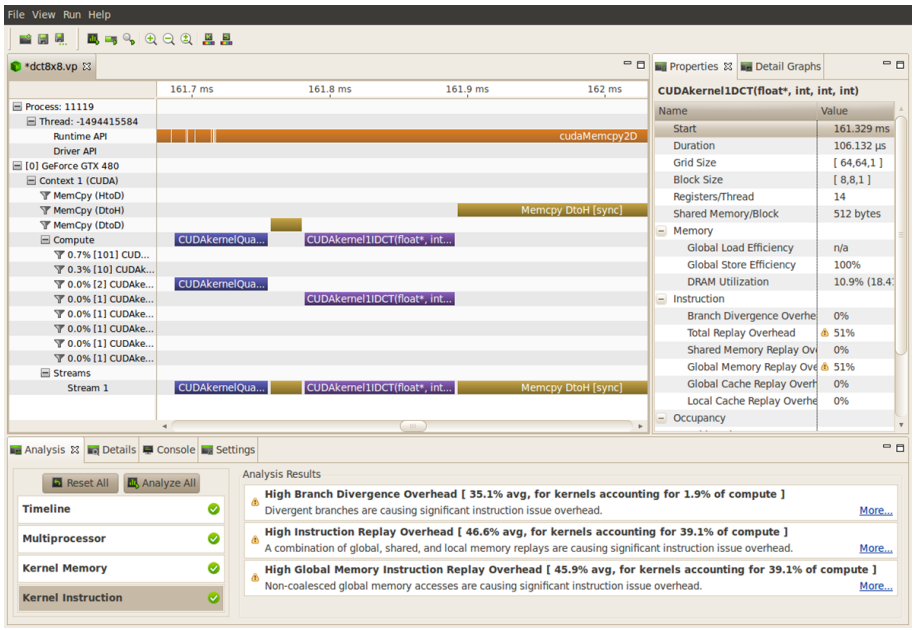



**Figure 4.** The NVIDIA Visual Profiler showing the timeline of the application execution, a detailed analysis of the selected kernel, and the results of the guided analysis

both a standalone application, as shown in Fig. 4 and, like NVIDIAs debugging solutions, as part of the Nsight IDE.

### *3.1.2. Nsight Compute*

NVIDIA Nsight Compute[7] is an interactive kernel profiler for CUDA applications. It provides similar features to nvvp's low-level kernel analysis, i.e. detailed performance metrics and the guided performance analysis. Nsight Compute provides a customizable and data-driven user interface (as shown in Fig. 5) Further, it has a command-line mode for manual and automated profiling and can be extended with analysis scripts for post-processing results. Additionally, its baseline feature allows users to compare results directly within the tool, very much like in the Visual Profiler.

[7] `https://developer.nvidia.com/nsight-compute-2019_5`

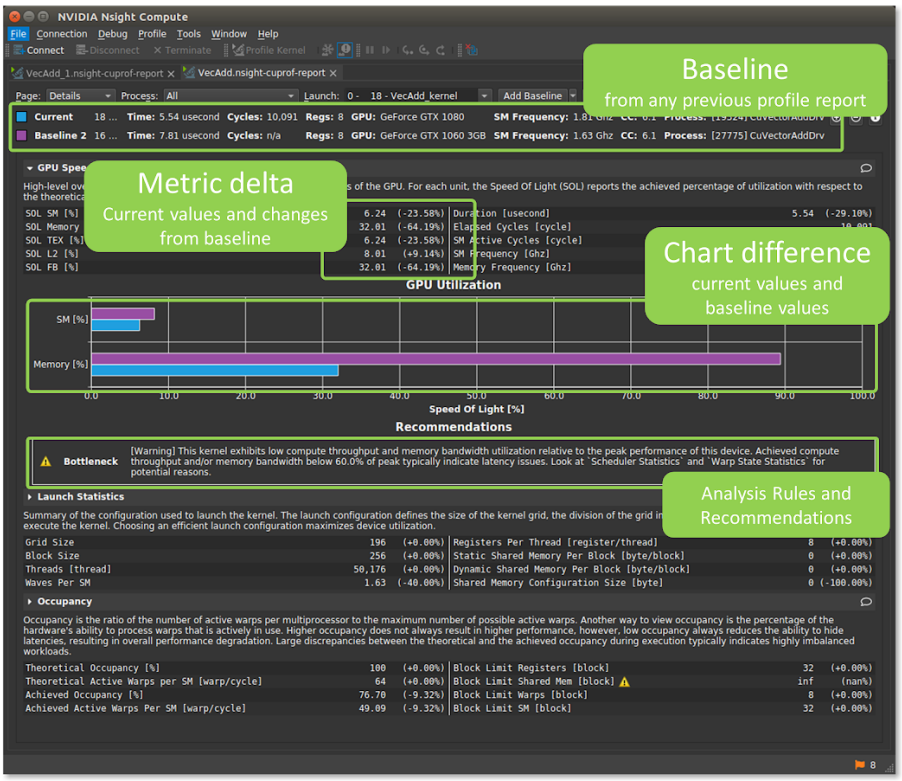



**Figure 5.** Nsight compute showing its detailed kernel analysis with the baseline comparison

### *3.1.3. Nsight Systems*

NVIDIA Nsight Systems[8] is a system-wide timeline-based performance analysis tool. It is designed to visualize the complete application execution help to identify the largest opportunities to optimize, and tune to scale efficiently across any quantity or size of CPUs and GPUs. Users will be able to identify issues, such as GPU starvation, unnecessary GPU synchronization, and insufficient overlap with CPU computation. An example timeline is shown in Fig. 6. It is possible to zoom in to any level of detail. Kernels showing unexpected behavior can be analyzed in detail with Nsight Compute, launched directly from the Nsight Systems GUI. NVTX is supported to get a more detailed picture of the CPU utilization. Currently Nsight System is focused on a single process, with more advanced support for MPI and OpenMP planned for a future release.

## 3.2. ARM Tools

Arm, since the acquisition of Allinea in 2016, provides several commercial cross-platform performance analysis tools, that can be obtained standalone or together with DDT in the Arm Forge[9] suite.

### *3.2.1. Performance Reports*

Arm Performance Reports is a gateway to the world of performance analysis. It is a very low-overhead tool working on unmodified optimized binaries that generates a one-page report characterizing the application performance at a rather high level. Performance Reports analyzes

[8] `https://developer.nvidia.com/nsight-systems`
[9] `https://www.arm.com/products/development-tools/server-and-hpc/forge`

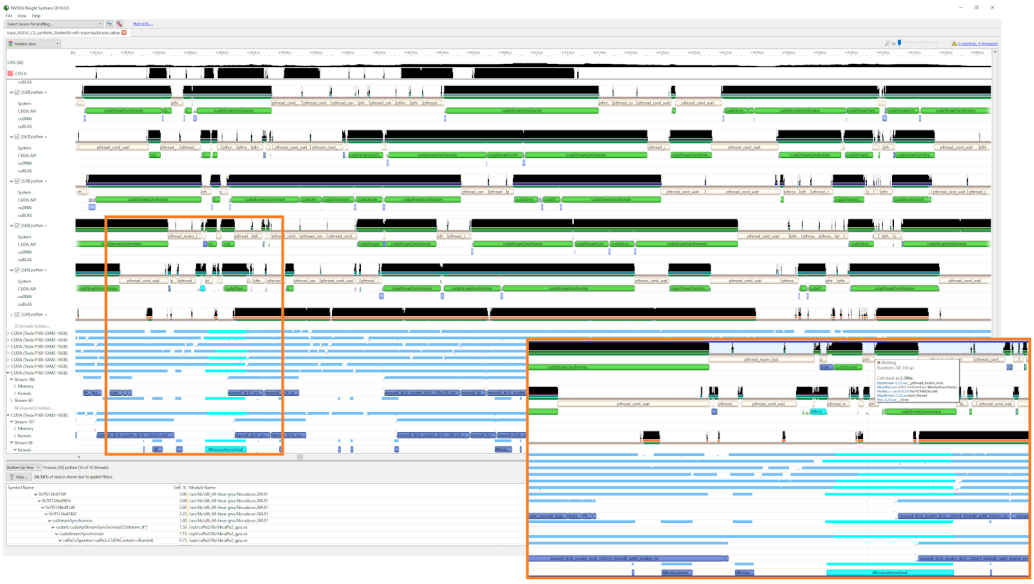


**Figure 6.** Nsight Systems showing the CPU timeline at the bottom and the activity of various CUDA stream on top

CPU utilization, MPI communication behavior, I/O and memory usage as well as accelerator usage. For all these categories it presents three to four sub-metrics to give more detailed information, e.g. the ratio of scalar and vector operations. For issues found, Performance Reports gives hints on how to proceed with more sophisticated analysis tools. An example of the accelerator breakdown of Performance Reports is shown in Fig. 7. This only gives a very brief overview of the GPU utilization, but in this case indicates that a thorough analysis with more advanced tools might be beneficial.

### *3.2.2. MAP*

Arm MAP [21] is a fully featured cross-platform source level performance analysis tool. It supports low-overhead sampling-based profiling of parallel multi-threaded C/C++, Fortran and Python codes. MAP providing in-depth analysis and bottleneck pinpointing to the source line as well as an analysis of communication and workload imbalance issues for MPI and multi-process codes. For accelerators, MAP offers a detailed kernel analysis with data obtained via CUPTI. This includes a line-level breakdown of warp stalls. Possible reasons for warp stalls include execution and memory dependencies or barriers. Knowing the reason for warp stalls can help the developer tuning the code accordingly. However, MAP currently supports only kernels generated

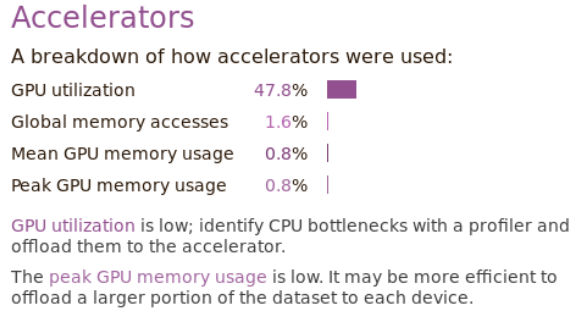



**Figure 7.** Performance Reports accelerator breakdown

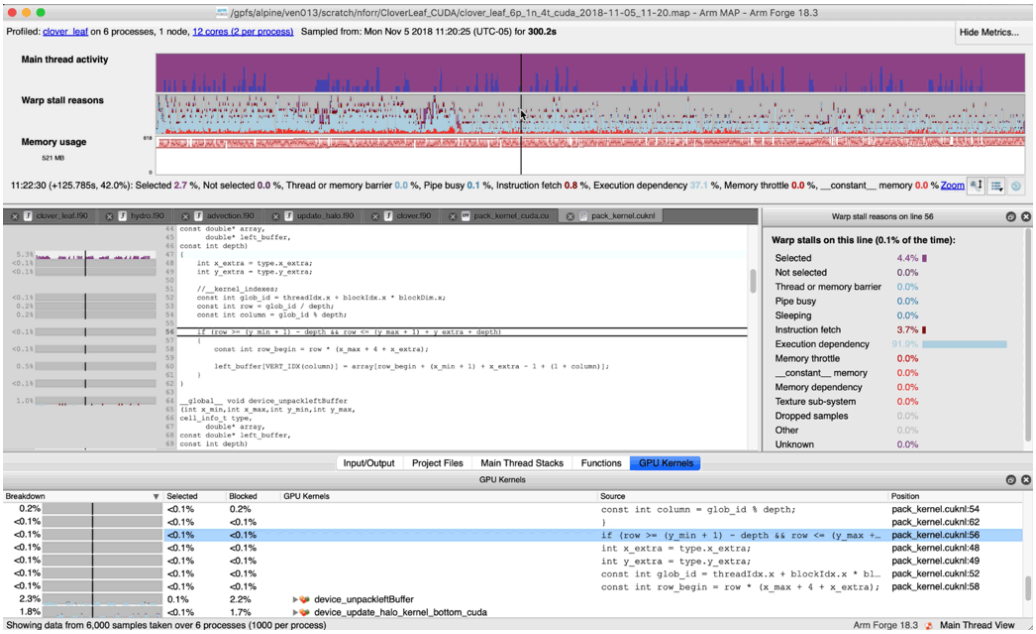



**Figure 8.** Arm MAP screenshot of a CUDA application analysis. It features the detailed warp stall analysis next to the source code

by CUDA-C++, not those generated by OpenACC, CUDA-Fortran or OpenMP offload. Figure 8 shows an example of MAP analyzing a CUDA application.

### 3.3. The Score-P Ecosystem

Score-P [23] is a community instrumentation and measurement infrastructure developed by a consortium of performance tool groups. It is the next-generation measurement system of several tools, including Vampir [22], Scalasca [16], TAU [42] and Periscope [3]. Common data formats for profiling (CUBE4) and tracing (OTF2 [14]) enable tools interoperability. Figure 9 gives an overview of the Score-P ecosystem. On the bottom are the various supported programming paradigms, which are implemented as independent adapters interacting with the measurement

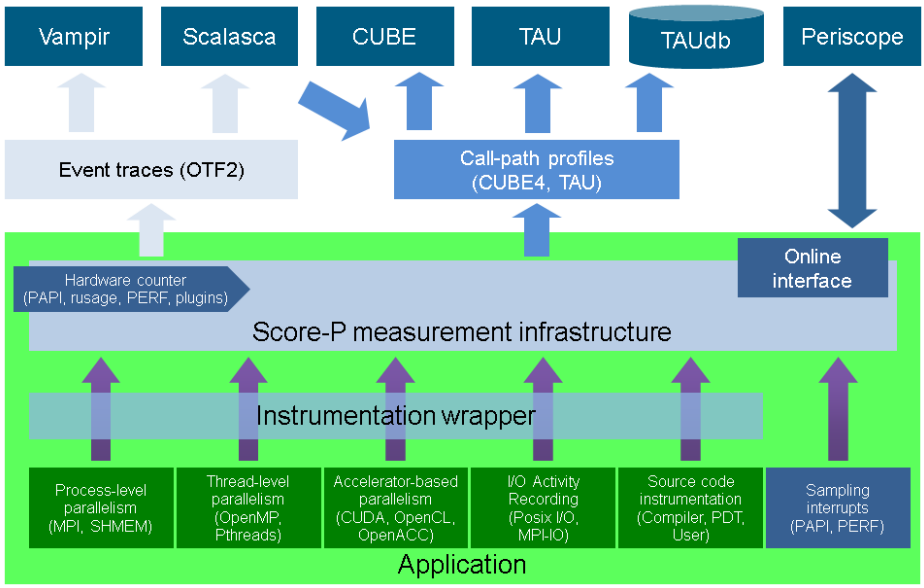
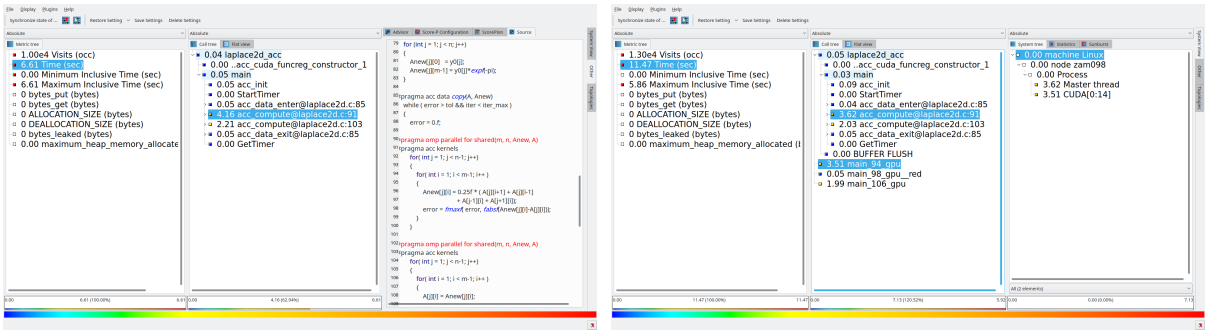



**Figure 9.** The Score-P ecosystem

system core. That eases adding support for new paradigms. The measurement data can be enriched with hardware counter information from PAPI [31], perf, or rusage. Score-P supports all major GPU programming models with CUDA [26], OpenACC [8], and OpenCL [9]. OMPT support for host-side measurement was recently added [15] and there is ongoing work to support OpenMP target directives [7]. Score-P also features a sampling mode for low-overhead measurements. It supports both profiling and tracing for all adapters. Profiles are generated in the CUBE4 format, that can be analyzed by TAU or Cube [39].

Cube is the performance report explorer for Score-P profiles as well as for the Scalasca trace analysis. The CUBE data model consists of a three-dimensional performance space with the dimensions (i) performance metric, (ii) call-path, and (iii) system location. Each dimension is represented in the GUI as a tree and shown in one of three coupled tree browsers, i.e. upon selection of one tree item the other trees are updated. Non-leaf nodes in each tree can be collapsed or expanded to achieve the desired level of granularity. Figure 10 shows a profile of a simple OpenACC application Cube GUI. On the left (Fig. 10a), the results of a pure OpenACC measurement are shown. Due to restrictions of the OpenACC tools interface, only the host-side calls are visible. However, if Score-Ps CUDA support is enabled as well, also the kernels generated by OpenACC get recorded (Fig. 10b).



(a) OpenACC only (b) OpenACC + CUDA

**Figure 10.** Screenshots of Cube showing the Score-P measurement of a simple OpenACC application, (a) with only OpenACC enabled, showing only the host side, and (b) with OpenACC and CUDA enabled, which additionally shows the kernels and device activity

OTF2 traces generated by Score-P can be analyzed automatically with Scalasca, which determines patterns indicating performance bottlenecks, and manually with Vampir. Unfortunately, Scalasca currently does not support the analysis of traces containing GPU locations, but can be used to analyze the communication of multi-node heterogenous programs if the corresponding adapter for the GPU programming model is disabled, i.e. only host-side events are recorded. In contrast to traditional profile viewers, which only present aggregated values of performance metrics, Vampir allows the investigation of the whole application flow. The main view is the Master Timeline which shows the program activity over time on all processes, threads, and accelerators. An example is shown in Fig. 11.

The Master Timeline is complemented by several other views, timelines, and tables, e.g. the Process Timeline to display the application call stack of a process over time or a Communication Matrix to analyze the communication between processes. Any counter metics, e.g. from PAPI or counter plugins, can be analyzed across processes and time with either a timeline or as a heatmap in the Performance Radar. It is possible to zoom into any level of detail, all views are updated automatically to show the information from the selected part of the trace.

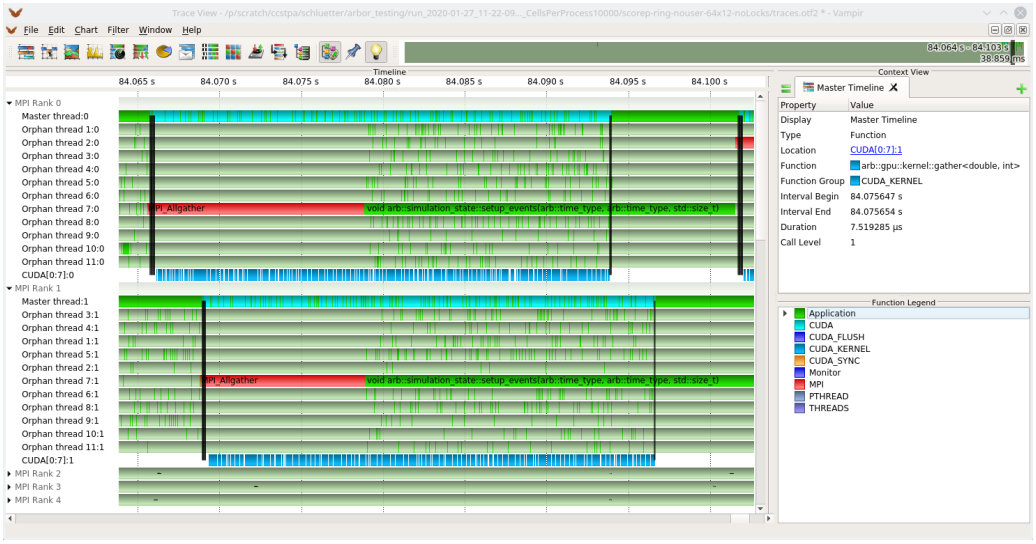



**Figure 11.** Vampir screenshot of an MPI + C++11 threads + CUDA application, showing a kernel launch on two processes in the Master Timeline. The line between the master process and the CUDA device indicates a data transfer, the thickness of the line represents the amount of data transferred

## 3.4. TAU

TAU [42] is a very portable tool-set for instrumentation, measurement and analysis of parallel multi-threaded applications. It features various profiling modes as well as tracing and various forms of code instrumentation as well as event-based sampling. All major HPC programming languages (C/C++, Fortran, Python) and programming models (MPI, OpenMP, Pthreads) are supported by TAU. TAU offers the widest support for accelerators, it allows measurement of CUDA [27], OpenACC, OpenCL, Kokkos [41], and also AMDs ROCm+HIP. For the analysis of 3-dimensional profile data, TAU includes ParaProf, which – like Cube – shows performance metric, call-path and location for an easy and quick investigation of bottlenecks. Figure 12 shows the visualization of a TAU trace file with Jumpshot[10].

## 3.5. Extrae/Paraver

Extrae[11] is a measurement system to generate Paraver trace files for post-mortem analysis. It supports C/C++, Fortran, and Python programs on all major HPC platforms, i.e. Intels x86, NVIDIA GPUs, Arm, and openPOWER. Extrae features several measurement techniques, which are configured through an XML file. The main source of information in Extrae is preloading shared libraries that substitutes symbols for many parallel runtimes, e.g. MPI, OpenMP and CUDA. Extrae also support dynamic instrumentation by modification of the application binary and parallel runtimes via Dyninst [4]. Further, Extrae supports sampling via signal timers and hardware performance counters. Since the Paraver trace format has no predefined semantics, adding support for new paradigms is relatively straightforward.

Paraver [37, 40] is a very flexible data browser working on the trace files generated by Extrae. Flexible means that there is no fixed set of metrics, the metrics can be programmed in the tool itself. Paraver offers a large selection of views, e.g. timelines, histograms, and tables, that can

[10] `https://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm`
[11] `https://tools.bsc.es/extrae`

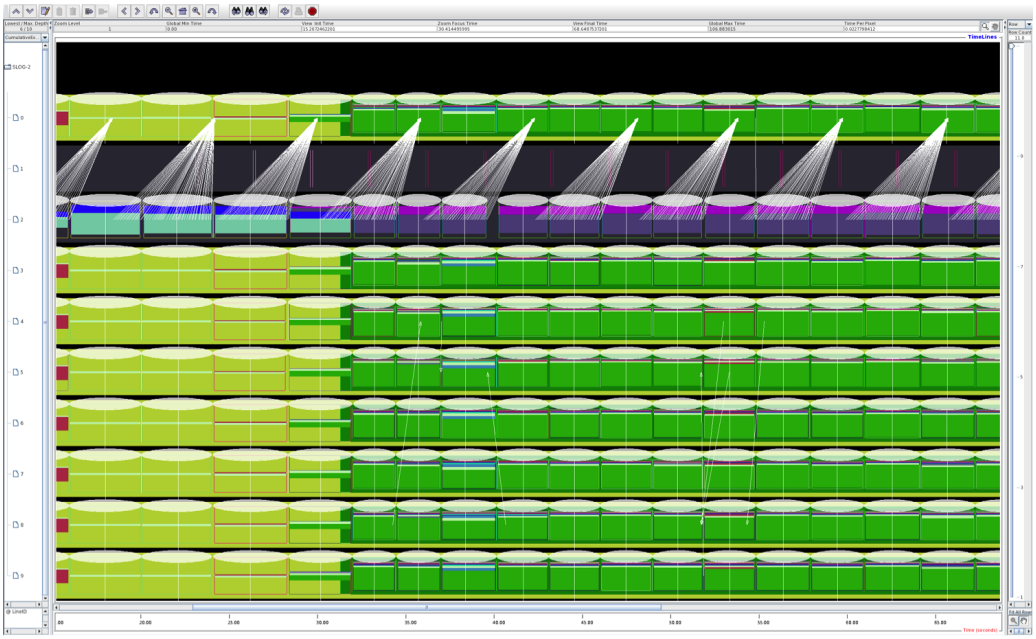


**Figure 12.** Jumpshot screenshot of a TAU trace measurement of a CUDA application

be combined to show virtually all the information that is present in the data. A view (or a set of views) can be saved as a Paraver configuration and recalculated with another trace file. CUDA streams are displayed in Paraver like any other data source, e.g. MPI processes or OpenMP threads.

## 3.6. HPCToolkit

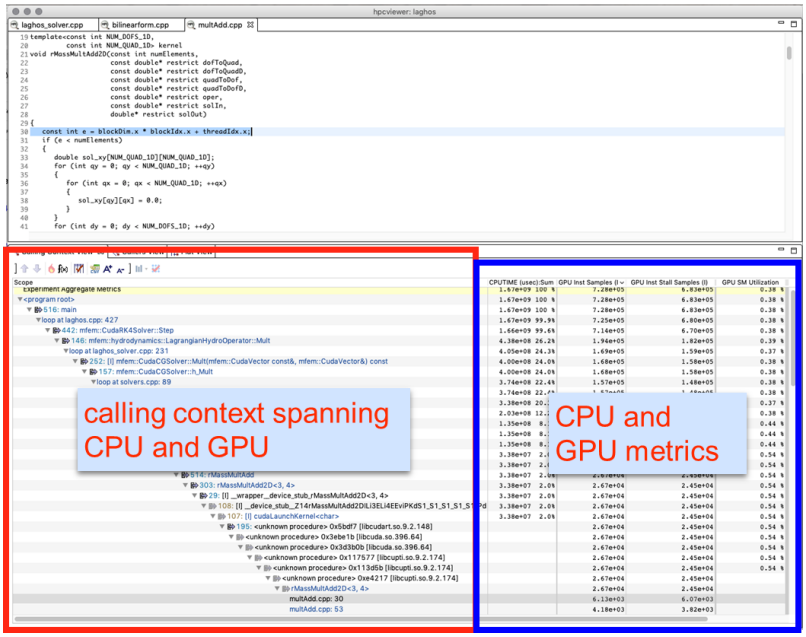



**Figure 13.** hpcviewer showing the profile of a CUDA application

HPCToolkit [1] is an integrated suite of tools for measurement and performance analysis of applications at all scales. Working on unmodified, fully optimized executables, it uses sampling to generate both call-path profiles and traces, independent of the language used. It supports multi-process (MPI) and multi-threaded (OpenMP, Pthreads) applications, but features no collection

or analysis of communication and I/O metrics. For GPU analysis, it supports CUDA [6] and, as a currently unique feature, OpenMP offload by shipping an experimental OpenMP runtime that implements OMPT for target constructs. It measures the execution time of each GPU kernel as well as explicit and implicit data movements. For CUDA codes it uses program counter sampling to pinpoint hotspots and to calculate the utilization of the GPU. A powerful analysis features of HPCToolkit is the blame shifting from symptoms to causes, so the user can quickly identify the real bottlenecks.

HPCToolkit consists of multiple programs that work together to generate a complete picture. **hpcrun** collects calling-context-sensitive performance data via sampling. A binary analysis to associate calling-context-sensitive measurements with source code structure is performed by **hpcstruct**. The performance data and the structure information are combined by **hpcprof** and finally visualized by **hpcviewer** for profiles and **hpctraceviewer** for traces.

## Conclusion

In this paper we showed that tools can support developers in programming heterogenous codes on current supercomputers, both in writing correct bug-free (debuggers) and efficient (performance analysis tools) applications.

There is one point in GPU programming where tools can't help – the decision which programming model to use. However, regardless of the choice, there is at least some tools support for each of the programming models. Due to the dominance of NVIDIA GPUs in today's datacenters, currently most developers choose CUDA or OpenACC, which are also the models with the best tools support. To use the tools as efficient as possible we recommend for that case to use the NVIDIA tools on a single node (with possibly multiple GPUs) when developing or porting the application to GPUs. When scaling up, i.e. inter-node data distribution and communication becomes an issue, we recommend the usage of more sophisticated tools like Score-P or TAU, which offer dedicated communication analysis. Errors occurring at scale can be debugged efficiently using TotalView or DDT.

Most supercomputing centers offer support to their users in porting and tuning applications to GPU architectures, sometimes in dedicated labs, like the JSC/NVIDIA Application Lab[12]. The tools community is also actively supporting users via mailing lists and trainings, e.g. the VI-HPS Tuning Workshops[13], which offer multi-day hands-on workshops covering a range of different tools.

So far the dominant player in GPU-enabled supercomputing is NVIDIA, but with the announced (Pre-)Exascale systems like Aurora[14], which will be based on Intels $\texttt{X}^e$ architecture, and Frontier[15] with purpose-build AMD GPUs, a wider variability of architectures becomes available. We will see systems using NVIDIA GPUs and Intel, AMD, IBM POWER and even Arm based CPUs, Intel Xeons with $\texttt{X}^e$ accelerators and completely AMD-based systems with EPYC CPUs and Radeon GPUs. Portability and maintainability of GPU applications will become more important, so developers might switch to more portable programming models like OpenMP or SYCL or even a higher-level abstraction model like Kokkos, to ensure performance portability. Tools will have to adapt to this increased variability and provide better support for more architectures and programming models.

[12] https://fz-juelich.de/ias/jsc/EN/Research/HPCTechnology/ExaScaleLabs/NVLAB/_node.html
[13] https://www.vi-hps.org/training/tws/tuning-workshop-series.html
[14] https://press3.mcs.anl.gov/aurora/
[15] https://www.olcf.ornl.gov/frontier/

### Acknowledgments

Parts of this work have received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824080.

The authors would like to thank the HPC performance tools community in general and the developers of the presented tools in particular as well as Andreas Herten (JSC) and Jiri Kraus (NVIDIA) from the joint JSC/NVIDIA Application Lab for many fruitful discussions, a long-standing collaboration, and the exchange of ideas.



## References

1. Adhianto, L., Banerjee, S., Fagan, M., et al.: HPCToolkit: Tools for performance analysis of optimized parallel programs. Concurrency and Computation: Practice and Experience 22(6), 685–701 (2010), DOI: 10.1002/cpe.1553

2. Beckingsale, D.A., Burmark, J., Hornung, R., et al.: RAJA: Portable Performance for Large-Scale Scientific Applications. In: 2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC, P3HPC, 22-22 Nov. 2019, Denver, CO, USA. pp. 71–81. IEEE (2019), DOI: 10.1109/P3HPC49587.2019.00012

3. Benedict, S., Petkov, V., Gerndt, M.: Periscope: An online-based distributed performance analysis tool. In: Tools for High Performance Computing 2009, Sept. 2009, Dresden, Germany. pp. 1–16. Springer (2010), DOI: 10.1007/978-3-642-11261-4_1

4. Bernat, A.R., Miller, B.P.: Anywhere, any-time binary instrumentation. In: Proceedings of the 10th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools, Szeged, Hungary. pp. 9–16. Association for Computing Machinery, New York, NY, USA (2011), DOI: 10.1145/2024569.2024572

5. Bradley, T.: GPU performance analysis and optimisation. In: NVIDIA Corporation (2012)

6. Chabbi, M., Murthy, K., Fagan, M., et al.: Effective sampling-driven performance tools for GPU-accelerated supercomputers. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 17-22 Nov. 2013, Denver, CO, USA. pp. 1–12. IEEE (2013), DOI: 10.1145/2503210.2503299

7. Cramer, T., Dietrich, R., Terboven, C., et al.: Performance analysis for target devices with the OpenMP tools interface. In: 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, 25-29 May 2015, Hyderabad, India. pp. 215–224. IEEE (2015), DOI: 10.1109/IPDPSW.2015.27

8. Dietrich, R., Juckeland, G., Wolfe, M.: OpenACC programs examined: a performance analysis approach. In: 2015 44th International Conference on Parallel Processing, 1-4 Sept. 2015, Beijing, China. pp. 310–319. IEEE (2015), DOI: 10.1109/ICPP.2015.40

9. Dietrich, R., Tschüter, R.: A generic infrastructure for OpenCL performance analysis. In: 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS, 24-26 Sept. 2015, Warsaw, Poland. vol. 1, pp. 334–341. IEEE (2015), DOI: 10.1109/IDAACS.2015.7340754

10. Dietrich, R., Tschüter, R., Cramer, T., et al.: Evaluation of Tool Interface Standards for Performance Analysis of OpenACC and OpenMP Programs. In: Tools for High Performance Computing 2015, Sept. 2015, Dresden, Germany. pp. 67–83. Springer, Cham (2016), DOI: 10.1007/978-3-319-39589-0_6

11. Edwards, H.C., Trott, C.R., Sunderland, D.: Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. Journal of Parallel and Distributed Computing 74(12), 3202–3216 (2014), DOI: 10.1016/j.jpdc.2014.07.003

12. Eichenberger, A., Mellor-Crummey, J., Schulz, M., et al.: OMPT and OMPD: OpenMP tools application programming interfaces for performance analysis and debugging. In: International Workshop on OpenMP, IWOMP 2013 (2013)

13. Eichenberger, A.E., Mellor-Crummey, J., Schulz, M., et al.: OMPT: An OpenMP Tools Application Programming Interface for Performance Analysis. In: OpenMP in the Era of Low Power Devices and Accelerators, IWOMP 2013, 16-18 Sept. 2013, Canberra, ACT, Australia. pp. 171–185. Springer, Berlin, Heidelberg (2013), DOI: 10.1007/978-3-642-40698-0_13

14. Eschweiler, D., Wagner, M., Geimer, M., et al.: Open Trace Format 2: The Next Generation of Scalable Trace Formats and Support Libraries. In: PARCO. vol. 22, pp. 481–490 (2011), DOI: 10.3233/978-1-61499-041-3-481

15. Feld, C., Convent, S., Hermanns, M.A., et al.: Score-P and OMPT: Navigating the Perils of Callback-Driven Parallel Runtime Introspection. In: International Workshop on OpenMP, IWOMP 2019, 11-13 Sept. 2019, Auckland, New Zealand. pp. 21–35. Springer (2019), DOI: 10.1007/978-3-030-28596-8_2

16. Geimer, M., Wolf, F., Wylie, B.J., et al.: The Scalasca performance toolset architecture. Concurrency and Computation: Practice and Experience 22(6), 702–719 (2010), DOI: 10.1002/cpe.1556

17. Gerfin, G., Venkataraman, V.: Debugging Experience with CUDA-GDB and CUDA-MEMCHECK. In: GPU Technology Conference, GTC (2012)

18. Gottbrath, C., Lüdtke, R.: Debugging CUDA Accelerated Parallel Applications with TotalView (2012)

19. Hammond, S.D., Trott, C.R., Ibanez, D., et al.: Profiling and Debugging Support for the Kokkos Programming Model. In: International Conference on High Performance Computing, 28 June 2018, Frankfurt/Main, Germany. pp. 743–754. Springer (2018), DOI: 10.1007/978-3-030-02465-9_53

20. Iyer, K., Kiel, J.: GPU Debugging and Profiling with NVIDIA Parallel Nsight. In: Game Development Tools, pp. 303–324. AK Peters/CRC Press (2016)

21. January, C., Byrd, J., Oró, X., et al.: Allinea MAP: Adding Energy and OpenMP Profiling Without Increasing Overhead. In: Tools for High Performance Computing 2014. pp. 25–35. Springer, Cham (2015), DOI: 10.1007/978-3-319-16012-2_2

22. Knüpfer, A., Brunst, H., Doleschal, J., et al.: The vampir performance analysis tool-set. In: Tools for High Performance Computing, July 2008, Stuttgart, Germany. pp. 139–155. Springer (2008), DOI: 10.1007/978-3-540-68564-7_9

23. Knüpfer, A., Rössel, C., an Mey, D., et al.: Score-P – A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir. In: Proc. of the 5th Int'l Workshop on Parallel Tools for High Performance Computing, Sept. 2011, Dresden, Germany. pp. 79–91. Springer (2012)

24. Kraus, J.: CUDA Pro Tip: Generate Custom Application Profile Timelines with NVTX. https://devblogs.nvidia.com/cuda-pro-tip-generate-custom-application-profile-timelines-nvtx/ (2013)

25. Lawrence Livermore National Laboratory: Sierra. https://computing.llnl.gov/computers/sierra (2020)

26. Malony, A.D., Biersdorff, S., Shende, S., et al.: Parallel performance measurement of heterogeneous parallel systems with gpus. In: Proceedings of the International Conference on Parallel Processing, ICPP 2011, 13-16 Sept. 2011, Taipei, Taiwan. pp. 176–185. IEEE (2011), DOI: 10.1109/ICPP.2011.71

27. Mayanglambam, S., Malony, A.D., Sottile, M.J.: Performance measurement of applications with GPU acceleration using CUDA. Advances in Parallel Computing 19, 341–348 (2010), DOI: 10.3233/978-1-60750-530-3-341

28. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard Version 3.1 (2015), https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

29. Messina, P.: The exascale computing project. Computing in Science & Engineering 19(3), 63–67 (2017), DOI: 10.1109/MCSE.2017.57

30. Mohr, B.: Scalable parallel performance measurement and analysis tools – state-of-the-art and future challenges. Supercomputing Frontiers and Innovations 1(2) (2014), DOI: 10.14529/jsfi140207

31. Mucci, P.J., Browne, S., Deane, C., et al.: PAPI: A portable interface to hardware performance counters. In: Proceedings of the department of defense HPCMP users group conference. vol. 710, pp. 7–10 (1999)

32. Nickolls, J., Buck, I., Garland, M., et al.: Scalable parallel programming with CUDA. Queue 6(2), 40–53 (2008), DOI: 10.1145/1365490.1365500

33. Oak Ridge National Laboratory: Summit. https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/ (2020)

34. OpenACC-Standard.org: The OpenACC Application Programming Interface 2.6 (2017), https://www.openacc.org/sites/default/files/inline-files/OpenACC.2.6.final.pdf

35. OpenMP Architecture Review Board: OpenMP Application Programming Interface Version 4.0 (2013), https://www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf

36. OpenMP Architecture Review Board: OpenMP Application Programming Interface Version 5.0 (2018), https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf

37. Pillet, V., Labarta, J., Cortes, T., et al.: Paraver: A tool to visualize and analyze parallel code. In: Proceedings of WoTUG-18: transputer and occam developments. vol. 44, pp. 17–31. CiteSeer (1995)

38. Reyes, R.: Codeplay contribution to DPC++ brings SYCL support for NVIDIA GPUs. https://www.codeplay.com/portal/02-03-20-codeplay-contribution-to-dpcpp-brings-sycl-support-for-nvidia-gpus (2020)

39. Saviankou, P., Knobloch, M., Visser, A., et al.: Cube v4: From Performance Report Explorer to Performance Analysis Tool. Procedia Computer Science 51, 1343–1352 (2015), DOI: 10.1016/j.procs.2015.05.320

40. Servat, H., Llort, G., Giménez, J., et al.: Detailed performance analysis using coarse grain sampling. In: European Conference on Parallel Processing, 25-28 Aug. 2009, Delft, The Netherlands. pp. 185–198. Springer, Berlin, Heidelberg (2009), DOI: 10.1007/978-3-642-14122-5_23

41. Shende, S., Chaimov, N., Malony, A., et al.: Multi-Level Performance Instrumentation for Kokkos Applications using TAU. In: 2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools, ProTools, 17 Nov. 2019, Denver, CO, USA. pp. 48–54. IEEE (2019), DOI: 10.1109/ProTools49597.2019.00012

42. Shende, S.S., Malony, A.D.: The TAU parallel performance system. The International Journal of High Performance Computing Applications 20(2), 287–311 (2006), DOI: 10.1177/1094342006064482

43. Wienke, S., Springer, P., Terboven, C., et al.: OpenACC – first experiences with real-world applications. In: European Conference on Parallel Processing, Euro-Par 2012, 27-31 Aug. 2012, Rhodes Island, Greece. pp. 859–870. Springer (2012), DOI: 10.1007/978-3-642-32820-6_85

DOI: 10.14529/jsfi200106

# Building a Vision for Reproducibility in the Cyberinfrastructure Ecosystem: Leveraging Community Efforts

*Dylan Chapp*[1], *Victoria Stodden*[2], *Michela Taufer*[1]



The scientific computing community has long taken a leadership role in understanding and assessing the relationship of reproducibility to cyberinfrastructure, ensuring that computational results – such as those from simulations – are "reproducible", that is, the same results are obtained when one re-uses the same input data, methods, software and analysis conditions. Starting almost a decade ago, the community has regularly published and advocated for advances in this area. In this article we trace this thinking and relate it to current national efforts, including the 2019 National Academies of Science, Engineering, and Medicine report on "Reproducibility and Replication in Science".

To this end, this work considers high performance computing workflows that emphasize workflows combining traditional simulations (e.g. Molecular Dynamics simulations) with *in situ* analytics. We leverage an analysis of such workflows to (a) contextualize the 2019 National Academies of Science, Engineering, and Medicine report's recommendations in the HPC setting and (b) envision a path forward in the tradition of community driven approaches to reproducibility and the acceleration of science and discovery. The work also articulates avenues for future research at the intersection of transparency, reproducibility, and computational infrastructure that supports scientific discovery.

*Keywords: reproducibility, replicability, transparency, high-performance computing, molecular dynamics, in situ analytics.*

## Introduction

In recent years, issues of reproducibility and replicability have come to the fore in venues as diverse as scholarly publications, numerous panels and presentations at conferences and other gatherings, and publications in the scholarly literature. These discussions and topics have engaged researchers in a diverse set of disciplinary areas such as scientific computing, the life sciences, statistics, geophysics, psychology, and more. A frequent thread in these discussions is the shortcomings in the clarity, completeness, and specificity of computational and data analysis methods in research dissemination. At the same time, journal editors and scientific societies have considered approaches to making available the code and data relied on published articles. In addition, national and international research funders have and are adopting requirements to promote transparency in research artifacts, such as data and code, that result from funded research.

Some of this activity can be rooted in examples that have been raised in the research community: the journal Nature recently reported that experiments at CERN had not shown neutrinos to be faster than light as originally reported [8]; data can be lost or unavailable and analysis algorithms in proprietary codes [46]; and a recent workshop at the ACM/IEEE Supercomputing (SC) conference discussed how parallelized simulation codes can in some cases produce unexpected nondeterminism in scientific findings [28]. Attention has recently been drawn to principles for advancing reproducibility in the computational context [41–43]. As a heuristic for

[1]University of Tennessee, Knoxville, Knoxville, TN, United States
[2]National Center for Supercomputing Applications, University of Illinois at Urbana Champaign, Champaign, IL, United States

understanding the salience of reproducibility issues, a Google Scholar search for "reproducibility" and "replicability" yields over 5,000 hits in 2019 alone, compared to 2009 with fewer than 800.

In this article, we trace the context and history of discussions and efforts regarding reproducibility in the high performance computing (HPC) context and list key efforts to improving our understanding of the costs and benefits of advancing reproducibility across the cyberinfrastructure ecosystem. We then relate these efforts to the National Academies of Science, Engineering, and Medicine 2019 report on "Reproducibility and Replication in Science" [33]. We use the framing of the report to discuss three of its recommendations regarding reproducibility and replication that are particularly actionable for research teams in HPC but whose level of abstractions may create interpretation ambiguity. To address the ambiguity, we discuss and interpret these recommendations in an exemplar case, the A4MD study [45], with the aim of enabling and advancing HPC communities in their current efforts to create reproducible, replicable, and transparent HPC ecosystems for smart cyberinfrastructures [10, 14, 20, 29, 39]. We then leverage to formalisms, PRIMAD [24] and Whole Tale's Tale [15], to apply the recommendations in the use case. We conclude with a call to extend these analysis to other use cases.

# 1. Reproducibility in HPC Driven Communities: Overview

## 1.1. Community Efforts

The notion of "really reproducible research" was introduced in 1992 [18, 19, 37] and coined in 1995 [9]. The term was intended to refer to the ability to computationally regenerate the results in a publication. Since these early days this idea has been developed and applied in many contexts [22], including policy development for journals [34] and research funding, as well as best practice and guidance development for institutions, repositories, and researchers. Many challenges to these ideas have been raised [6, 16, 32]. Most recently two of the authors participated in the development of seven guidance points for the community when stepping toward computational reproducibility [42]. Several conferences including the SC conference, the PPoPP conference, and the CGO conference, are taking steps toward to enable the integration of transparency in their paper artifacts and engaging students in the effort to promote reproducibility, replicability, and transparency. One of the authors has led the effort in the past five years to make sure that the papers accepted to the SC conference have enough information to trust their results. At SC19, for the first time, all accepted papers included an appendix with a detailed artifact description of environments and methodologies that were used for achieving the key results in the papers. In pursuing the success of the reproducibility initiative, the conference has engaged the next HPC generation through the Reproducibility Challenge in the Student Cluster Competition (SCC): a paper accepted to a past SC conference is used as source for the Reproducibility Challenge of the next SC conference. SCC is an SC program that engages 16 teams (of 6 undergraduate students each) every year who are tasked to work with a vendor to build a HPC cluster from scratch and run a set of key HPC benchmarks on it during the conference. These benchmarks now include the replication of artifacts in the selected paper on the 16 different cluster architectures, creating a unique setting for practitioners to study the impact of different hardware platforms on the performance of a single common application.

## 1.2. Identifying Sources of Irreproducibility

First efforts to address sources of irreproducibility tackled numerical reproducibility [11, 44]. Numerical reproducibility focuses on the relationship between system software, hardware, and the ability to return bit-wise identical output [21, 27]. In the scientific domains there is generally less concern with obtaining bit-wise identical results from one study or experiment to another, however changes in the underlying computational system can give rise to uncertainties that can affect the scientific interpretation of computational results [40]. There are several possible computational sources of irreproducibility including:

- Hardware: Many fundamental operations of a computer are inherently non-deterministic. I/O devices report interrupts at unpredictable times, affecting scheduling of processes and progress of I/O, each visible to the application at the system call layer.
- Concurrency: Current systems provide high degrees of concurrency at all levels (e.g., applications use multiple processes, multiple threads, multiple cores, and/or rely on parallel accelerators like GPUs).
- Algorithmic Randomness: Many fundamental scientific algorithms rely upon random number generators: Monte Carlo sampling algorithms, random walks, and so on.
- Application Complexity: The overall application extends beyond the application code, and includes supporting libraries and services, configuration files, the operating system, and perhaps even the configuration of the network upon which it relies. It is typical to employ more than one application in the discovery process, adding to the complexity as interactions and dependencies between applications may not be well understood. Each of the environment elements may be configured and updated independently by different parties, e.g. end user(s), system and network administrators, and automatic processes.
- Provenance Capture: Assessing and verifying the significance of a data or computationally-enabled scientific finding typically requires understanding the statistical, modeling, and calibration steps taken, including the capture and reporting of negative findings and the steps used to create visualizations and figures that present results. In addition, many applications embed state information into their output to help with debugging and general provenance, however such information may not be sufficient to assess whether results that differ bit-wise are scientifically equivalent.

Applications such as Coulomb n-body atomic system simulations, planetary orbit calculations, supernova simulations all require stringent bit-wise numerical reproducibility [4].

## 1.3. Formalisms and Abstractions

The community is taking a structured approach to reason about and assess reproducibility in the cyberinfrastructure context at large, beyond bit-wise reproducibility. We outline the use of two formalisms to allow the community to understand the impact of changes including costs and benefits: the PRIMAD model designed to understand changes when research is replicated [24], and the "Tale" description of reproducible published computational research [15].

**The PRIMAD Model**: PRIMAD is a general model intended to guide reproducibility. PRIMAD helps meet an acute need in the scientific community to ground reproducibility, yet it is inherently abstract due to its applicability across all scientific domains, leading to challenges in establishing a useful level of specificity. PRIMAD breaks reproducibility into six named components (Platform, Research objective, Implementation, Methods, Actors, and Data), each of

which represents an element of a computational experiment where reproducibility can be enforced by design, or conversely where a lack of such design can allow irreproducibility to seep in and potentially corrode the overall integrity of the experiment. As a first example of a PRIMAD applicability study, two of the authors have successfully evaluated the efficacy of PRIMAD as a tool for characterizing the reproducibility of more traditional applications such as real-world computational science workflows. Specifically, we examined computational workflows used to detect gravitational waves using data from the Laser Interferometer Gravitational-Wave Observatory (LIGO) [1] and the Virgo Observatory [2]. Our findings outlined how PRIMAD can be used as a general model to guide reproducibility from publications [12].

**The Whole Tale "Tale"**: The object defined as a "Tale" is a digital bundle of artifacts and descriptors for the dissemination and publication of computational scientific findings in the scholarly record [15]. The NSF-funded Whole Tale project is developing a computational environment designed to capture the entire computational pipeline associated with a scientific experiment and thereby enable computational reproducibility [7]. In other words, research published from the Whole Tale project is published in the Tale format, which allows researchers to create and package the code, data, and information about the workflow and computational environment necessary to support, review, and recreate the computational results reported in published research. As shown in Tab. 1, the Tale captures the artifacts and information needed to facilitate greater understanding, transparency, and executability of the Tale for review and reproducibility at the time of publication.

**Table 1.** A manifest of objects that comprise the Whole Tale "Tale" and whose descriptions are included as Tale metadata. Adapted from [15]

| **Metadata** | **Description** |
|---|---|
| Authors | List of Tale authors |
| Creators | Tale Creators (may differ from authors) |
| Title | Title of the Tale |
| Description | Description of the Tale |
| Categories | List of subject categories (keywords) |
| Illustration | Illustration for the Whole Tale browse page |
| Create Date | Date the Tale was created |
| Update Date | Date the Tale was last updated |
| License | License selected by the user |
| Environment | Computational environment information |
| Workspace | Code/scripts, workflow, narrative, documentation, data, results |
| External data | Data by reference to external source |
| Identifier | Persistent identifier for published Tale |

Without standardization, decisions about what constitutes "relevant information" are inevitably ad-hoc, and may not be uniform from publication to publication or across multiple workflows within a single publication. Thus, formalisms such as PRIMAD and the Tale offer an abstraction with which to build sustainable reproducibility in a uniform fashion across scientific domains.

## 2. The 2019 National Academies Report Recommendations and the HPC Ecosystem

The 2019 National Academies of Science, Engineering, and Medicine (NASEM) consensus report "Reproducibility and Replication in Science", of which one of us was a committee member, found its origin in the 2017 "American Innovation and Competitiveness Act". In this Act Congress made a provision that directed the National Science Foundation to assess "research and data reproducibility and replicability issues in interdisciplinary research" and make "recommendations for improving rigor and transparency in scientific research". This opportunity offered a chance to understand the problem and the current state of reform efforts, and to articulate ways the National Science Foundation and others might improve reproducibility and replicability in research. The NASEM report set forth definitions of the terms "reproducibility" and "replication" and offers a number of recommendations regarding reproducibility and replication [33]. We discuss each of those aspects in turn.

Key words used in reproducibility discussions may have different interpretations or meanings in different disciplines and even in different discussions. For the purposes of the NASEM report the committee established the following definitions (reproduced without modification). We follow this convention in the current writing as it is consistent with previous efforts [38].

> **Reproducibility** is obtaining consistent results using the same input data, computational steps, methods, and code, and conditions of analysis. This definition is synonymous with "computational reproducibility", and the terms are used interchangeably in this report.
>
> **Replicability** is obtaining consistent results across studies aimed at answering the same scientific question, each of which has obtained its own data. Two studies may be considered to have replicated if they obtain consistent results given the level of uncertainty inherent in the system under study.
>
> **Generalizability**, another term frequently used in science, refers to the extent that results of a study apply in other contexts or populations that differ from the original one. A single scientific study may include elements or any combination of these concepts.

Reproducibility involves the original data and code; replicability involves carrying out new studies or experiments to ascertain consistency with previous answers to the same research question. In addition, these definitions suggest that when underlying digital artifacts are made accessible, the results should ideally be reproducible. However, a study conducted according to best practices and utilizing correct analysis may of course fail to replicate due to inherent uncertainties of other factors.

Among the recommendations regarding reproducibility and replication provided by the report, some are more actionable than others for research teams in the HPC setting. Accordingly, we prioritize discussion of recommendations that both describe or refer to potential changes to computational scientists' day-to-day engineering practices that could encourage or enable reproducibility and replicability of their research, and advocate for enhancements of computational scientists' software infrastructure, where success will positively impact computational science ranging from workstation-scale prototyping to studies run on leadership-class HPC systems.

Our empirical and experiential evaluation identified three NASEM report recommendations that are particularly suitable to be tailored for HPC workflows and HPC practitioners. These are reproduced from the report without modification:

> **RECOMMENDATION 4-1:** To help ensure the reproducibility of computational results, researchers should convey clear, specific, and complete information about any computational methods and data products that support their published results to enable other researchers to repeat the analysis, unless such information is restricted by non-public data policies. That information should include the data, study methods, and computational environment:
>
> - the input data used in the study either in extension (e.g., a text file or a binary) or in intension (e.g., a script to generate the data), as well as intermediate results and output data for steps that are non-deterministic and cannot be reproduced in principle;
> - a detailed description of the study methods (ideally in executable form) together with its computational steps and associated parameters; and
> - information about the computational environment where the study was originally executed, such as operating system, hardware architecture, and library dependencies (which are relationships described in and managed by a software dependency manager tool to mitigate problems that occur when installed software packages have dependencies on specific versions of other software packages).
>
> **RECOMMENDATION 5-1:** Researchers should, as applicable to the specific study, provide an accurate and appropriate characterization of relevant uncertainties when they report or publish their research. Researchers should thoughtfully communicate all recognized uncertainties and estimate or acknowledge other potential sources of uncertainty that bear on their results, including stochastic uncertainties and uncertainties in measurement, computation, knowledge, modeling, and methods of analysis.
>
> **RECOMMENDATION 6-3:** Funding agencies and organizations should consider investing in research and development of open-source, usable tools and infrastructure that support reproducibility for a broad range of studies across different domains in a seamless fashion. Concurrently, investments would be helpful in outreach to inform and train researchers on best practices and how to use these tools.

We refer the reader to the report for more details on these and the other recommendations [33]. In the next section we interpret these recommendations in the context of a specific HPC workflow, Analytics for Molecular Dynamics (A4MD).

## 3. Assessing the Impact of the 2019 NASEM Report Recommendations on an HPC Workflow: The A4MD Use Case

In this section we discuss the applicability of the three targeted NASEM recommendations to a real HPC use case, the Analytics for Molecular Dynamics (A4MD) workflow [45]. This

use case focuses on molecular dynamics simulations that are augmented with *in situ* analytics components, thus allowing us to study a research workflow that integrates data factors into a traditionally compute intensive only project. We utilize the specific use case approach to concretize and interpret the NASEM recommendations.

### 3.1. The A4MD Use Case

Molecular Dynamics (MD) simulations studying the time evolution of a molecular system at atomic resolution. The fields of chemistry, material sciences, molecular biology, and drug design widely utilize MD simulations. The system sizes and time-scales accessible to MD simulations have been steadily increasing. Next-generation HPC systems will have dramatically larger compute performance than do current systems. This increase in computing capability directly translates into the ability to execute an increasing number of longer simulations and thus to expand the range of biomolecular phenomena that can be studied by MD simulation.

### 3.2. The NASEM Recommendations in the Context of the A4MD Workflow

The A4MD workflow presents unique challenges for compliance with the best practices outlined in NASEM recommendations, particularly in terms of capturing and disseminating the A4MD computational environment, and all of its relevant data products. Recommendation 4-1 explicitly states that the "operating system, hardware architecture, and library dependencies" of a computational experiment should be captured and shared. Since A4MD consists of three distinct computational components (the molecular dynamics simulation itself, the data staging server, and the *in situ* analytic packages), each of which may execute on separate hardware resources, the difficulty of fulfilling this requirement is magnified. We summarize in Fig. 1 a set of metadata for each of the three A4MD components that can conceivably fulfill the requirements of environment sharing specified in Recommendation 4-1. These metadata can, in principle, be captured in an automated fashion as part of the job scripts that comprise the workflow.

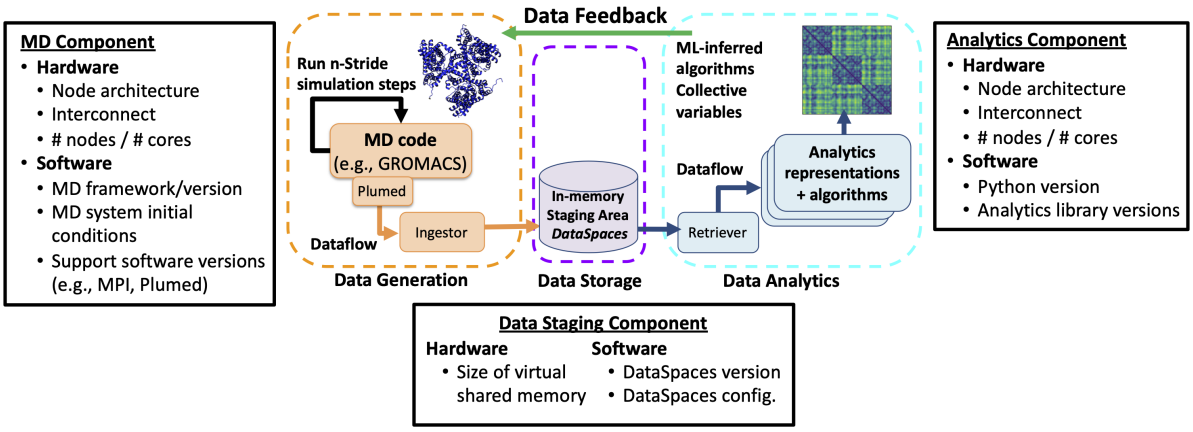



**Figure 1.** Capturing A4MD's computational environment

Beyond capture of the computational environment, Recommendation 4-1 also calls for "input data used in the study either in extension (e.g., a text file or a binary) or in intension (e.g., a script to generate the data), as well as intermediate results and output data for steps that are nondeterministic and cannot be reproduced in principle". While capture of intermediate data products can potentially bolster efforts to achieve reproducibility, doing so necessarily comes at

the cost of scalability, especially in the HPC setting. Efforts to achieve scalable record and replay of HPC applications indicate that capturing fine-grained data about the intermediate state of parallel executions remains an active and challenging area of research [13]. Hence, in our view the feasibility of Recommendation 4-1's guidelines regarding capture of intermediate data must be managed on a case-by-case basis. In Fig. 2 we sketch out a possible set of data products that could be recorded per execution of the A4MD workflow, ranging from the highly feasible, e.g., the input files to the MD simulation, to the highly challenging (e.g., the evolving state of the data staging server).

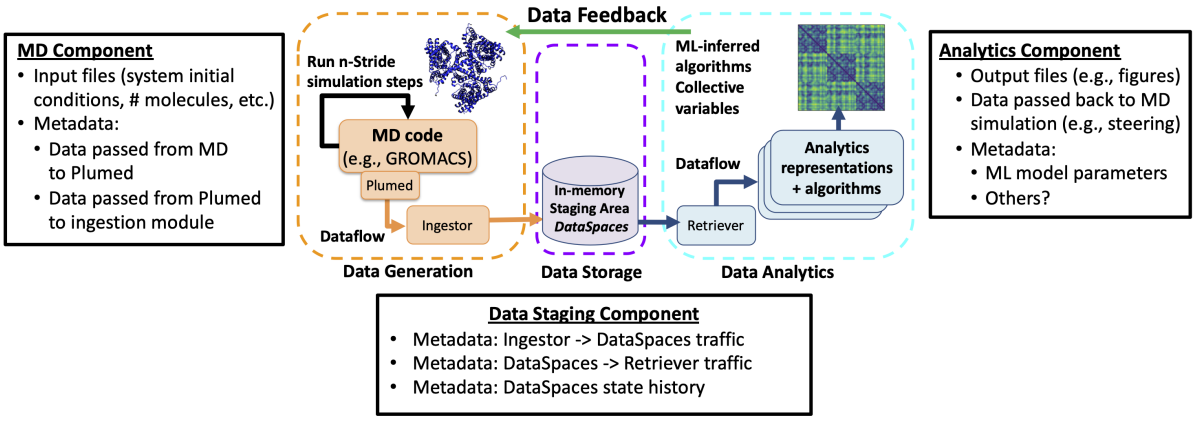



**Figure 2.** Capturing A4MD's input, output, and intermediary data

NASEM Recommendation 5-1 stresses the importance of uncertainty quantification in computational experiments. In an effort to comply with this recommendation, an empirical evaluation of the A4MD workflow was conducted in [45] (specifically, in Section III.C) to quantify the effect of imbalances the rate at which the MD simulation produces data and the rate at which the *in situ* analytics module consumes that data. That evaluation has provided insights for the revision of Rec. 5-1 in the next section.

Finally, in an effort to make the A4MD workflow more accessible to other researchers, the experiments described in [45] are packaged as a Jupyter notebook. However, the A4MD workflow typically involves submission of multiple interdependent jobs to a batch-scheduler; an activity not native supported by the Jupyter notebook environment. Instead, the authors of A4MD had to leverage various third-party workarounds (e.g., the signac workflow manager [3]) to encapsulate these implementation details away from the user experience of A4MD. These efforts highlight the need of the NASEM Recommendation 6-3 revision presented below, namely the need for investment in open-source, reproducibility-oriented software infrastructure.

The NASEM recommendations have broad applicability across computational settings, and we interpret the recommendations focused on in this work in the specific setting of leadership-class high performance computing (HPC) platforms. These recommendations have the highest potential for positive impact on trust and traceability of scientific findings and face unique challenges due to the characteristics of HPC platforms and HPC software.

## 4. The NASEM Recommendations in the HPC Setting

We leverage lessons learned from the use case in Section 3.1 to address the challenges that research teams will face when implementing a subset of the NASEM recommendations for their HPC-enabled scientific workflows. Additionally, and where appropriate, we propose HPC-

tailored refinements of the recommendations with the intent of facilitating their widespread adoption. We will refer to the NASEM recommendations by their numerical identifier as given in the report (e.g., Rec. 4-1).

### 4.1. Recommendation 4-1: Sharing Methods, Data, and Environment

Scientific workflows, especially those deployed on HPC resources, rarely consist of a single computational element. More often, workflows consist of multiple executables that generate or consume data sets, and multiple scripts that serve functions such as pre- and post-processing of data sets, visualization, and gluing together other computational elements. Furthermore, these computational elements are not executed directly by researchers on a static resource (e.g., a single workstation), but instead are scheduled onto available resources–either by a traditional batch-scheduler (e.g., SLURM) or a more sophisticated workflow manager (e.g., Pegasus [20]).

All of these factors motivate the need to refine Rec. 4-1s request to "convey clear, specific, and complete information about any computational methods". It may be tempting to interpret this as being fulfilled by English text descriptions in publications (i.e., the text of a "Methods" or "Evaluation" section), we contend that this is insufficient–especially in the HPC setting. Instead, the "computational methods" should be described as a directed graph of executable elements. Each vertex in this graph would represent a single executable, script, or scheduled job, each associated with metadata such as commit hashes, library versions, and input parameters. In contrast with a possibly-ambiguous or incomplete English language description of "computational methods", this representation affords researchers with the ability to distinguish between structurally similar, but nevertheless distinct computational methods which is critical to investigating the reproducibility of scientific findings.

Even if an unambiguous and formal, yet shareable and ergonomic representation of "computational methods" gains traction in the scientific community, a further challenge remains: namely updating the peer-review process to appropriately evaluate study methods expressed in this form. This challenge is starting to take shape in the present day as computational notebooks (e.g., Jupyter) become more and more popular as vehicles for sharing scientific findings.

### 4.2. Recommendation 5-1: Broadening Notions of Uncertainty Quantification

Despite the relative ubiquity of uncertainty quantification (UQ) in the HPC setting, it is usually targeted towards probing the effect of uncertainty of inputs to simulations, rather than uncertainty inherent in the HPC platform itself. However, we contend that in the HPC setting, three factors contribute to the need to treat HPC platforms as dynamic environments in need of UQ just as much as the inputs of sensitive simulations: (1) use of multiple parallel runtimes; (2) multi-tenancy on HPC systems; and (3) opacity of code generation.

For *multiple parallel runtimes*, to cope with evolving HPC system architectures, the use of multiple parallel runtimes (e.g., MPI + OpenMP) in a single codebase has become increasingly common in scientific computing. The effect of mixing these runtimes on application-level non-determinism has been identified as a major challenge in the push to exascale [26], and the scarcity of tools for mitigating non-determinism in these types of codebases has been documented [13].

For *multi-tenancy*, beyond the challenge of reproducing the internal state of non-deterministic applications from run to run, a greater challenge lies in reproducing the state

of the system on which those applications ran, at the time that they ran. The majority of computational science on HPC systems is performed on systems in which the investigator is not the sole tenant. Thus, contention for resources such as network bandwidth between compute nodes or IO bandwidth between the system and a parallel file system can conceivably contribute to reproducibility challenges.

Finally, for *opacity of code generation*, the increasing complexity of scientific codebases coupled with the rising popularity of high-level user-friendly interfaces to them (e.g., computational notebooks) contributes to an increased risk of computational scientists being fundamentally unfamiliar with the code they execute. Incremental increases in complexity may be unavoidable for scientific codebases, we encourage computational scientists to familiarize themselves with modern tools that can increase their awareness of code-generation effects that may impede reproducibility. For example, the FLiT tool [35, 36] allows users to assess the effects of various combinations of compiler options on their codes numerical properties, while tools like Spack [25] ease the burden of maintaining and organizing multiple versions of complex scientific software built against multiple toolchains.

## 4.3. Recommendation 6-3: Investment in Open Source Tools to Facilitate Reproducible Research

The NASEM recommendations advocate for increased investment from funding agencies in open-source tools tailored towards reproducible research. While tools and infrastructure have emerged recently (e.g., Whole Tale [7], Popper [30, 31], ReproZip [17], Repo2Docker [23]) these tools may require that their users adhere to specific organizational patterns for their projects, or simply require additional steps in setup that researchers may find cumbersome. We contend that the fundamental tools by which researchers develop experiments ought to have reproducibility-oriented features baked in as first-class citizens [43]. In particular, we contend that computational notebooks are an attractive candidate for such an overhaul due to their increasing ubiquity; their design that co-locates data, code, and exposition; and their as-of-yet untapped capacity to capture metadata about computational experiments in support of reproducibility.

Were a funding body to invest in greenfield development of a reproducibility-oriented computation notebook environment, we suggest that the following features be prioritized: (1) automated experiment metadata collection; (2) interoperability with existing version control systems; and (3) interoperability with HPC system software.

**Automated Metadata Collection:** Computational notebooks present a user-friendly environment where typically, a scripting languages readevalprint loop (REPL), data visualization capabilities, and free form textual exposition, are able to be colocated. We suggest that in addition to these advantages, computational notebooks are uniquely positioned to capture metadata about computational experiments (e.g., versions of third-party libraries, identifiers for datasets, configuration details for how figures were generated) that are essential for achieving reproducibility. The HPC community has stressed the importance of collecting this metadata and provided tools for doing so, such as the SC Reproducibility Initiatives Artifact Descriptor Toolkit [5]. However the inherent drawback of tools like this is that they constitute an extra, post-hoc step for researchers–separate from their day-to-day experimental workflow. Were this functionality to be integrated directly into a reproducibility-oriented computational notebook, this metadata would be captured as a matter of course–and consequently more likely to be available to the greater scientific community.

**Interoperability with version control:** Currently, computational notebooks present challenges for version control. The notebook is typically stored in a hierarchical format such that small changes from the perspective of the user interface may induce relatively large changes in the underlying document (e.g., swapping cell orders in a Jupyter notebook). While this does not exclude notebooks from versioning via, e.g., Git, per se, it does render the commit history for a notebook significantly less transparent and informative than the commit history for a regular source file. A future reproducibility-oriented redesign of the computational notebook should prioritize improving integration with version control.

**Interoperability with HPC system software:** Despite the ease-of-use computational notebooks have enjoyed for prototyping experiments, there remain pain-points when it comes to porting these prototypes to run on large-scale HPC resources [14]. We argue that it is imperative that computational notebooks evolve to integrate seamlessly with batch schedulers so that researchers may more easily and reproducibly scale up their prototypes.

# 5. Applying Formalisms to Assess the NASEM Recommendations in the HPC Ecosystem

The formalisms discussed in Section 1.3 suggest guidance to understanding how to generalize findings from use cases and thereby indicate potential avenues to build sustainable reproducibility efforts in a uniform fashion across scientific domains. We identify how specific components of the two proposed formalisms (i.e., PRIMAD and the Whole Tale) can be identified or defined in order to support applicability of the three targeted recommendations (i.e., 4-1, 5-1, and 6-3) across workflows in a specific domain and for desired levels of reproducibility. The first step is to apply the formalism, the second is to update the interpretation of the three recommendations targeted in this work.

## 5.1. Applying the PRIMAD Formalism

We begin by presenting the elements of the PRIMAD formalism in Tab. 2. The second column of the Table applies these elements to the A4MD use case discussed in this work.

A clear division between implementation and methods in the PRIMAD model is fundamental for Rec. 4-1 but such a separation is subjective. Minor adjustments to an algorithm generally fall into implementation, yet it is hard to determine when changes are substantial enough to call it a new algorithm and thus a change in methods. In other cases, the effects of the human actors on reproducibility may be difficult to document. Even within the same research group and under consistent leadership, research objectives, and computational environments, changes in team members and shifts in member responsibility can introduce unacknowledged sources of variability. Appropriately documenting the knowledge and experience that is applied to the elements of a workflow is a challenge and it is important to understand when and how scientific results rely on specific human actions for example.

## 5.2. Applying the Tale Formalism

The Tale description is given in Tab. 3 with associated detail for the A4MD as best as we are able since the implementation of the A4MD use case in Whole Tale is currently underway. However, some aspects of the Tale format could be refined to fit the A4MD workflow better. In

**Table 2.** Applying the PRIMAD Formalism in the A4MD Use Case

| PRIMAD Element | Application to A4MD Use Case |
|---|---|
| Platform | NERSCs Cori Cray XC40 System |
| Research | Molecular Dynamics (MD) simulations executed on a state-of-the-art supercomputer that characterize the impact of in situ and in transit analytics on overall MD workflow performance, and the capability for capturing rapid, rare events in the simulated molecular system. |
| Implementation | Two workflow configurations are run that represent in situ and in transit analytics on Haswell nodes of NERSCs Cori. Each Haswell node has two16-core Intel Xeon processors, 128GB memory, and are connected by a Cray Aries interconnect. |
| Methods | In the first type of workflow, because the analysis is not able to consume the frame in a timely manner, the MD either simulation waits in I/O to write to the in-memory staging area of DataSpaces (idle simulation time) or discards any frame that cannot be ingested into the staging area. In the second type of workflow the MD simulation generates a new frame with large strides and the analytics are waiting in I/O and the associated resources are idle. Trends for the time spent waiting in I/O for the simulation and idle time for the analytics are measured and observed. |
| Actors | Researchers at multiple institutions. |
| Data | MD-generated data created as output from the workflow. |

particular, since the A4MD workflow consists of multiple applications (i.e., the MD simulation, the data staging server, and the *in situ* analytics modules) that potentially execute on different hardware platforms, the monolithic "environment" component of the Tale ought to be decomposed into a collection of environments. Links between various sub-components of the Tale's "workspace" and individual environments within that collection could then make explicit the correct way to set up and execute an equivalent A4MD workflow in a future replication study.

### 5.3. Application of the Formalisms to Our Analysis of Three NASEM Recommendations

In our analysis of the NASEM recommendations in the HPC setting, we observe significant overlap between aspects of the recommendations and the common components of reproducibility formalisms such as PRIMAD and Whole Tale. In particular, there is a clear parallel between Recommendation 4-1's emphasis on sharing study methods, computational environment, and data, and "methods", "platform", and "data" components of PRIMAD, or the "environment", "workspace", and "external data" components of Whole Tale. Our approach of leveraging formalisms helps refine and define what this might mean in particularly research settings. In Section 4.1 we discuss the potential pitfalls of compliance with Recommendation 4-1 in the HPC setting, and suggest possible refinements. Reproducibility formalisms are a natural vehicle by which those refinements can be made explicit and actionable for research teams.

**Table 3.** Applying the Whole Tale "Tale" Formalism in the A4MD Use Case

| Tale Element | Application to A4MD Use Case |
|---|---|
| Authors | Thomas, S., Wyatt, M., Do, T.M.A., Pottier, L., da Silva, R.F., Weinstein, H.,Cuendet, M.A., Estrada, T., Deelman, E., Taufer, M. |
| Creators | C. Willis |
| Title | Characterizing in-situ and in transit analytics of molecular dynamics simulations for next generation supercomputers. |
| Description | This tale implements the computational pipeline associated with the publication cited in [45]. |
| Categories | Scientific workflows, data analytics, performance, workload modeling, remote direct memory access. |
| Illustration | Figure 1 "Capturing A4MD's computational environment". |
| Create Date | February 2020 |
| Update Date | February 2020 |
| License | [License selected by the user] |
| Environment | [Computational environment information] |
| Workspace | Code/scripts, workflow, results |
| External data | None. |
| Identifier | As yet unpublished Tale |

Elsewhere, specifically with respect to Recommendation 5-1, there is less overlap with existing reproducibility formalisms. Neither PRIMAD nor Whole Tale explicitly guide researchers towards incorporating uncertainty quantification into their studies. We contend that failure to quantify and report potential uncertainties of the computational environment can have dramatic impacts on reproducibility, and thus warrants explicit incorporation into future reproducibility formalisms. As Whole Tale is an active and ongoing development effort, there is potential to align aspects of the Tale format with Recommendation 5-1.

Finally, while in our discussion in Section 4.3 we focus on potential improvements to computational notebooks, we also see in well-funded open-source software a natural avenue for reproducibility formalisms to become useful and ubiquitous. As software tools for computational scientists mature, integration of a reproducibility formalism and tools into the common software stacks can reduce the degree of effort required for research teams to conduct reproducible experiments and disseminate sufficient information for the broader community to build on their work.

## Conclusion

Even in the absence of a community-standardized formalism for reproducibility, individual research teams can nevertheless strive to comply with the NASEM recommendations, to the extent that the NASEM recommendations are sensibly interpreted for their specific use case. In this work, we presented one example of this with the A4MD workflow, and based on our example we articulated a set of refinements to Recommendations 4-1, 5-1, and 6-3 that renders them more suitable for computational science conducted in the HPC setting. We also showed an

approach to making the recommendation implementations explicit and actionable through the use of a reproducibility formalism.

The results presented in this work are intended to indicate areas for further investigations. We see two principal avenues to extend our work.

First, our analysis was augmented with one single HPC use case. Still, our use case allowed us to concretize and interpret the NASEM recommendations, as well as to indicate future directions while opening the door to the empirical analysis of the impact of the recommendations in a broader HPC settings and workflows. The extension of our empirical approach based on use cases to a larger and diverse suites of HPC workflows can allow scientists and practitioners to understand the impact, costs, and benefits of the NASEM recommendations on the reproducibility of more and more complex HPC ecosystems.

Second, the two considered formalisms were not initially designed to resolve questions tackled in this work such as "how do suggested adjustments to research workflows affect HPC ecosystems as a whole and improve their reproducibility"? Still, the clarity they can each bring is an important step. Ultimately reproducibility formalisms should be further refined to completely and automatically capture the appropriate elements of the HPC ecosystem that are most impacted by the implementation of increased computational reproducibility.

## Acknowledgments

This work was performed with the support of the National Science Foundation awards: OAC #1941443 EAGER: Reproducibility and Cyberinfrastructure for Computational and Data-Enabled Science; OAC #1841399 Collaborative: EAGER: Exploring and Advancing the State of the Art in Robust Science in Gravitational Wave Physics; OAC #1839010: EAGER: Preserve/Destroy Decisions for Simulation Data in Computational Physics and Beyond; and OAC #1541450: CC*DNI DIBBS: Merging Science and Cyberinfrastructure Pathways: The Whole Tale.

The authors thank Cuendet MA, and Deelman E, Estrada T, Ferreira Da Silva R, and Weinstein H, for their contributions with the A4MD workflow design and implementation.



## References

1. Aasi, J., Abbott, B.P., Abbott, R., et al.: The LIGO scientific collaboration. Classical and Quantum Gravity 32(7), 074001 (2015), DOI: 10.1088/0264-9381/32/7/074001

2. Acernese, F., Agathos, M., Agatsuma, K., et al.: Advanced Virgo: a second-generation interferometric gravitational wave detector. Class.Quant. Grav. 32, 2 32(2) (2015), DOI: 10.1088/0264-9381/32/2/024001

3. Adorf, C.S., Dodd, P.M., Ramasubramani, V., et al.: Simple data and workflow management with the signac framework. Computational Materials Science 146, 220–229 (2018), DOI: 10.1016/j.commatsci.2018.01.035

4. Bailey, D., Barrio, R., Borwein, J.: High-precision computation: Mathematical physics and dynamics. Applied Mathematics and Computation 218(20), 10106–10121 (2012), DOI: 10.1016/j.amc.2012.03.087

5. Barba, L.A.: SC reproducibility initiative author-kit. `https://github.com/SC-Tech-Program/Author-Kit` (2013)

6. Barba, L.A.: The hard road to reproducibility. Science 354(6308), 142–142 (2016)

7. Brinckman, A., Chard, K., Gaffney, N., et al.: Computing environments for reproducibility: Capturing the "Whole Tale". Future Generation Comp. Syst. 94, 854–867 (2019), DOI: 10.1016/j.future.2017.12.029

8. Brumfiel, G.: Neutrinos not faster than light. ICARUS experiment contradicts controversial claim. Nature (2012)

9. Buckheit, J.B., Donoho, D.L.: WaveLab and Reproducible Research, pp. 55–81. Springer, New York, NY (1995), DOI: 10.1007/978-1-4612-2544-7_5

10. Canon, R.S., Younge, A.: A case for portability and reproducibility of HPC containers. In: IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC, CANOPIE-HPC, 18 Nov. 2019, Denver, CO, USA. pp. 49–54. IEEE (2019), DOI: 10.1109/CANOPIE-HPC49598.2019.00012

11. Chapp, D., Johnston, T., Taufer, M.: On the need for reproducible numerical accuracy through intelligent runtime selection of reduction algorithms at the extreme scale. In: Proceedings of the 2015 IEEE International Conference on Cluster Computing. pp. 166–175 (2015), DOI: 10.1109/CLUSTER.2015.34

12. Chapp, D., Rorabaugh, D., Brown, D.A., et al.: Applicability study of the PRIMAD model to LIGO gravitational wave search workflows. In: Proceedings of the 2nd International Workshop on Practical Reproducible Evaluation of Computer Systems, P-RECS@HPDC 2019. pp. 1–6 (2019), DOI: 10.1145/3322790.3330591

13. Chapp, D., Sato, K., Ahn, D., et al.: Record-and-replay techniques for HPC systems: A survey. Supercomputing Frontiers and Innovations 5(1), 11–30 (2018), DOI: 10.14529/jsfi180102

14. Chard, K., Gaffney, N., Hatigan, M., et al.: Toward enabling reproducibility for data-intensive research using the Whole Tale platform. In: Proceedings of the International Conference on Parallel Computing, PARCO 2019. Advances in Parallel Computing, IOS Press (2019)

15. Chard, K., Gaffney, N., Jones, M.B., et al.: Implementing computational reproducibility in the Whole Tale environment. In: Proceedings of the 2nd International Workshop on Practical Reproducible Evaluation of Computer Systems, P-RECS '19. pp. 17–22. ACM, New York, NY, USA (2019), DOI: 10.1145/3322790.3330594

16. Chen, X., Dallmeier-Tiessen, S., Dasler, R. et al.: Open is not enough. Nature Physics 15, 113–119 (2019), DOI: 10.1038/s41567-018-0342-2

17. Chirigati, F., Shasha, D., Freire, J.: Reprozip: Using provenance to support computational reproducibility. In: Presented as part of the 5th USENIX Workshop on the Theory and Practice of Provenance. USENIX (2013)

18. Claebout, J.: Hypertext documents about reproducible research (1994), http://sepwww.stanford.edu/doku.php

19. Claerbout, J.F., Karrenbach, M.: Electronic documents give reproducible research a new meaning. In: SEG Technical Program Expanded Abstracts 1992, pp. 601–604. Society of Exploration Geophysicists (1992), DOI: 10.1190/1.1822162

20. Deelman, E., Vahi, K., Juve, G., et al.: Pegasus: a workflow management system for science automation. Future Generation Computer Systems 46, 17–35 (2015), DOI: 10.1016/j.future.2014.10.008

21. Demmel, J., Nguyen, H.D.: Fast reproducible floating-point summation. In: 2013 IEEE 21st Symposium on Computer Arithmetic, 7-10 April 2013, Austin, TX, USA. pp. 163–172. IEEE (2013), DOI: 10.1109/ARITH.2013.9

22. Donoho, D.L., Maleki, A., Rahman, I.U., Shahram, M., Stodden, V.: Reproducible research in computational harmonic analysis. Computing in Science Engineering 11(1), 8–18 (2009), DOI: 10.1109/MCSE.2009.15

23. Forde, J., Head, T., Holdgraf, C., et al.: Reproducible research environments with repo2docker. In: ICML 2018 Reproducible Machine Learning. ICML (2018)

24. Freire, J., Fuhr, N., Rauber, A.: Reproducibility of Data-Oriented Experiments in e-Science (Dagstuhl Seminar 16041). Dagstuhl Reports 6(1), 108–159 (2016), DOI: 10.4230/DagRep.6.1.108

25. Gamblin, T., LeGendre, M., Collette, M.R., et al.: The Spack package manager: bringing order to HPC software chaos. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC'15, 15-20 Nov. 2015, Austin, TX, USA. pp. 1–12. IEEE (2015), DOI: 10.1145/2807591.2807623

26. Gopalakrishnan, G., Hovland, P.D., Iancu, C., et al.: Report of the HPC correctness summit, Jan 25–26, 2017, Washington, DC. CoRR abs/1705.07478 (2017), https://arxiv.org/abs/1705.07478

27. He, Y., Ding, C.H.: Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications. The Journal of Supercomputing 18(3), 259–277 (2001), DOI: 10.1023/A:1008153532043

28. Honarmand, N., Torrellas, J.: Replay debugging: Leveraging record and replay for program debugging. SIGARCH Comput. Archit. News 42(3), 445–456 (2014), DOI: 10.1145/2678373.2665737

29. James, D., Wilkins-Diehr, N., Stodden, V., Colbry, D., Rosales, C., et al.: Standing together for reproducibility in large-scale computing: Report on reproducibility@xsede. CoRR abs/1412.5557 (2014), http://arxiv.org/abs/1412.5557

30. Jimenez, I., Arpaci-Dusseau, A., Arpaci-Dusseau, R., et al.: PopperCI: Automated reproducibility validation. In: 2017 IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, 1-4 May 2017, Atlanta, GA, USA. pp. 450–455. IEEE (2017), DOI: 10.1109/INFCOMW.2017.8116418

31. Jimenez, I., Sevilla, M., Watkins, N., et al.: The Popper convention: Making reproducible systems evaluation practical. In: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW, 29 May-2 June 2017, Lake Buena Vista, FL, USA. pp. 1561–1570. IEEE (2017), DOI: 10.1109/IPDPSW.2017.157

32. Mirowski, P.: The future(s) of open science. Social Studies of Science 48(2), 171–203 (2018), DOI: 10.1177/0306312718772086

33. National Academies of Sciences, Engineering, and Medicine: Reproducibility and Replicability in Science. The National Academies Press, Washington, DC (2019), DOI: 10.17226/25303

34. Peng, R.D.: Reproducible research and biostatistics. Biostatistics 10(3), 405–408 (2009), DOI: 10.1093/biostatistics/kxp014

35. Sato, K., Laguna, I., Lee, G.L., et al.: PRUNERS: Providing reproducibility for uncovering non-deterministic errors in runs on supercomputers. The International Journal of High Performance Computing Applications 33(5), 777–783 (2019), DOI: 10.1177/1094342019834621

36. Sawaya, G., Bentley, M., Briggs, I., et al.: FLiT: Cross-platform floating-point result-consistency tester and workload. In: 2017 IEEE international symposium on workload characterization, IISWC, 1-3 Oct. 2017, Seattle, WA, USA. pp. 229–238. IEEE (2017), DOI: 10.1109/IISWC.2017.8167780

37. Schwab, M., Karrenbach, N., Claerbout, J.: Making scientific computations reproducible. Computing in Science Engineering 2(6), 61–67 (2000), DOI: 10.1109/5992.881708

38. Stodden, V.: Resolving irreproducibility in empirical and computational research. IMS Bulletin (November 2013), http://bulletin.imstat.org/2013/11/resolving-irreproducibility-in-empirical-and-computational-research/

39. Stodden, V., Borwein, J., Bailey, D.H.: Setting the default to reproducible in computational science research. SIAM News 46(5), 4–6 (2013), https://sinews.siam.org/Details-Page/setting-the-default-to-reproducible-in-computational-science-research

40. Stodden, V., Krafczyk, M.: Assessing reproducibility: An astrophysical example of computational uncertainty in the HPC context. In: The 1st Workshop on Reproducible, Customizable and Portable Workflows for HPC, HPC18 (2018)

41. Stodden, V., Leisch, F., Peng, R.D.: Implementing Reproducible Research. The R Series, Chapman & Hall/CRC (2014)

42. Stodden, V., McNutt, M., Bailey, D.H., et al.: Enhancing reproducibility for computational methods. Science 354(6317), 1240–1241 (2016), DOI: 10.1126/science.aah6168

43. Stodden, V., Miguez, S.: Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. Journal of Open Research Software 2(1) (2014), DOI: 10.5334/jors.ay

44. Taufer, M., Anderson, D., Cicotti, P., et al.: Homogeneous redundancy: A technique to ensure integrity of molecular simulation results using public computing. In: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 4-8 April 2005, Denver, CO, USA. IEEE (2005), DOI: 10.1109/IPDPS.2005.247

45. Thomas, S., Wyatt, M., Do, T.M.A., et al.: Characterizing in situ and in transit analytics of molecular dynamics simulations for next generation supercomputers. In: Proceedings of the International Conference on eScience, eScience'19, 24-27 Sept. 2019, San Diego, CA, USA. pp. 188–198. IEEE (2019), DOI: 10.1109/eScience.2019.00027

46. Wild, S.: Irreproducible astronomy. Physics Today (2018), DOI: 10.1063/PT.6.1.20180404a