Supercomputing Frontiers and Innovations

2023, Vol. 10, No. 2

Scope

- Future generation supercomputer architectures
- Exascale computing
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Novel approaches to computing targeted to solve intractable problems
- Convergence of high performance computing, machine learning and big data technologies
- Distributed operating systems and virtualization for highly scalable computing
- Management, administration, and monitoring of supercomputer systems
- Mass storage systems, protocols, and allocation
- Power consumption minimization for supercomputing systems
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Scientific visualization in supercomputing environments
- Education in high performance computing and computational science

Contents

DOI: 10.14529/jsfi230201

November 2022 Top500 List Overview

Nikolay S. Abramov[1], *Sergei M. Abramov*[1]

The article is devoted to the analysis of the current state of the supercomputer industry and the prospects for its development. In terms of methodological approach and tools, the work is a continuation of a series of similar analytical reviews by the authors. The main source of information for analysis is the archive of editions (releases) of the world ranking of the five hundred most productive supercomputers in the world. The novelty of this work lies not only in updating the information, taking into account the latest editions of the Top500 list, but also in focusing on the following circumstance: the global supercomputer industry is undergoing a radical restructuring – transition from the "petascale era" to the "exascale era". Technological trends and features of solutions for the most productive systems in the world in recent years are given. The pace of development of supercomputer technologies, development trends are discussed: hybrid architectures, interconnect technologies and changes in the positions of supercomputer manufacturing companies. Based on the results of the analysis, reliable forecasts were made for the coming years about the general appearance of exascale systems.

Keywords: Top500, HPC, supercomputers, hybrid architectures, interconnect.

Introduction

In leading countries, supercomputer technologies (SCT) have been considered by government and society as the only means of ensuring competitive advantages for quite some time [1]. Today, in the era of the digital economy, the roles of the supercomputer industry and the supercomputer cyberinfrastructure (SC infrastructure) in leading countries are becoming increasingly important. It is currently important to have an accurate assessment of the current state of the SC infrastructure in these countries, a reliable comparison of their positions in the supercomputer industry, and a clear understanding of modern trends in SCT development.

This article is a continuation of a series of similar analytical reviews by the authors, which use the same methodological approach and toolkit [2, 3]. The novelty of the material presented in this work lies not only in the updating of information based on the latest revisions to the Top500 list, but also in the focus on the following circumstance.

The global supercomputer industry is currently undergoing a radical restructuring, transitioning from the "petaflops era" to the "exascale era". This is not simply about increasing the performance of supercomputers by 1,000 times. In the process of this transition, solutions had to be found to a large number of problems in both hardware and software [4].

Understanding of recent technological trends will make it possible to identify established solutions that have enabled the transition to exascale systems and apply this knowledge in current and future supercomputer solutions.

The article is organized as follows. Section 1 is devoted to Top500 list. In Section 2 we look at pace of supercomputing technology development. Section 3 contains most powerful public supercomputers from June 2011 to November 2022. Trends in the development of hybrid architectures are considered in Section 4. Section 5 is devoted to trends in interconnect technologies. Conclusion summarizes the study.

[1]Ailamazyan Program Systems Institute of Russian Academy of Sciences, Pereslavl-Zalessky, Russian Federation

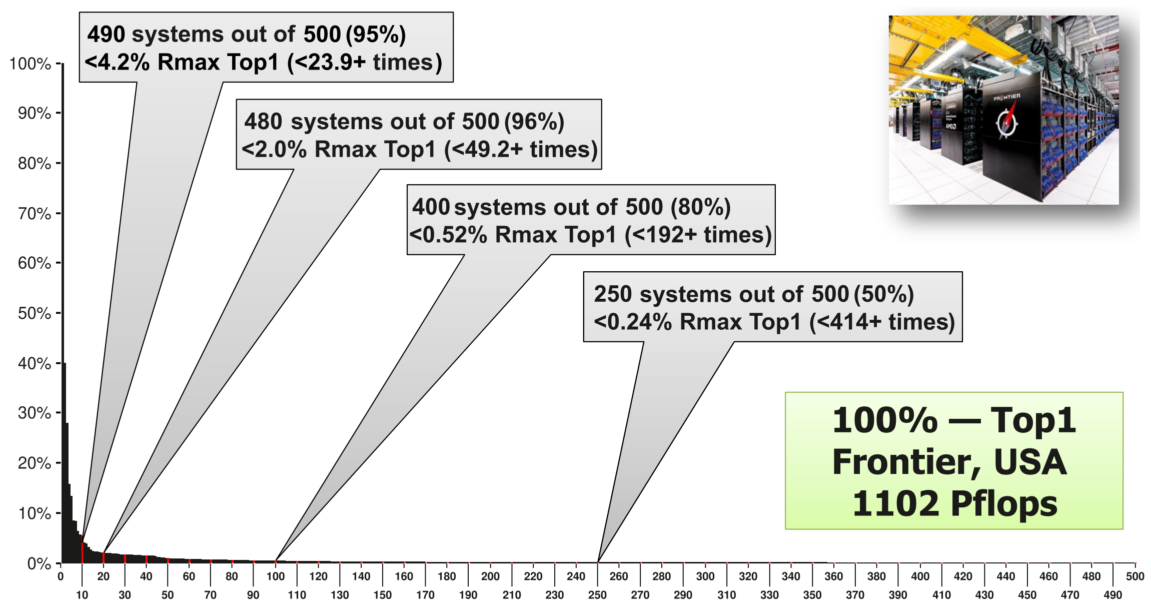


Figure 1. Relative performance of the 500 systems that entered the Top500 ranking in November 2022. The performance of Top1 system is taken as 100%

1. Top500 List

The main source of information for the analysis in this article is the worldwide ranking of the top 500 most powerful supercomputers in the world, known as the Top500 list [5]. The list is updated twice a year, in June and November. Thus, from June 1993 (the first edition) to November 2022, there have been 60 revisions of the list. In the Top500 list, systems are ranked by real performance measured on the Linpack benchmark test – Linpack performance. From now on, we will understand performance only as Linpack – performance, even if it is not specifically emphasized. In the Top500 list, it is denoted as *Rmax*.

Given the fact that some systems are not included in the list for various reasons, the term *supercomputer* here will be defined as follows: *a supercomputer is a computing system that has the performance corresponding to the performance of the machines listed in the corresponding revision of the Top500 list.*

With this definition, the data available on the Top500 website is reliable and multi-dimensional and provides a retrospective description of the state of the supercomputer industry. Moreover, it is detailed, covering all 60 revisions of the list at intervals of six months. Each edition is a table of 500 records. The key aspects of the methodology used to analyze this data set are as follows:

- Supercomputers listed in the Top500 list have a great variety of performance and, as a result, differ significantly in the technical solutions used, price, consumer properties, and so on. The extremely strong "layering" of supercomputers on the example of the November 2022 edition of the Top500 list are illustrated in Fig. 1.
- As a result, it is incorrect to consider supercomputers as "units". It is essential to rely on their key characteristic – their performance. For example, when comparing the equipment of different countries with supercomputer technology, it is necessary to pay more attention to the total performance of the available supercomputers in the country rather than their number.

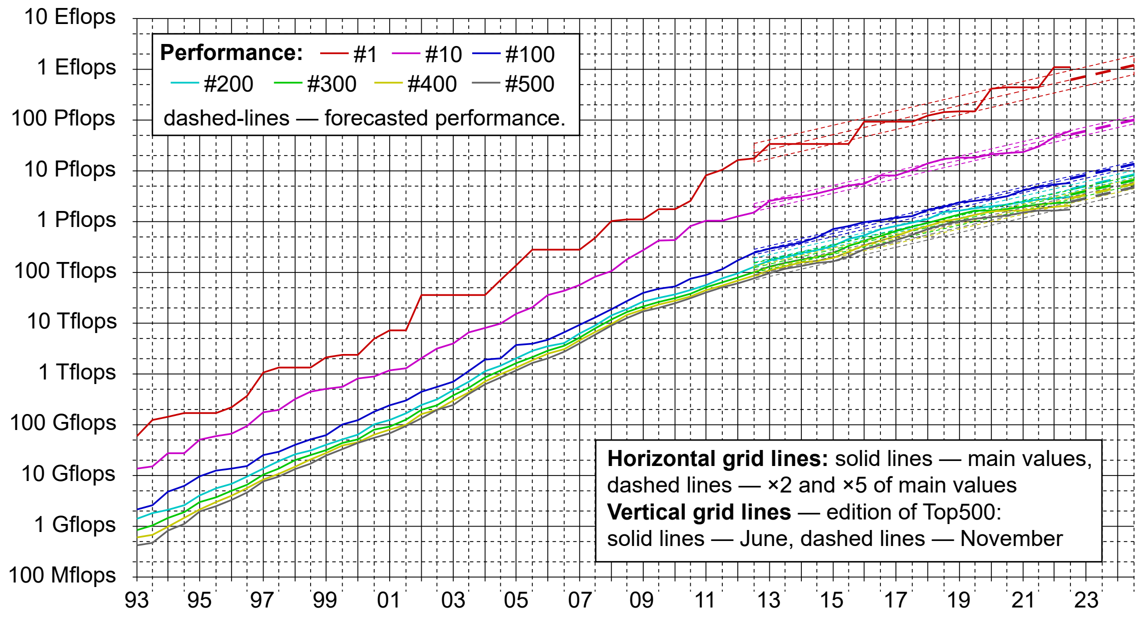


Figure 2. Performance of supercomputers included in various editions of the Top500 ranking

2. Pace of Supercomputing Technology Development

A general assessment of the pace of SCT development can be obtained by analyzing the achieved performance of supercomputers at various points in time. Figure 2 provides the relevant information.

Some explanations are necessary for Fig. 2:

- The X axis shows the times (month and year) of the Top500 editions, and the Y axis shows the performance on a semilogarithmic scale.
- Solid color lines represent performance of systems occupying different positions in the ranking: the top red graph shows the performance of the system in the first place, and the graphs of systems occupying the 10th (lavender), 100th (blue), 200th (light turquoise), 300th (light green), 400th (light grassy), and 500th (dark green) places follow in descending order.
- From June 2012 to November 2024, dotted lines indicate a forecast of the performance of systems occupying the 1st, 10th, 100th, 200th, 300th, 400th, and 500th places in the ranking. The average value of the projection and a 90% confidence interval are indicated. The forecast was calculated using group linear regression (discussed in Section 2.1) over the interval of the last 10 years, from November 2012 to November 2022.
- Colored dots on the graph indicate systems installed in Russia and included in the relevant Top500 edition. Blue dots represent supercomputers purchased abroad, and red dots represent domestic supercomputers.

Graphs similar to those shown in Fig. 2 are often published and serve as an illustration of the exponential pace of development in the SCT industry. Indeed, at first glance, the color lines in Fig. 2 are close to straight lines, which with the logarithmic scale corresponds to an exponential dependence... However, a more careful analysis shows that the graphs are most similar to a broken line consisting of at least two straight segments: before 2008, the linear regression has one slope, and after 2008, another, less steep one.

2.1. Group Linear Regression

Let us describe the methodology of the above-mentioned analysis. Let us number all releases of the Top500 rating by numbers $i \in [1, 60]$. We denote the performance of the system that took a certain place $j \in [1, 500]$ in the Top500 release with number i as $r_{i,j}$. Let us choose a set of places in the Top500 rating $J \subseteq [1, 500]$ and a rating number $k \in [8, 53]$ – here we deviate by 7 releases of the rating to the left and right in the interval $[1, 60]$.

Then by the method of group linear regression for the given k, J, the trend of the performance is determined by finding such a set of parameters a and b_j, where $j \in J$, that the data series $s_j = \{\, (i, \ln r_{i,j}) \mid i \in [k-7, k+7] \,\}$ for $j \in J$ is best approximated by the linear functions $f_j(i) = a \cdot i + b_j$ of the variable i. Specifically, the sum of squares is minimized:

$$\sum_{i \in [k-7, k+7], j \in J} (a \cdot i + b_j - \ln r_{i,j})^2 \to \min.$$

The minimization problem is solved by the method of least squares (LS) and, in essence, this is a simple modification of the standard linear regression algorithm: the regression is carried out for a group of data sets s_j given by the indices $j \in J$, assuming that the slope of all approximating straight lines (the parameter a) is the same, and the offsets along the Y axis (the parameters b_j) are different.

By the meaning of the solved problem, we can say that at the time of the Top500 release number k, in half a year the performance of systems occupying places $j \in J$, in average, increased e^a times. Thus, this performance will increase on average a thousand times during $y(k, J) = \frac{\ln 1000}{2 \cdot a}$ years.

2.2. Over How Many Years Did the Performance of Supercomputers Increase by a Factor of 1000?

The quantity $y(k, J)$ was calculated for several sets of J and for all $k \in [8, 52]$. Figure 3 shows the results of the calculations. The calendar date of the release of the Top500 rating with number k is labeled on the X axis instead of the quantity k. The value of $y(k, J)$ (in years) is indicated on the Y axis.

It can be seen that on the interval from 1998 to 2008, it was indeed possible to say that Rmax increased by a factor of 1000 in approximately 11 years. This was true for both the leading positions in the supercomputing rating (red and purple lines on the graph) and for the entire ranking as a whole (green line). Later (2009–2016), we observe signs of clear technical difficulties that restrained the growth of technologies in the industry. Recently, the situation has been improving for the higher-ranked systems in the ranking, and the negative trends have persisted for the ranking as a whole, i.e., the *gap is widening* between the higher-ranked systems and all others.

2.3. Main Conclusions from the Analysis of Supercomputing Technology Development Rates

We can state that at the turn of 2008, while maintaining the exponential nature of performance growth, the rate of growth decreased – the base of this very exponent decreased. Until 2008, the following was precisely maintained: the performance of the most powerful system in the world doubled every 18 months; its performance increased by a factor of 1000 in 11 years. Thus,

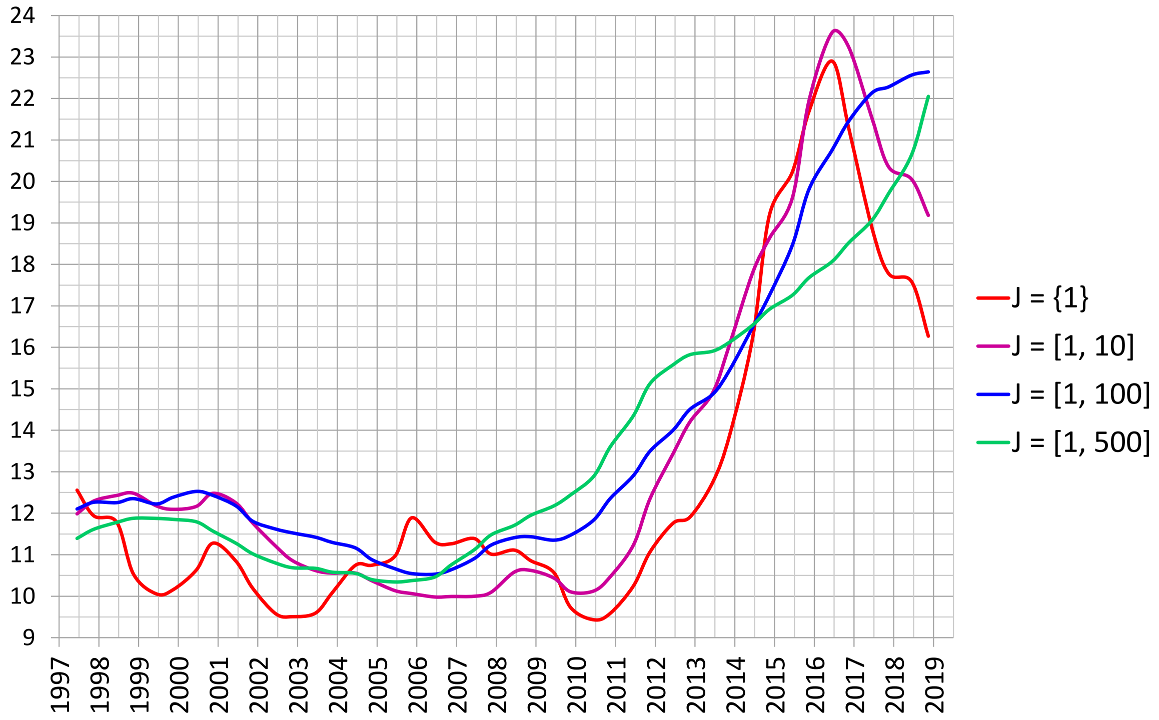


Figure 3. The average forecast for supercomputers occupying positions J is: how long will it take for Rmax to increase by 1,000 times?

the performance milestones were overcome: 1 Mflops (10^6 flops) in 1975, 1 Gflops (10^9 flops) in 1986, 1 Tflops (10^{12} flops) in 1997, and 1 Pflops (10^{15} flops) in 2008. If this trend had persisted, a performance of 1 Eflops (10^{18} flops) would have been achieved in 2019. However, it was achieved in June 2022, see Fig. 2.

Starting from 2008, the rate of growth of achieved maximum performance has clearly changed: instead of an increase of "1000 times in 11 years", we now have an increase of "1000 times in 13–17 years".

Undoubtedly, at the turn of 2008 the world supercomputing industry faced scientific and technical difficulties on the path of SCT development. This led to a review of future performance milestones: 1 Eflops (10^{18} flops) was achieved in June 2022, and one can expect 1 Zflops (10^{21} flops) to be achieved in 2035–2039.

3. Most Powerful Public Supercomputers from June 2011 to November 2022

Table 1 provides brief information about the most powerful supercomputers in the world from June 2011 to November 2022.

Analysis of Tab. 1 reveals obvious trends and allows for reliable (for the next few years) assumptions regarding the general features of exascale systems:

- Number of nodes ranges from 10,000 to 50,000, with 5 to 10 million cores.
- Power consumption ranges from 15 to 30 megawatts.
- Interconnect is, in most cases (87%) of proprietary or customized solutions, or a top standard interconnect with modifications from the manufacturer (13%).
- Cooling systems are closed-water or, in the future, more advanced immersion or boiling.

Table 1. Brief information about the most powerful supercomputers in the world from June 2011 to November 2022

Top500 release, system, ▷ computing subsystem	Rmax, power consumption	Interconnect. Cooling technology
6/2011, K computer, Japan, Fujitsu, RIKEN AICS [6] ▷ ≈**0.7M cores**, 88,128 CPU SPARC64 VIIIfx 8C 2.00 GHz 8-cores, 4 processors per node	10.5 Pflops, 12.6 MW	Custom interconnect. Water cooling
06/2012, Sequoia, USA, BlueGene/Q, IBM [7] ▷ ≈**1.6M cores**, ≈99K CPU Power BQC 16C, 64-bit RISC, 1.6 GHz 16 cores	16.3 Pflops, 7.9 MW	Custom interconnect. Water cooling
11/2012, Titan, USA, Cray XK7 [8] ▷ ≈**0.56M cores**, AMDOpteron – ≈300K cores, NVIDIA K20x – ≈260K cores	17.6 Pflops, 8.2 MW	Custom interconnect. Water cooling
06/2013, Tianhe-2, China, UDT, Inspur [9] ▷ ≈**3.1M cores**, 32K Intel Ivy Bridge + 48K Intel Xeon Phi MIC	33.9 Pflops, 24 MW	Custom interconnect. Water cooling
11/2016, Sunway TaihuLight, China, National Supercomputing Center in Wuxi, NRCPC, Inspur [10] ▷ ≈**10.65M cores**, CPU Sunway SW26010 260C 1.45 GHz, 40,960×(4+256 cores)	93.0 Pflops, 15.4 MW	Custom interconnect. Water cooling
06/2019, Summit, USA, IBM/NVIDIA [11] ▷ ≈**2.4M cores**, IBM POWER9 22C 3.07 GHz – ≈203K cores, NVIDIA Volta GV100 – ≈2.2M cores	148 Pflops, 9.8 MW	Interconnect: Dual-Rail EDR Infiniband. Water cooling
6/2020, Fugaku, Japan, Fujitsu, RIKEN [12] ▷ ≈**7.63M cores**, 158,976 CPUs Fujitsu ARM A64FX 48C 2.2GHz 48 cores	442 Pflops, 29.9 MW	Custom interconnect. Water cooling
6/2022, Frontier (OLCF-5, HPE Cray EX235a), USA, DoE, SC/ORNL, HPE [13] ▷ ≈**8.7M cores**, 9,248 CPUs AMD Optimized 3rd Generation EPYC 64C 2GHz 64 cores, 36,992 GPUs AMD Instinct MI250X 220 cores	1,102 Pflops, 21.1 MW	Custom interconnect Slingshot-11. Water cooling

- Computer might be homogeneous (50%) or hybrid (50%), depending on the purpose.
- Processors and accelerators are either proprietary (44%) or top standard (56%) and hard available.

4. Trends in Development of Hybrid Architectures

Hybrid supercomputers are systems in which computing nodes are equipped with specialized processors, called accelerators, in addition to standard processors. The idea of accelerators in the supercomputing field has been around for a long time, but hybrid architectures have only become widely adopted in the last decade. Fields detailing the use of accelerators in supercomputers (Accelerator and Accelerator Cores) started appearing in the Top500 list beginning in June 2011. Since then, we have been analyzing hybrid architectures and categorizing each supercomputer into the following classes:

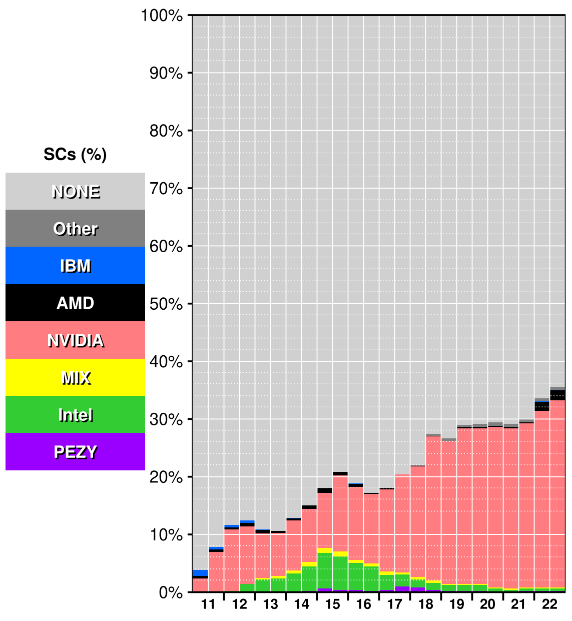
- NONE: accelerators are not used in the supercomputer – homogeneous (non-hybrid) architecture.
- IBM: special-purpose IBM PowerXCell 8i processors are used as accelerators.
- AMD: various specialized processors from AMD are used as accelerators.
- NVIDIA: various specialized processors from Nvidia are used as accelerators.
- Intel: various specialized processors from Intel are used as accelerators – families of Intel MIC / Intel Xeon Phi.
- MIX: various specialized processors from both Nvidia and Intel are used as accelerators.
- PEZY: various specialized processors from PEZY Computing are used as accelerators.
- Other: specialized processors not mentioned above are used as accelerators. Currently, the only accelerators in this category are those developed in China: Matrix/2000 and Deep Computing Processor.

We examined 24 versions of the Top500 ranking from June 2011 to November 2022. Figure 4 shows the proportions of different classes of hybrid supercomputers in the Top500 list for these 24 versions. The labels on the X axis represent years, and the two bar graphs in each year correspond to the two bi-annual versions of Top500 in that year. In the left part of the figure, the proportions are shown in the number of systems (SCs), with each system representing 0.2% of the 500 systems, while in the right part, the proportions are shown in terms of performance (RMax), with the total performance of the entire Top500 list (i.e., ΣRmax) taken as 100%. The following trends can be observed:

- While the number of hybrid supercomputers increased by 12.8% $\nearrow$ 35.6% from 2014 to 2022, their contribution to the total Top500 list performance ΣRmax, which is the true industry share, increased by 34.4% $\nearrow$ 63.8% over this period.
- IBM's accelerators practically disappeared by 2013.
- Intel's accelerator share has fallen by 18.8% $\searrow$ 0.1% ΣRmax from 2013 to 2022.
- NVIDIA's share has increased by 16.4% $\nearrow$ 30.9% ΣRmax from 2014 to 2022.
- AMD's accelerators have practically disappeared by 2017, but then increased by 0.1 $\nearrow$ 31.4% ΣRmax.
- Interesting new solutions that deserve attention have emerged (5.4% ΣRmax in the best, 2018) – short and successful projects to create fully proprietary accelerators:
  - PEZY – 2015, PEZY Computing, Japan;
  - Matrix/2000 – 2017, NUDT, China;
  - Deep Computing Processor – 2017, Sugon, China.

4.1. Quintiles of Supercomputers in the Top500 List

It is very important to understand the applicability and suitability of each technology in supercomputers of various performance levels. For example, it is important to understand in which systems a particular interconnect technology or accelerator class is applicable and in demand today. To do this, we will divide the entire Top500 list (which is sorted by supercomputer performance) into five groups: quintile A – several of the most powerful supercomputers in



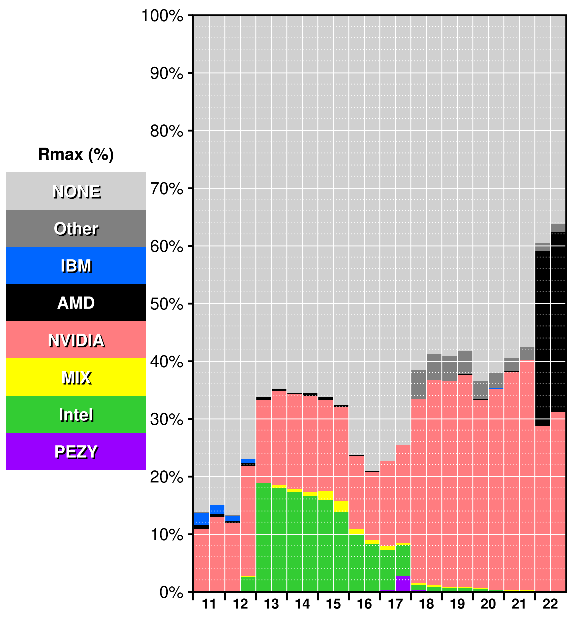


Figure 4. Proportions of different classes of hybrid supercomputers in the Top500 ranking, editions from June 2011 to June 2022

the list, followed by quintiles B, C, D, E. The size of the quintiles will be chosen so that the sum of the performance of all supercomputers in one quintile is as close as possible to 20% of the total performance ΣRmax of the entire Top500 list. That is, quintiles A, B, C, D, and E have (approximately) equal performance, but of course contain different numbers of supercomputers. For each release of the Top500 ranking, the composition of quintiles A, B, C, D, and E (the number of supercomputers in them) is recalculated, based on the requirement of approximating the total performance of each quintile to 20% ΣRmax as accurately as possible.

For the November 2022 edition of the Top500 ranking, the quintile breakdown was as follows:

- A: 1 system, Top1, 22.7% ΣRmax;
- B: 3 systems, Top2–4, 19.0% ΣRmax;
- C: 22 systems, Top5–26, 19.9% ΣRmax;
- D: 111 systems, Top27–137, 20.0% ΣRmax;
- E: 363 systems, Top138–500, 18.4% ΣRmax.

4.2. Distribution of Different Classes of Hybrid Architectures across Quintiles

For each quintile – A, B, C, D, E – let us look at the shares of the None, Other, IBM, AMD, NVIDIA, MIX, Intel, PEZY classes in it (Fig. 5).

By paying attention to the shape and center of gravity of the figures with the corresponding fill color, we can make the following judgments for the November 2022 trends:

- AMD accelerators are predominantly used in the most powerful systems – the entire quintile A and a significant fraction in B;
- NVIDIA accelerators and non-hybrid systems (class NONE) are more or less evenly represented in quintiles B, C, D, E (30%–50% in B, C, D, E);
- other class systems (Chinese accelerators) are mostly represented in quintile C (7%);

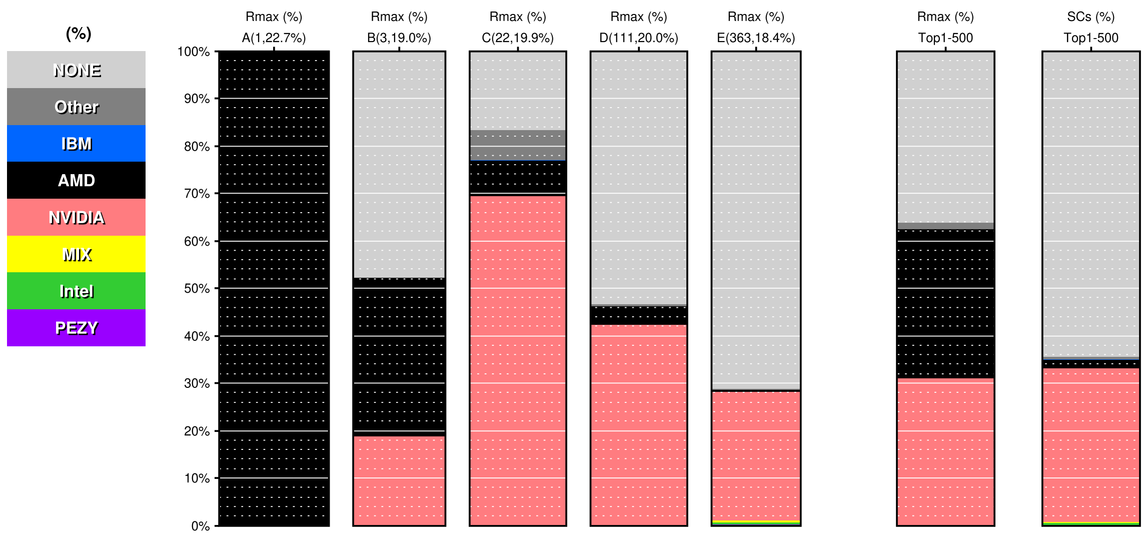


Figure 5. Shares of various classes of hybrid supercomputers in the total performance of quintiles A, B, C, D and E of the Top500 rating November 2022 edition

- MIX, Intel, PEZY accelerators are weakly represented in quintile E – leaving the rating and industry;
- IBM class accelerators have left the industry.

5. Trends in Interconnect Technologies

The interconnect connects all nodes in a supercomputer into a single system. The network technologies used in the interconnect are often different from those used in local, regional, and global computer networks, due to the specific requirements placed on the interconnect: not only high bandwidth, but also minimal latency and maximum message rate, as well as several other specific requirements. In the supercomputing industry, this plays a significant role in determining the interconnect technologies. In the list of 10 critical technological challenges that the industry is facing on the way to exascale systems [14], the development of advanced interconnect technologies occupies a very high (second) place.

Fields that detail the use of various interconnect technologies in supercomputers are present in all editions of the Top500 list. Based on the data in these fields, each supercomputer in all editions of the Top500 list can be classified into the following categories:

- Infiniband: if the supercomputer uses Infiniband technology to implement the interconnect. Such supercomputers have been present in the Top500 list since June 2003 and are still present with several versions of Infiniband having different technical characteristics: Infiniband SDR (8 Gbit/s), Infiniband DDR (16 Gbit/s), Infiniband QDR (32 Gbit/s), Infiniband FDR (54 Gbit/s), Infiniband EDR (100 Gbit/s), Infiniband HDR (200 Gbit/s).
- Ethernet: if the supercomputer uses Ethernet technology to implement the interconnect. Such supercomputers have been present in the Top500 list since June 1996 and with several versions of Ethernet having different technical characteristics ranging from FastEthernet (100 Mbit/s) to 100G Ethernet (100 Gbit/s).

- Myrinet: if the supercomputer uses the Myrinet technology to implement the interconnect. Such supercomputers have been present in the Top500 list since November 1998 until November 2020.
- Quadrics: if the supercomputer uses the Quadrics technology to implement the interconnect. Such supercomputers have been present in the Top500 list since June 1999 until November 2011.
- SCI: if the supercomputer uses the SCI technology to implement the interconnect. Such supercomputers have been present in the Top500 list since June 2002 until November 2004.
- OpTsIntel: if the supercomputer uses the Intel Omni-Path/TrueScale family of solutions to implement the interconnect. Such supercomputers have been present in the Top500 list since November 2013 and are still present.
- Custom: if the supercomputer uses proprietary network solutions or solutions that are not available as a separate commercial product – only available as part of a complete system.

It is important to note the difference between the categories:

- Infiniband, Ethernet, Myrinet, Quadrics, SCI – these interconnect technologies are available as separate commercial products. Any developer of their own supercomputer (for example, with their own architecture of computing nodes, with certain processors) can buy and apply any of these solutions.
- Custom – these interconnect technologies are not available as separate commercial products. They are either unavailable or only available as part of a complete system from a single supplier.
- OpTsIntel – Intel's Omni-Path / TrueScale technologies occupy an intermediate position. On the one hand, these interconnect technologies are available as commercial products. On the other hand, these solutions cannot be applied in a supercomputer if the computing node does not have the appropriate architecture and processor from Intel. And this situation is very close to the situation where the interconnect is only available as part of a complete system from a single supplier – close in terms of the rigidity of technological dependence on a single developer and supplier, and dependence on Intel.

Intel's Omni-Path / TrueScale technology had very high technical characteristics at the time of its appearance, which allowed it to successfully enter the industry and gain a share of 11.0% of the ΣRmax in November 2019. The weak point of the OpTsIntel solution is the rigid technological dependence of the entire system on Intel. And today, this technology is leaving the industry. Figure 6 shows the performance shares of supercomputers with different interconnects in the total performance of the Top500 ranking ΣRmax for 60 editions from June 1993 to November 2022.

Captions on the X axis represent the years, with two bar graphs corresponding to two editions of the list in a particular year.

Analyzing the figure, the following trends of the past 10 years can be noted:

- In recent years, technologies corresponding to classes Myrinet, Quadrics, and SCI are not used in the industry. The figure clearly shows the years of introduction, peak popularity, and decline of these technologies. Today, only 4 classes remain relevant: Infiniband, Ethernet, OpTsIntel, and Custom.
- Since 2012, the highest share of the total performance of the Top500 belongs to supercomputers with Custom interconnects. The dynamics for 2012–2017–2022 are as follows:
  - Custom 54.7% ↘ 42.4% ↗ 53.4%.

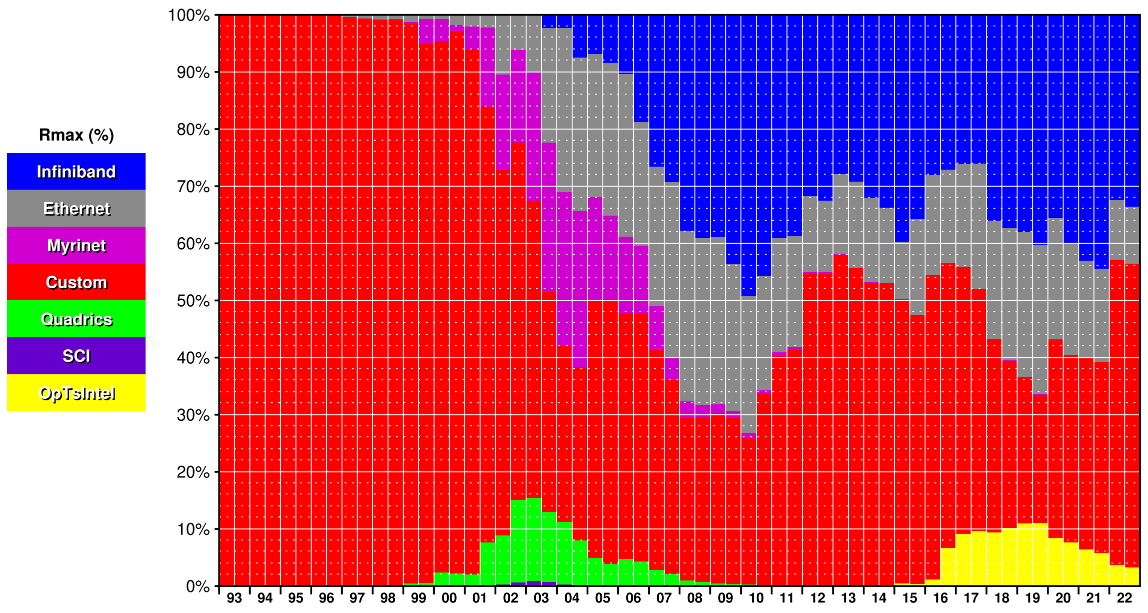


Figure 6. Shares of supercomputers with various interconnects in the total performance of the Top500 ranking, from June 1993 to November 2022

- Next most significant share belongs to Infiniband. The dynamics for 2012–2017–2022 are as follows:
  - Infiniband 32.5% $\searrow$ 26.1% $\nearrow$ 33.6%.
- This is followed by Ethernet. The dynamics for 2012–2017–2022 are as follows:
  - Ethernet 13.2% $\nearrow$ 21.9% $\searrow$ 10.0%.
- OpTsIntel technologies are leaving the Top500 and the industry. The dynamics for 2012–2017–2022 are as follows:
  - OpTsIntel 0.0% $\nearrow$ 9.6% $\searrow$ 3.2%.

5.1. Distribution of Different Interconnect Classes by Quintiles

It is very important to understand the areas of applicability of each interconnect class. To do this, let us look at the distribution of the shares of different interconnect classes by quintiles. The definition of quintiles is given in Section 4.1.

For each quintile – A, B, C, D, E – let us look at the shares of Infiniband, Ethernet, OpTsIntel and Custom interconnect classes – Fig. 7.

It can be seen that:

- Most powerful and mighty supercomputers (quintiles A and B) predominantly use Custom interconnect. The red color fills the triangle, which expands towards the most powerful systems;
- Infiniband technology is predominantly used for medium and lower-level systems – with the largest shares in quintiles C and D. The blue triangle has its center of gravity in this zone;
- Ethernet technology is more often used in the weakest systems (D, E). The grey triangle significantly expands towards the weakest systems;
- OpTsIntel solutions are leaving the list and the industry.

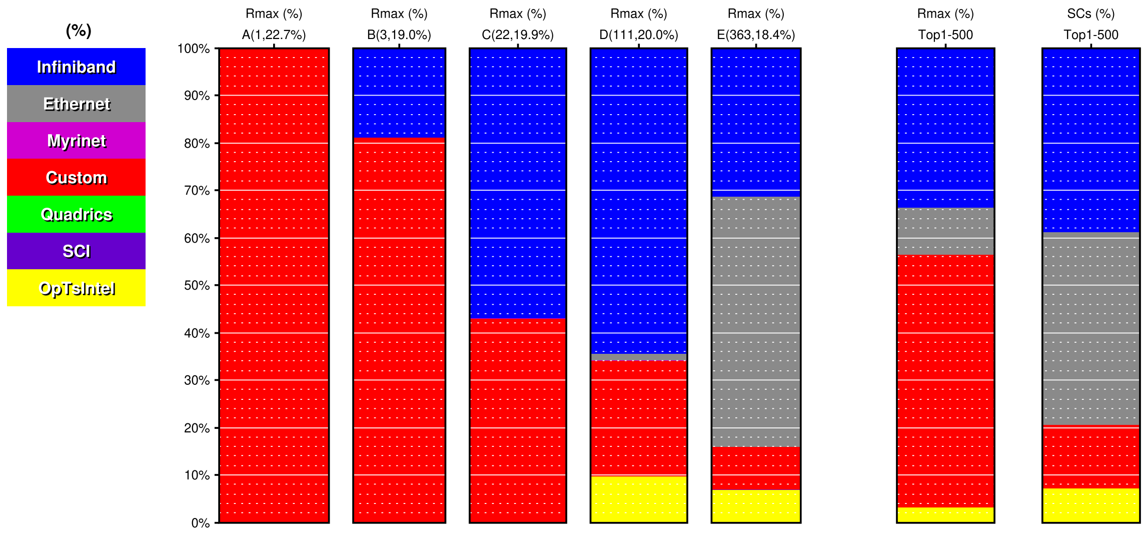


Figure 7. Shares of supercomputers with different interconnect classes in the total performance of quintiles A, B, C, D and E of the Top500 ranking for November 2022 edition

Conclusion

At the turn of 2008 the world supercomputing industry undoubtedly faced scientific and technical difficulties in the development of HPC. Since 2008, the rate of growth of the achieved maximum performance has changed: instead of an increase "1000 times in 11 years", we have an increase "1000 times in 13–17 years". This led to a revision of future forecasts for achieving new performance milestones: 1 Eflops (10^{18} flops) was achieved in June 2022, and 1 Zflops (10^{21} flops) is expected to be achieved in 2035–2039.

Analysis of the most powerful supercomputers of the last decade allows us to make the following generally reliable (for the next few years) assumptions about the overall appearance of exascale-level systems:

- number of nodes – 10–50 thousand, number of cores – 5–10 million;
- power consumption – 15–30 MW;
- interconnect – most likely (7/8 = 87% of cases) Custom (proprietary network, specific solutions or not available as a separate commercial solution), or top (difficult to access) standard interconnect, but with modifications by the manufacturer (1/8 = 13%);
- cooling – either closed water-based or more advanced immersion, boiling, etc.;
- computational subsystem – depending on the purpose – either homogeneous (50%), or hybrid (50% of cases);
- processors and accelerators – either proprietary (44%), or top (limited availability) standard (56% of cases).

Analysis of hybrid architectures in the Top500 rankings indicates that the contribution of hybrid architectures to the overall performance of the Top500 has grown and reached 63.8% ΣRmax. Among the current solutions for hybrid systems, notable accelerators include AMD (31.4% ΣRmax) and NVIDIA (30.9% ΣRmax). Interesting new solutions also deserve attention (5.4% ΣRmax in 2018) – short and successful projects for creating fully proprietary accelerators: PEZY (2015, PEZY Computing, Japan), DeepComputingProcessor (2017, Sugon, China), Matrix-2000 (2017, NUDT, China).

Various interconnect technologies continue to intensively develop in the world. They play an exceptionally important role for the entire HPC complex. Currently, commercially available technologies include Custom (53.4% ΣRmax), Infiniband (33.6% ΣRmax), and Ethernet (10.0% ΣRmax). Since 2012, the largest share of the total performance of Top500 supercomputers is for the interconnects that are not available as individual commercial solutions (Custom class).

For building high-performance systems, top-end (and therefore limited availability) models of the most successful commercial developments such as AMD processors are advanced, as well as proprietary processors based on one of the available architectures, such as ARM (which has proven to be promising for this use in supercomputers) and possibly RISC-V.

Significant changes have taken place among computer manufacturers in recent years. Chinese manufacturers have made a serious intervention in the industry, they accounted for 63.0% of the number of deals and 43.4% of volumes (in terms of ΣRmax) in November 2019. China has the largest share of the entry-level systems.

The following fact is important for building high-performance computing systems:

- they are always used for research and development; for generating new knowledge, technologies, materials; for scientific calculations;
- they are never used directly for real-world economic tasks, such as engineering calculations, financial applications, communication services, internet services, etc.

This has an impact on the class of problems solved: on the cutting edge of scientific research during the lifespan of the record installation (about 5–7 years), special problems may arise that require special mathematical and algorithmic tools for their solution. This should be taken into account when developing architectural solutions, both in terms of hardware and software for such systems.

Currently, the US, China, Japan, and the EU have built a solid HPC infrastructure as a basis for the transition to a digital economy and have significant shares of their countries' overall global supercomputer performance. These shares are greater than their shares in the global GDP (with the exception of China, which hides its achievements in HPC). This ratio illustrates the real progress of countries towards a digital economy.

References

1. Ezell, S.J., Atkinson, R.D.: The Vital Importance of High-Performance Computing to U.S. Competitiveness. The Information Technology and Innovation Foundation (2016). 58 p.

2. Abramov, S.M.: Facts that distort reality. How to analyze Top500? Bulletin of South Ural State University: Series Computational Mathematics and Software Engineering 2(3), 5–31 (2013). `https://doi.org/10.14529/cmse130301` (in Russian)

3. Abramov, S.M.: June 2019: analysis of the development of the supercomputer industry in Russia and in the world. Program Systems: Theory and Applications 42, 3–40 (2019). `https://doi.org/10.25209/2079-3316-2019-10-3-3-40` (in Russian)

4. United States Department of Energy. Top Ten Exascale Research Challenges. Department of Energy ASCAC Subcommittee Report (2014). 86 p.

5. Top500 Official cite. https://www.top500.org/, accessed: 2023-06-06

6. Supercomputer “K computer” Takes First Place in World. https://www.fujitsu.com/global/about/resources/news/press-releases/2011/0620-02.html, accessed: 2023-06-06

7. With 16 petaflops and 1.6M cores, DOE supercomputer is worlds fastest. https://arstechnica.com/information-technology/2012/06/with-16-petaflops-and-1-6m-cores-doe-supercomputer-is-worlds-fastest/, accessed: 2023-06-06

8. ORNL Debuts Titan Supercomputer. https://www.olcf.ornl.gov/wp-content/themes/olcf/titan/Titan_Debuts.pdf, accessed: 2023-06-06

9. Chinas Tianhe-2 Caps Top 10 Supercomputers. https://spectrum.ieee.org/tianhe2-caps-top-10-supercomputers, accessed: 2023-06-06

10. Dongarra, J.: Sunway TaihuLight supercomputer makes its appearance. National Science Review 3(3), 265–266 (2016). https://doi.org/10.1093/nsr/nww044

11. Oak Ridge National Laboratory, IBM, NVIDIA: Introducing the Summit Supercomputer (2020). 24 p.

12. Dongarra, J.: Report on the Fujitsu Fugaku system. University of Tennessee-Knoxville Innovative Computing Laboratory, Tech Report ICLUT-20-06 (2020). 18 p. https://www.icl.utk.edu/files/publications/2020/icl-utk-1379-2020.pdf, accessed: 2023-06-06

13. Frontier spec sheet. https://www.olcf.ornl.gov/wp-content/uploads/2019/05/frontier_specsheet.pdf, accessed: 2023-06-06

14. Coghlan, S.: Argonnes Aurora Exascale Computer. Smoky Mountain Computational Sciences And Engineering Conference (2019). 17 p. https://smc.ornl.gov/wp-content/uploads/2019/09/Coghlan-presentation-2019.pdf, accessed: 2023-06-06

DOI: 10.14529/jsfi230202

High-Level Synthesis Toolchain "Theseus" for Multichip Reconfigurable Computer Systems

Aleksey I. Dordopulo[1]***, Ilya I. Levin***[2]***, Vyacheslav A. Gudkov***[1,2]***, Andrey A. Gulenok***[1]

In the paper we consider the high-level synthesis toolchain for transformation of programs written in C (the standard ISO/IEC 9899:1999) into configuration files of field programmable gate arrays (FPGAs) used in multichip reconfigurable computer systems. Unlike most academic (DWARV, BAMBU, LEGUP) and commercial (CatapultC, Vivado HLS, Vivado Vitis) high-level synthesis tools, "Theseus" uses the original methodology of transformation (porting) sequential calculations into a parallel-pipeline configuration of FPGA hardware. For a sequential program, an information graph is created and transformed into the maximally parallel structure, which is then ported to a specified configuration of the reconfigurable computer system using formal methods of reduction of performance and hardware costs without marking the source text with auxiliary parallelization directives. The distinctive feature of the approach is a significantly smaller number of analyzed variants in comparison to parallelizing compilers. Due to this, it is possible to reduce the porting time of sequential programs in the synthesis of solutions for reconfigurable computer systems with a set of FPGA chips interconnected by a spatial communication system. In the paper we show the results of porting a number of application tasks to the architecture of various reconfigurable computer systems using the proposed "Theseus" toolchain.

Keywords: high-level synthesis, HLS, program translation, C language, performance reduction, reconfigurable computer system, programming of multiprocessor computer systems.

Introduction

Reconfigurable computer systems (RCS) containing field programmable gate arrays (FPGAs) provide adaptation of the system's architecture to the task's structure and reduction of the additional charges for organization of calculations. This provides a significant gain in task solution time in comparison to multiprocessor systems [1] even with a 10-fold difference in operating frequencies. RCSs that contain many FPGA chips connected by a switching system [2, 3], significantly exceed the cluster computer systems in real performance of applications and power effectiveness on many real-life tasks, but their programming and debugging require extensive and deep knowledge of the FPGA circuitry and architecture from programmers. Simplification of RCS programming is possible with the development of high-level synthesis software [4, 5], which converts sequential programs, written in high-level languages, into FPGA configuration files.

In this paper we consider the description of methods, used for automatic transformation of a sequential program by "Theseus" tools for high-level synthesis of multichip RCS configuration files. Section 1 of this paper provides a brief overview of high-level synthesis tools and assessment of their applicability for multichip RCSs. Section 2 describes the structure of the Theseus toolchain and intercomponent communication during synthesis of the multichip solution from the initial program. Section 3 presents methods of transformations of sequential programs for automatic adaptation of tasks to the available RCS hardware resource for each component of the toolchain. Section 4 presents "Theseus" porting results for a number of tasks compared with Vivado HLS results. In conclusion, we analyse the obtained results.

[1]Supercomputers and Neurocomputers Research Center, Taganrog, Russian Federation
[2]Southern Federal University, Taganrog, Russian Federation

1. Overview of High-Level Synthesis Tools

Currently, high-level synthesis tools or HLS compilers [4, 5] are widely used for programming FPGAs. Such tools convert a high-level program into configuration files of special-purpose hardware, using HDL hardware description languages. Depending on the language of the input program, HLS compilers refer to translators of either problem-oriented(for a certain problem area) or general-purpose languages.

The classification of high-level synthesis tools according to these criteria is shown in Fig. 1. Here, $\triangleright$ means that currently an HLS compiler is developed and supported, $\|$ means that a project is suspended and there is no information concerning future development plans, $\square$ means that a project is finished and not supported anymore.

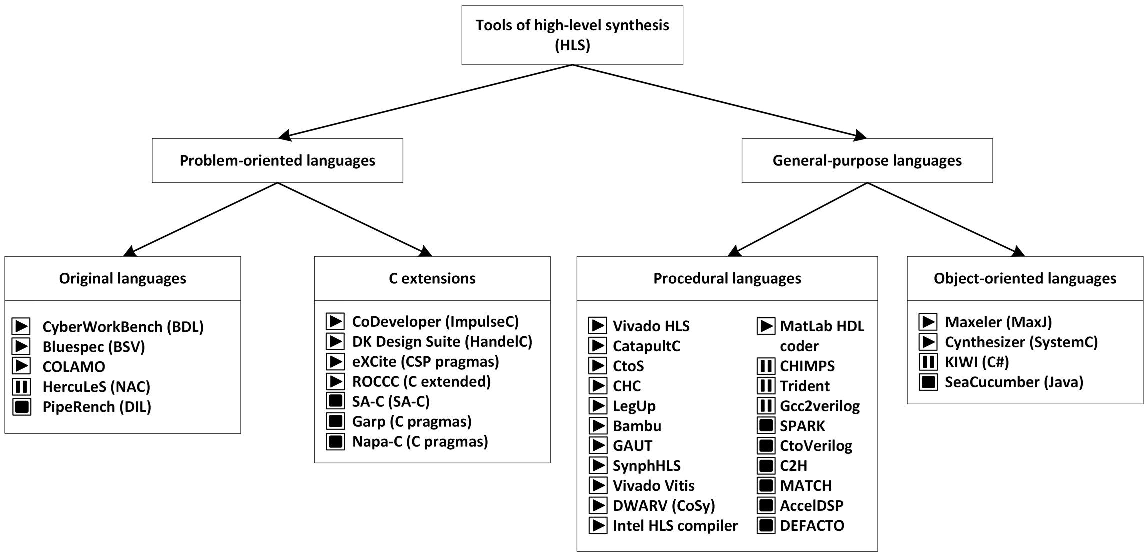


Figure 1. Classification of high-level synthesis tools

The most well-known academic (DWARV [6], BAMBU [7], LEGUP [8]) and commercial (CatapultC, Vivado HLS [9], Vivado Vitis [10]) HLS compilers convert written in C (or its dialect) program into a VHDL program of digital devices. Xilinx Vivado HLS and Vitis computer-aided design systems have become generally accepted standards of high-level synthesis tools.

Vivado HLS [9] is a tool for fast project design. It contains a number of optimization tools, typical for both compilers and digital circuit design systems [11]. Xilinx Vitis [10] is a development environment that combines graphical tools, compilers, analyzers and debuggers to speed up the execution of code fragments of sequential programs implemented in the FPGA architecture. Vitis is focused on Xilinx FPGA-based accelerators for server and cloud applications and/or Alveo accelerators for embedded devices.

The use of HLS compilers does not guarantee [11] an effective implementation of calculations in RCS for any program written in the C language. For a synthesized IP core, the gain in the speed of calculations compared to a general-purpose microprocessor is provided by the properties of the FPGA architecture. Computer-aided design of an IP core simplifies the porting of calculations to the FPGA architecture, and the programmer must scale IP cores according to the available hardware resource. There are no computer-aided tools of scaling IP cores and organizing data flows (at least within one chip), and this task is assigned to the user. For multichip RCS [2, 3], where many FPGAs are connected by a spatial switching system, the complexity of scaling and matching of IP cores for an efficient solution is rising significantly [12].

Unlike the HLS tools mentioned in [4, 5], the developed toolchain "Theseus" converts a sequential C program into the most parallel form, which is adapted to a specified hardware computing resource and translated into FPGA configuration files of multichip RCS. The toolchain converts the input program without users **#pragma** directives or other manual code marking and provides automatic synchronization of data and control signals for synthesizing multichip solutions. The basis for automatic adaptation of an application to the architecture and configuration of a specified RCS is the original porting methodology, which provides the search for an efficient solution for a priori unknown hardware resource of the computer system.

2. Structure of the "Theseus" Toolchain

The "Theseus" toolchain (Fig. 2) consists of four programs called Angel, Centaur, Procrustes and Sinis, each of which performs a functionally completed transformation:

– Angel converts the input C program into the maximally parallel structure, and then translates it with an implicit description of parallelism into the syntax of the COLAMO programming language;
– Centaur converts the maximally parallel structure into a resource-independent parallel-pipeline COLAMO-program;
– Procrustes automatically, with no users code, selects the parameters of the structure for its efficient implementation in the RCS architecture;
– if the hardware resource is insufficient for structural implementation of the main fragment of the task, then Sinis reduces the parallelism of the structure, increasing the task solution time, but making it possible to execute on the available hardware resource.

The input sequential C program (ISO/IEC 9899:1999) is translated by Angel into the maximally parallel structure, which is then converted into a scalable form and ported by Procrustes to the available RCS hardware resource. The result of the "Theseus" toolchain is a program written in the high-level language COLAMO with the parameters that match the limitations of the resource and real performance rate. The COLAMO [2] translator and the multichip solution synthesizer Fire!Constructor [2] translate the COLAMO program into configuration files for FPGAs of the multichip RCS. Then, the Xilinx Vivado synthesizer generates bitstream configuration files (* .bit) for each chip.

3. Transformation of a Sequential Program by the "Theseus" Toolchain to Available RCS Resource

3.1. Angel: Transformation of an Input Sequential Program into Maximally Parallel Structure

The sequential written in C program is transformed by the Angel translator into the maximally parallel form – the task information graph, which is then transformed into the COLAMO program with an implicit description of parallelism. The task information graph (TIG) is a finite oriented acyclic graph whose vertices correspond to operations on data, and arcs reflect the data dependencies between them. All vertices of the TIG are distributed in layers and iterations [2]. Layers, like the layers of the algorithm graph [13], contain data-independent vertices (subgraphs), and iterations describe the data dependence among vertices (subgraphs) of different layers. Unlike other graph forms used for the representation of calculations [13], layers and

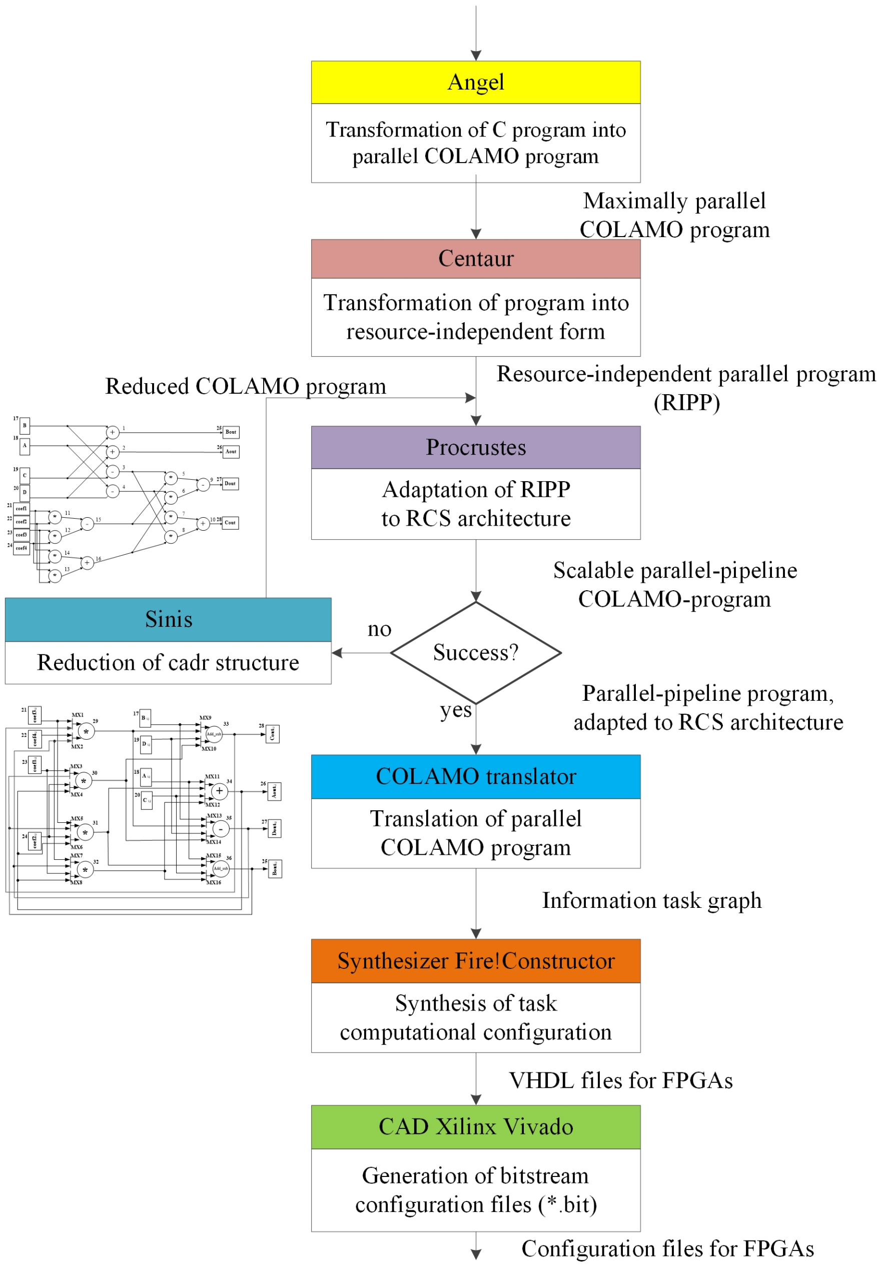

Angel
Transformation of C program into parallel COLAMO program
Maximally parallel COLAMO program
Centaur
Transformation of program into resource-independent form
Reduced COLAMO program
Resource-independent parallel program (RIPP)
Procrustes
Adaptation of RIPP to RCS architecture
Scalable parallel-pipeline COLAMO-program
Sinis
Reduction of cadr structure
no
Success?
yes
Parallel-pipeline program, adapted to RCS architecture
COLAMO translator
Translation of parallel COLAMO program
Information task graph
Synthesizer Fire!Constructor
Synthesis of task computational configuration
VHDL files for FPGAs
CAD Xilinx Vivado
Generation of bitstream configuration files (*.bit)
Configuration files for FPGAs

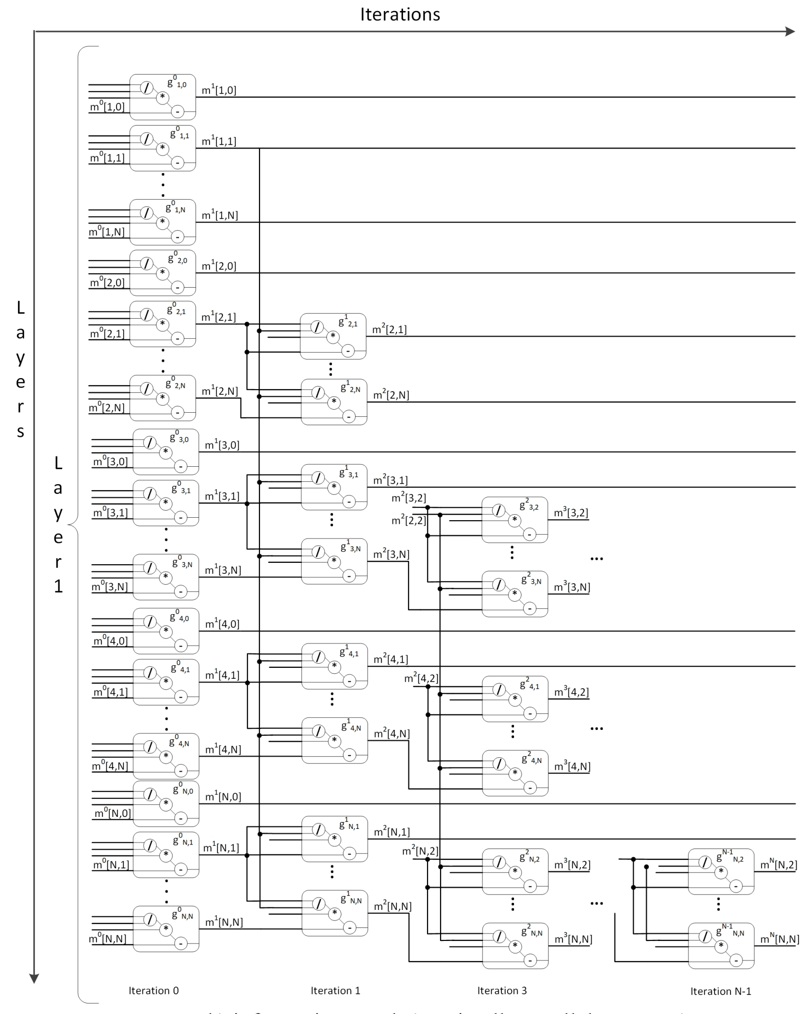
Figure 2. Structure of the "Theseus" toolchain

iterations of the TIG describe not only arithmetic and logical operations, but also subgraphs corresponding, for example, to a loop body in sequential programs. It is rather easy to generate a TIG from a sequential program by cycles unrolling. The TIG describes the maximum, theoretically possible, parallel form of the tasks calculations with the subgraphs that are distributed across layers and iterations.

Owing to the distribution of vertices and/or subgraphs by layers and iterations, it is possible to represent data dependencies among the task's fragments at different levels of the hierarchy. The variants of the sequential program, differing in the execution order of the cycle's iterations (subgraphs), differ in the TIG only in the topology of data-independent subgraphs in the layer, and are considered data-indistinguishable or equivalent. Therefore, sequential programs describing the same task with the same data dependencies, which are different in syntax and form, will correspond to the TIGs that are equivalent in the results of calculations, but differ only in the topology of subgraphs in the layer. Due to the invariance of the representation of topologically different parallel calculations in the TIG, we can, unlike a parallelizing compiler, reduce the number of possible variants for their representation. C program which solves a system of linear algebraic equations (SLAE) by the method of Gaussian elimination is shown in Fig. 3a and its TIG is shown in Fig. 3b.

```
for (i = 0; i < N; i++) {
  for (j = i+1 ; j < N ; j++) {
   d = (m[j][i]/m[i][i]);
    for (k = i; k < N+1 ; k++)
         m[j][k] = m[j][k] - (m[i][k]*d);
  }
  }
```

a) fragment of a sequential program



b) information graph (maximally parallel structure)

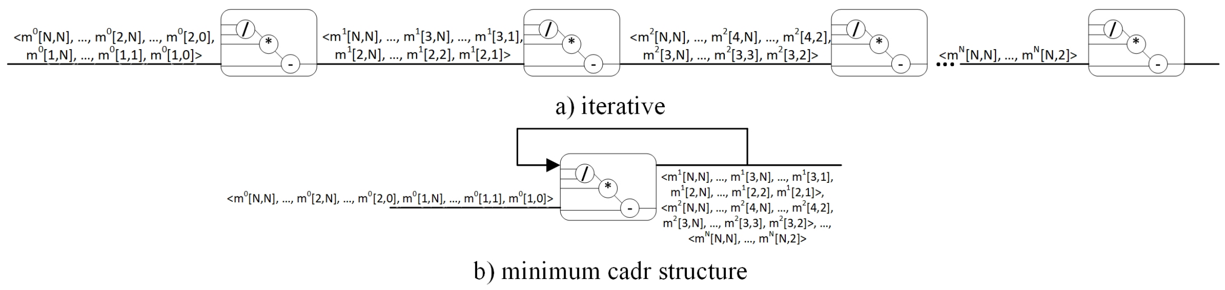
Figure 3. Solution of a SLAE using the method of Gaussian elimination

The subgraphs g^i_{jk} in Fig. 3b contain 3 operation vertices: division, multiplication, and subtraction, corresponding to the operations of the loop bodies on **j** and **k** in lines 3 and 5 in program Fig. 3a. The data-independent subgraphs g^i_{jk} belong to the layers, and dependence among subgraphs of different layers is specified by connections among subgraphs: the output $m^1[1,1]$ of the subgraph $g^0_{1,1}$ of the zero iteration is the input of the subgraphs $g^1_{(2..N,1..N)}$ of the first iteration, the output $m^2[3,2]$ of the subgraph $g^1_{3,2}$ is the input of $g^2_{(3..N,2..N)}$ of the second iteration, etc.). The specificity of the SLAE solution by the method of Gaussian elimination is reducing the processed data by one line at each iteration of the loop on **j**, so each next layer of the TIG contains fewer subgraphs g^i_{jk}.

The TIG is transformed by the Angel translator into a *cadr* structure [2]. Cadr structure is an indivisible unity of computational structure and rules for organizing input and output data flows, for which a reduction of the computational structure leads to an increase in data streams. So, for the same task a variety of cadr structures differing in computational structures and data flows is possible. The Angel translator transforms TIG into a maximally parallel cadr structure: the operation vertices are replaced by computing devices, and the arcs are replaced by the connections of the switching system. The maximally parallel structure contains not the vertices of the graph, but computing devices with latency, frequency and performance, data processing interval, etc. Due to this, it is possible to calculate the performance characteristics and the execution time on the RCS. The maximally parallel structure, obtained as a result of transformation of a sequential program by the Angel translator, is described according to the rules of the COLAMO programming language with an implicit description of parallelism.

3.2. Centaur: Transformation of the Maximally Parallel Structure into the Resource-Independent Parallel-Pipeline Form

Maximally parallel structure performs all task operations with the minimum latency and maximum performance. This requires the hardware implementation of all operations and simultaneous supply of all input data, which is usually unattainable for real computer systems. For implementation in a computer system, the maximally parallel structure can be transformed into cadr structures that are more rational in terms of the occupied hardware resource and provide data equivalence of the calculation results. For example, the maximally parallel structure (Fig. 3b) can be transformed to a cadr structure with all hardware-implemented iterations (Fig. 4a) by defining the order of execution of its subgraphs (ordering by layers). With further ordering of the subgraphs (by iterations), the cadr structure with the implemented iterations (Fig. 4a) can be transformed to a minimum cadr structure (Fig. 4b) with a single hardware-implemented (basic) subgraph.



<m^0[N,N], ..., m^0[2,N], ..., m^0[2,0], m^0[1,N], ..., m^0[1,1], m^0[1,0]>
/
*
-
<m^1[N,N], ..., m^1[3,N], ..., m^1[3,1], m^1[2,N], ..., m^1[2,2], m^1[2,1]>
<m^2[N,N], ..., m^2[4,N], ..., m^2[4,2], m^2[3,N], ..., m^2[3,3], m^2[3,2]>
<m^N[N,N], ..., m^N[N,2]>
a) iterative
<m^0[N,N], ..., m^0[2,N], ..., m^0[2,0], m^0[1,N], ..., m^0[1,1], m^0[1,0]>
<m^1[N,N], ..., m^1[3,N], ..., m^1[3,1], m^1[2,N], ..., m^1[2,2], m^1[2,1]>, <m^2[N,N], ..., m^2[4,N], ..., m^2[4,2], m^2[3,N], ..., m^2[3,3], m^2[3,2]>, ..., <m^N[N,N], ..., m^N[N,2]>
b) minimum cadr structure

Figure 4. Cadr structures of the SLAE solution by the Gaussian method

For convertion of the maximally parallel structure into one of many possible cadr structures, it is transformed by the Centaur tool into a resource-independent parallel-pipeline form, which makes possible to change the number of hardware-implemented subgraphs using several parallelism parameters. Centaur identifies basic subgraphs for large fragments of calculations, analyzes the data dependencies in these fragments and among them, ensuring the implementation of the rules of single substitution and single assignment. Arbitrary access to the memory of a sequential program is transformed to data flow processing. All arrays and loops, used in the descriptions of the cadr structures in the COLAMO program, are automatically split into parallel (vector) and sequential (stream) components. As a result, Centaur generates a COLAMO program containing a cadr structure with parallelism parameters. Variation of these parameters transforms the cadr structure.

3.3. Procrustes: Transformation of Resource-Independent Cadr Structure to Available RCS Hardware Resource

Cadr structures (see Fig. 3b, 4a and 4b) differ in their hardware resource and the task solution times. Parallelism, solution time and hardware resource occupied by the cadr structures are specified by the following parameters: the number of layers (information-independent subgraphs), the number of iterations (information-dependent subgraphs), the number of instructions (devices) in the basic subgraph, the capacity and interval of data processing. Therefore, each cadr structure can be represented by a point (or vector) in the 5D space of possible cadr structures (Fig. 5).

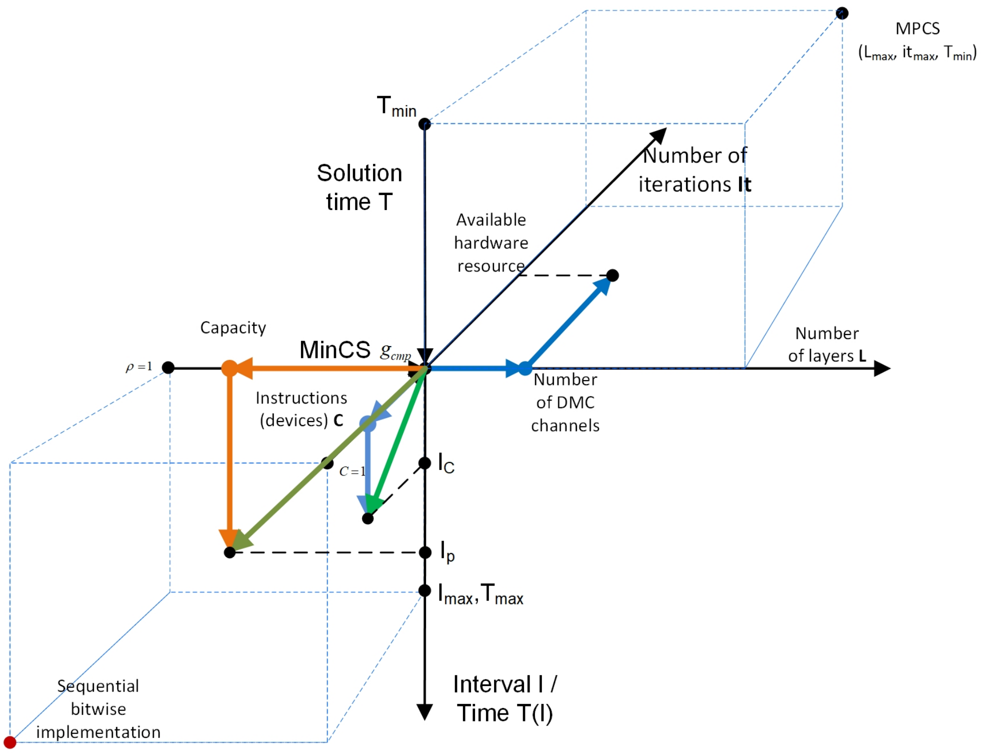


Figure 5. Space of implementations of cadr structure calculations

Space points correspond to cadr structures with various parameters of parallelism (performance) and occupied hardware resource. Variation of the performance parameters (shown by colour arrows in Fig. 5) changes the hardware resource occupied by the cadr structure, which is graphically represented as a segment between two points of space. So, porting is the variation of the performance parameters and the occupied hardware resource performed to achieve the available hardware resource. Porting is a continuous movement from the maximally parallel structure to a cadr structure located in the area of the available resource of the computer system. Porting can be defined as the sequential approaching to local and global goals. The local goal is to reach the area of available resource, and the global goal is to find a rational cadr structure that provides a specified performance rate. The search for the performance parameters of the cadr structure that satisfy both the limitations of the available RCS hardware resource and the specified performance rate is provided by Procrustes with the help of the following methodology.

According to the methodology, a cadr structure CS with hardware costs $A_{CS} = (a_1^{CS}, a_2^{CS}, \ldots, a_n^{CS})$ and performance parameters $P_{CS} = (p_1, p_2, \ldots, p_n)$ is transformed to the hardware resource $A_{HCS} = (a_1^{HCS}, a_2^{HCS}, \ldots, a_n^{HCS})$ of a reconfigurable or hybrid computer system. It should be noted that the hardware costs A_{CS} of the cadr structure non-linearly depend on the performance parameters P_{CS}: $A_{CS} = \Psi(P_{CS})$, and one performance parameter p_i may simultaneously influence several parameters of the hardware resource.

The purpose of porting is to detect the performance parameters P'_{CS}, which are the solution of

$$\begin{cases} A'_{CS} = \Psi(P'_{CS}) \leq A_{HCS}, \\ Perf(P'_{CS}) \geq Perf_3, \end{cases}$$

where $Perf(P'_{CS})$ is the performance of the cadr structure, and $Perf_3$ is the specified real performance rate.

To achieve the purpose, the following actions are required.

```
1. Calculation of reduction parameters
  1.1. Determine the critical resource and the reduction coefficient of hardware costs
       $K_{cr} = max\left(\frac{a_i^{CS}}{a_i^{HCS}}\right) > 1$.
  1.2. Determine the reduction coefficient $R = \lceil K_{cr} \rceil$.
  1.3. Arrange the performance parameters $p_i$ in the tuple $< P_{CS} >$ according to the degree
       of influence on the critical resource.
  1.4. If <an unreduced parameter exists in $< P_{CS} >$>, then
       – select the performance parameter with the maximum impact on the critical resource:
       $p_i : \psi(p_i) = maxK_{cr}$.
       else: go to item 7 to select a card structure with the maximum performance.
2. Determination of the effective reduction step $R^*$.
  2.1. For the selected performance parameter $p_i$, determine one of the possible options for
       the effective reduction step $R$:
       1) $R^* = R$;
       2) $R^* = f(R, p_i), R^* < R$;
       3) $R^* = \|p_i\|, R^* > R$.
3. Reduction (decreasing of the performance parameters) of the cadr structure
  3.1. Reduce the parameter $p_i$ with the effective step $R^*$: $p'_i = \Theta(p_i, R^*)$ and save it to a
       tuple: $< P'_{CS} >=< P_{CS} >\leftarrow p'_i$.
4. Evaluation of the current cadr structure and selection of alternatives
  4.1. Calculate hardware costs and performance of the cadr structure
       $A'_{CS} = \Psi(P'_{CS}, Perf(P'_{CS}))$.
  4.2. Evaluate the achievement of porting purpose from the conditions
       $A'_{CS} = \Psi(P'_{CS}) \leq A_{HCS}$ and $Perf(P'_{CS}) \geq Perf_3$.
  4.3. If both conditions are fulfilled, then
         – go to item 6 to save the current parameters of the cadr structure
       else: {Analysis of alternatives for further reduction:}
       If $A'_{CS} > A_{HCS}$ {the hardware costs exceed the available RCS resource}, then:
       If $R^* = R$, then
         – increase the reduction coefficient $R := R + \Delta$, where $\Delta = \left\lceil \frac{A'_{CS}}{A_{HCS}} - 1 \right\rceil$;
         – restore the original value of the performance parameter $< P'_{CS} >=< P_{CS} >\leftarrow p'_i$;
         – go to item 1.3 for reduction with the increased coefficient $R$.
       Else
         – go to item 1 to decrease the critical resource using the next performance parameter
           $p_{i+1}$.
```

{if $A'_{CS} \le A_{HCS}$ and $Perf(P'_{CS}) < Perf_3$, then we go naturally to induction}

5. **Induction (increasing of the previously reduced performance parameters) of the cadr structure**
   - 5.1. Form a tuple of the previously reduced parameters $< p_{i-1}, \dots, p_1 >$.
   - 5.2. Select the next item of the tuple p_k, determine its scaling coefficient according to the hardware resource $K_M = min\Big(\frac{A'_{CS}}{A_{HCS}}\Big)$ and the induction coefficient $Ind = \lfloor K_M \rfloor$.
   - 5.3. For the selected performance parameter p_k, determine one of the possible effective induction steps:
     1) $Ind^* = Ind$;
     2) $Ind^* = \phi(Ind, P_k) < Ind$;
     3) $Ind^* = \|p_k\|$.
   - 5.4. Increase p_k by the selected step Ind^*: $p'_k = \Theta^{-1}(p_k, Ind^*)$ and save it into the tuple: $< P'_{CS} > = < P'_{CS} > \leftarrow p'_k$.
   - 5.5. Go to item 4.
6. **Save the cadr structure**
   - 6.1. Add the current parameters of the cadr structure to the list, ordered by the minimum performance.
   - 6.2. **If** $Perf(P'_{CS}) < Perf_3$ **and** <there were other critical resources>, **then**
     – select the next critical resource and go to item 1.1.
7. **Select a reasonable variant for reduction of the cadr structure**
   - 7.1. Issue the first item of the list of cadr structures with the highest performance.

Unlike most HLS compilers [4, 5], the proposed methodology changes the parameters of the cadr structure in order to provide the rational use of available hardware resource and to achieve a specified rate of real performance. If the specified performance rate $Perf_3$ is not achievable, the result of the porting will be the cadr structure with the highest performance among all analyzed cadr structures. Unlike the methods of structural and procedural organization of calculations, which provided a rational cadr structure for a priori known and fixed resource, the transformation, according to the Procrustes' methodology, is a continuous function which depends on the architecture and configuration of the computer system. This fact ensures movement in the space of cadr structures not only "down", towards reducing parallelism (reduction), but also towards its increase (induction), if the hardware resource allows. Due to combination of reduction and induction, it is possible to find rational performance parameters of cadr structures even if the hardware resource is insufficient for the hardware implementation of the basic subgraph.

The total number of analyzed variants depends on the number of the performance parameters of the cadr structure that effect hardware costs. For the FPGA architecture, the number of memory channels, the number of Look-Up Table cells in FPGA chips, the number of Block RAM internal memory blocks, the number of High Bandwidth Memory blocks, the number of Digital Signal Processor blocks, and the number of Flip-Flop registers may vary in the cadr structure. Therefore, for FPGAs, the total number of analyzed variants of rational cadr structures with different parameters is small and does not exceed $6! = 720$.

Procrustes generates a parallel-pipeline COLAMO program, which contains the cadr structure with the performance parameters calculated for the available resource of the specified RCS. This program is transformed by the COLAMO translator and the Fire!Constructor synthesizer into configuration files for FPGAs of multichip RCSs. Then, these files are translated by Xilinx Vivado into bitstream configuration files (*.bit).

3.4. Sinis: Transformation of the Minimum Cadr Structure When the RCS Hardware Resource is Insufficient

In some cases, the available RCS hardware resource may be insufficient for hardware implementation even of the basic subgraph. Previously, the only way to organize calculations in this case was a sequential implementation on a processor. Due to the methodology, presented in the previous section, it is possible to continuously reduce the parallelism of the cadr structure, using the performance reduction by devices and by capacity, and micro-cadrs – a new form of organization of calculations. The main difference between a micro-cadr (m-cadr) MCS_2^{Op} (Fig. 6c) and a cadr structure CS (Fig. 6a) is the combination of several operations in one computing device, which leads to increase of the data processing interval, but provides one and the same balanced data processing rate. There is no need to use additional memory to store intermediate results of the input data flow processing when the cadr structure CS is implemented structurally and procedurally as two reduced cadr structures $CS/2$ (Fig. 6b).

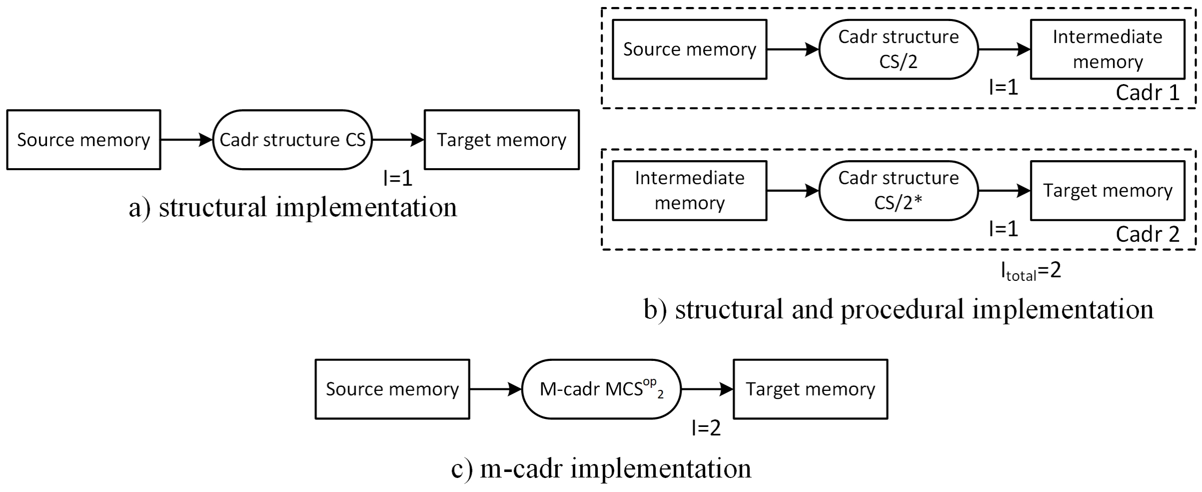


Figure 6. Methods of implementation of the cadr structure CS

Variants of generation of m-cadrs depend both on the problem area and on the solving task. The possible strategies for their creation are discussed in [14], where several m-cadrs are proposed for the task of digital signal processing. These transformations are performed by the Sinis tool, which, if necessary, is called by the Procrustes tool. Sinis obtains the results and returns them to Procrustes that uses them to find a rational version of the cadr structure according to the methodology from Section 3.3.

4. Results of Task Porting Performed by the "Theseus" Toolchain

The research of the efficiency of the Theseus high-level synthesis toolchain was carried out by porting a number of model tasks to various RCS architectures. The real performance rate of the ported solution for the Theseus toolchain, as well as for circuit engineers, was specified at least 0.6 from the peak one. Operability of the solutions obtained with the help of the Theseus toolchain was checked by running them on the corresponding RCS, and the characteristics of each task were compared with the results of FPGA designers. Transformation and porting was performed on a PC with Intel (R) Core (TM) i7-8750H @ 2.2 GHz processor, 16 GB of RAM, and

Windows 10 Pro operating system. Automatic porting was carried out for five model tasks: the symmetric-key block cipher DES, the MD5 and SHA-1 hash functions, and the Gaussian method and Jacobi method for 3-diagonal matrices. Each task was ported by FPGA designers and the toolchain to three different RCS hardware platforms (Fig. 7): Taygeta [3, 15], Tertius [15] and Tertius-3 [15].

The Taygeta RCS with the performance of 2.66 Tflops contains 4 20-layer printed circuit boards with double-sided mounting of 8V7-200 elements. Each circuit board contains 8 XC7VX485T-1FFG1761 FPGAs with 48.5 million equivalent gates, 16 SDRAM DDR2 distributed memory chips with a total volume of 2 GB, LVDS and Ethernet interfaces and other components.

The FPGA field of the desktop reconfigurable computers Tertius and Tertius-3 is not a separate board. It is integrated with a motherboard with an Intel Core I5 6300U processor. Tertius has the performance of 2.5 Tflops and contains 4 Xilinx Kintex UltraScale XCKU095 FPGAs with the capacity of 100 million equivalent logic gates each, interconnected in a ring by LVDS and GTY/GTH channels. Two dynamic memory modules with a capacity of 1 GB are connected to each FPGA, so the total memory size is 8 GB. Tertius-3 has the performance of 5.6 Tflops and contains twice as many chips of another FPGA type – 8 Virtex XCVU095-1FFVB1760C FPGAs.

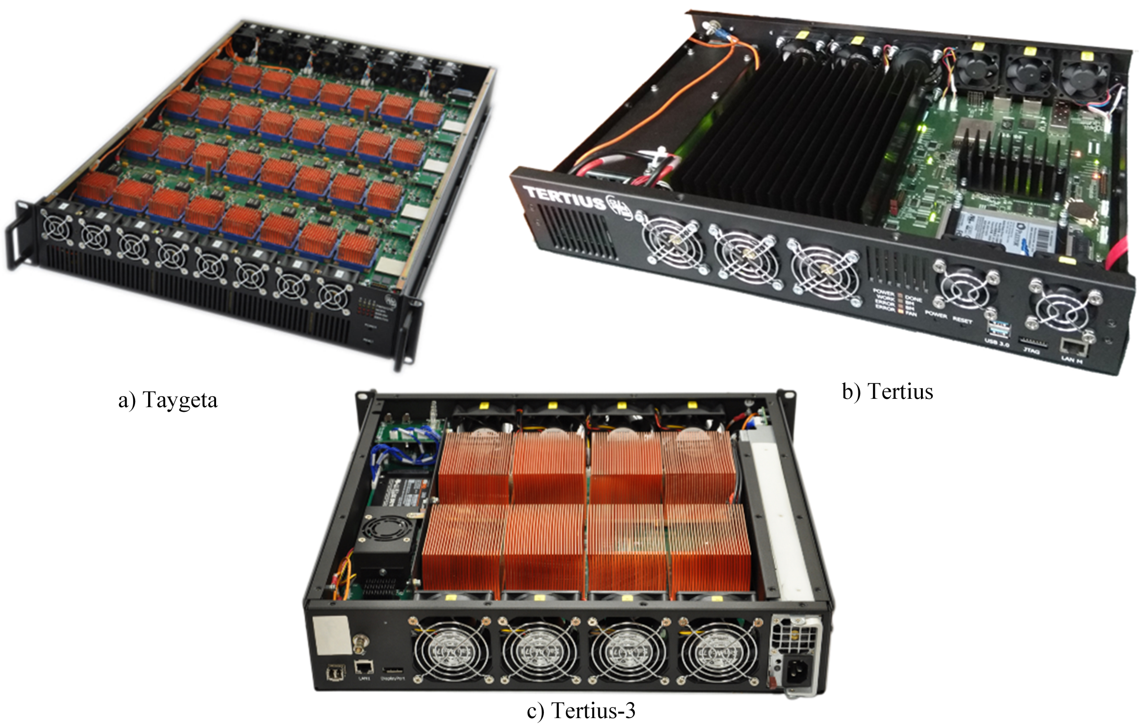


Figure 7. RCS hardware platforms for task porting

For each task, we measured the transformation time of its cadr structure, the task porting time, and the achieved performance. The cadr structure transformation time was defined as the working time of Procrustes. The porting time was considered as the sum of the cadr structure transformation time and the synthesis time of the FPGA bitstream configuration file, which depends on the logical capacity and the utilisation of the FPGA chip. With 90% utilisation of FPGA chips, it is at least 3 hours (10.800 seconds) for Taygeta and 7 hours (25.200 seconds) for Tertius and Tertius-3. According to experience of solving the same tasks by FPGA designers, the cadr structure transformation time for all hardware platforms was taken equal to two 8-hour working days or 57.600 seconds. The achieved real performance rate was defined as the ratio of the number of subgraphs in solutions obtained by the toolchain and FPGA designer (Tab. 1).

The transformation of the cadr structure by the Procrustes tool is performed significantly, by 1–3 decimal orders, faster than by FPGA designers, which is the expected effect of automation. Since the project synthesis time for FPGAs is significantly longer than the cadr structure transformation time, the total gain in the porting time of model tasks with the synthesis of bitstream configuration files for the selected RCS hardware platforms in Tab. 1 is equal to 3–6.3 times. The real performance rate achieved by the toolchain in porting model tasks does not drop below the specified level of 0.6 and varies slightly for different RCS hardware platforms, which is explained by the architectural features of different FPGA crystals.

Table 1. Results of porting model problems

Problems	DES	MD5	SHA-1	Jacobi	Gauss
RCS Taygeta					
Porting time in seconds	10836.62	10938.86	10841.66	13429.90	12316.38
Gain	6.31	6.25	6.31	5.09	5.55
Real performance rate	**0.63**	**0.63**	**0.86**	**0.86**	**0.86**
RCS Tertius					
Porting time in seconds	25238.58	25345.13	25241.09	25912.87	26989.25
Gain	3.28	3.27	3.28	3.20	3.07
Real performance rate	**0.65**	**0.67**	**1.00**	**0.85**	**0.80**
RCS Tertius-3					
Porting time in seconds	25239.02	25344.79	25241.76	25914.02	26991.12
Gain	3.28	3.27	3.28	3.20	3.07
Real performance rate	**0.65**	**0.67**	**1.0**	**0.85**	**0.80**

The results of porting the model tasks DES and SLAE solution with the help of the Gaussian method, performed by the toolchain, were compared with the solution obtained by the Vivado HLS compiler for the Taygeta RCS. The solution of the DES model task obtained by Vivado HLS with one pipelined IP-core, which was scaled by FPGA designer to the available hardware resource of the Taygeta RCS, while its real performance rate was 5 times lower than that of the solution obtained by the toolchain. The solution, obtained by Vivado HLS as a result of porting the Gauss model task, contains one iterative stage versus 720 stages when translating the task by the toolchain. Using manual code marking with *#pragma* directives in Vivado HLS, we could get a solution for two iterative steps of the Gaussian elimination algorithm. According to the comparison of the results of porting model tasks by the toolchain, FPGA designers and Vivado HLS, we claim that the proposed methodology for the cadr structure transformation gets advantages over Vivado HLS.

Conclusion

The “Theseus” high-level synthesis toolchain, described in the paper, provides scalable solutions for multichip reconfigurable computer systems unlike the academic (DWARV, BAMBU, LEGUP) and commercial (CatapultC, Vivado HLS, Vivado Vitis) HLS tools. Automatic transformation of the input sequential written in C program with no specialized code marking is performed by presenting the task in the form of a cadr structure, and by the original method of its porting to the available RCS hardware resource using formal methods of reduction of performance and hardware costs. The application of the developed methodology of cadr structure porting to the available hardware resource of the RCS significantly reduces the number of analyzed variants of parallel calculations and the porting time. Due to the use of the "Theseus" toolchain in porting a number of model tasks, it is possible to find rational solutions for multichip RCSs (with the effectiveness not less than 60% from the results of FPGA designers) for a significantly lower (in comparison with parallelizing compilers) number of transformations. Unlike well-known HLS compilers, the input C program is transformed automatically, without manual code marking or other user instructions. As a result, we get multichip configuration files with automatic synchronization of information and control signals.

Acknowledgements

The research is partially funded by the Ministry of Science and Higher Education of the Russian Federation as part of state assignments "Development of a Multi-Agent Resource Manager for a Heterogeneous Supercomputer Platform Using Machine Learning and Artificial Intelligence" (topic FSEG-2022-0001).

References

1. Antonov, A.S., Afanasyev, I.V., Voevodin, Vl.V.: High-performance computing platforms: current status and development trends. Num. Meth. Prog. 22(2), 135–177 (2021). https://doi.org/10.26089/NumMet.v22r210

2. Guzik, V.F., Kalyaev, I.A., Levin, I.I.: Reconfigurable computer systems. SFedU Publishing, Taganrog (2016). 472 p.

3. Levin, I., Dordopulo, A., Fedorov, A., Kalyaev, I.: Reconfigurable computer systems: from the first FPGAs towards liquid cooling systems. Supercomputing Frontiers and Innovations 3(1), 22–40 (2016). https://doi.org/10.14529/jsfi160102

4. Nane, R., Sima, V., Pilato, C. *et al.*: A Survey and Evaluation of FPGA High-Level Synthesis Tools. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35(10), 1591–1604 (2016). https://doi.org/10.1109/TCAD.2015.2513673

5. Numan, M.W., Phillips, B.J., Puddy, G.S., Falkner, K.: Towards Automatic High-Level Code Deployment on Reconfigurable Platforms: A Survey of High-Level Synthesis Tools and Toolchains. IEEE Access 8, 174692–174722 (2020). https://doi.org/10.1109/ACCESS.2020.3024098

6. Nane, R., Sima, V.-M., Olivier, B., *et al.*: DWARV 2.0: A CoSy-based C-to-VHDL Hardware Compiler. In: 22nd International Conference on Field Programmable Logic and Applications

(FPL), Oslo, Norway, August 29-31, 2012. pp. 619–622. IEEE (2012). https://doi.org/10.1109/FPL.2012.6339221

7. Pilato, C., Ferrandi, F.: Bambu: A Modular Framework for the High Level Synthesis of Memory-intensive Applications. In: 2013 23rd International Conference on Field programmable Logic and Applications, Porto, Portugal, September 2-4, 2013. pp. 1–4. IEEE (2013). https://doi.org/10.1109/FPL.2013.6645550

8. Canis, A., Choi, J., Aldham, M., *et al.*: LegUp: High-Level Synthesis for FPGA-based Processor/Accelerator Systems. In: Proceedings of the ACM/SIGDA 19th International Symposium on Field Programmable Gate Arrays, FPGA 2011, Monterey, California, USA, February 27 – March 1, 2011. pp. 33–36. ACM (2011). https://doi.org/10.1145/1950413.1950423

9. Make Slow Software Run Fast with Vivado HLS. https://www.xilinx.com/publications/xcellonline/run-fast-with-Vivado-HLS.pdf, accessed: 2023-03-23

10. Vitis Unified Software Platform Documentation. Application Acceleration Development. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug1393-vitis-application-acceleration.pdf (2019), accessed: 2023-03-23

11. Tarasov, I.: Designing for Xilinx FPGAs using high-level languages in Vivado HLS environment. Components and Technologies 12 (2013), https://kit-e.ru/fpga/vivado-hls/

12. Kolganov, A.S. An experience of applying the parallelization regions for the step-by-step parallelization of software packages using the SAPFOR system. Num. Meth. Prog. 21(66), 388–404 (2020). https://doi.org/10.26089/NumMet.v21r432

13. Voevodin, V.V., Voevodin, Vl.V.: Parallel computing. BHV-Petersburg, Saint-Petersburg (2002). 608 p.

14. Dordopulo, A.I., Levin, I.I.: Performance Reduction For Automatic Development of Parallel Applications For Reconfigurable Computer Systems. Supercomputing Frontiers and Innovations 7(2), 4–23 (2020). https://doi.org/10.14529/jsfi200201

15. Computational blocks of SRC of supercomputers and neurocomputers. http://superevm.ru/index.php?page=modern-developments, accessed: 2023-04-11

DOI: 10.14529/jsfi230203

The Semantic Model Features of the Statically Typed Language of Functional-dataflow Parallel Programming[*]

Alexander I. Legalov[1], *Nickolay K. Chuykin*[1]

The features of a statically typed functional-dataflow model of parallel computation and its mapping to the statically typed language of functional-dataflow parallel programming Smile are considered. To provide support for architecture-independent parallel programming, we used: a functional style, an implicit managing of calculations on data readiness, structured data objects that provide representation of various types of parallelism. A distinctive feature of the approach is the inclusion in the model of special asynchronous data objects that can generate events on partial filling. These data objects are stream and swarm. Each of these data objects has its own specifics to control by parallel calculations. A stream is used to process data of the same type that arrives sequentially and asynchronously at random intervals. A swarm is used to describe independent data of the same type or different types, on which it is possible to perform massive parallel operations. The use of streams and swarms in various situations as well as their mapping into each other and other program objects are shown. An analysis is made of the possibilities of transforming the formed language constructs into programming languages used in writing programs for modern parallel architectures.

Keywords: parallelism, parallel computation model, architecture-independent parallel programming, functional-dataflow parallel programming, transformation of parallel programs.

Introduction

The modern development of parallel programs is focused on the use of methods that take into account the specific architectures of target computing systems. This is due to the desire to improve the performance of parallel computing. Research in the field of architecture-independent parallel programming has not yet formed to the final practical solutions and is carried in the following areas:

- automatic or semi-automatic parallelization of sequential programs with their subsequent transformation to the target architecture [1];
- development of programs or algorithms that has unlimited parallelism, determined by the problem being solved, with the subsequent "compression" of this parallelism in accordance with the restrictions determined by the target architecture [2].

There is a semantic gap between the written program and the real parallel computing system (PCS) with using any of these approaches. There is a loss of efficiency and balance, when program is transformed into machine code, since the characteristics of the "architecture-independent" serial or parallel algorithm conflict with the organization of calculations in specific PVS. That is why unlimited parallelism is often manually "compressed" when fine-tuning the program to suit the features of the computer, that defines an approach opposite to parallelization of sequential programs. Manual transformation leads to a loss of efficiency in the process of developing parallel software and do not allow to write a program once and for different architectures. In this regard, the problem is actual of searching for models of parallel computing and building on their basis programming languages and tools that provide effective transformation of the parallelism of a once written program for various computing resources.

[*]The paper is recommended for publication by the Program Committee of the International Scientific Conference "Parallel computational technologies (PCT) 2023".

[1]Higher School of Economics, National Research University, Moscow, Russian Federation

One of the ways that determines a more efficient transformation of architecture-independent programs to programs for real architectures is the use of static data typing, which provides efficient compilation into a typeless representation at the level of the command system architecture [3]. This approach is widely used in both sequential and parallel programming. However for writing architecture-independent parallel programs only static typing is not enough. It is also necessary to take into account the features of the constructions that describe parallelism, because their dynamic characteristics can make it difficult to transform into machine code for existing parallel computers.

The article is organized as follows. Section 1 is devoted to the analysis of the factors that determine the architectural independence of parallel programs. Section 2 discusses the features of a statically typed functional-dataflow parallel computing model. In Section 3 we present the semantics of interpretation statements for various relationships between data and functions. The conclusion summarizes the results of the study and indicates directions for further work.

1. Determining Factors of the Architectural Independence of Parallel Programs

Support for architectural independence from real computing resources in the description of parallel processes is generally provided both by the peculiarities of the representation of data storage methods and by the use of appropriate strategies for controlling of calculations [4].

The independence of data storage from memory is supported by a functional paradigm focused on representing programs as interacting functions. In contrast to the imperative approach, the data memory is presented in an implicit form. Using recursions instead of iterations allows you to get rid of the reuse of variables at the level of describing algorithms. These solutions largely ensure the architectural independence of programs and are implemented in various functional programming languages (FPL) [5]. However, most of these languages have limitations for the implicit representation of parallelism. That is due to the peculiarities of data structure organization as lists with sequential access to their elements. The presence in the list of access only to the head and tail does not allow to organize parallel computing directly in many modern functional programming languges. Therefore, to present concurrency in programming languages, explicit control of computations is usually used, on the basis of which threads or processes are created [6]. This leads to directive impact on concurrency of the programmer and is a factor that makes it difficult to port programs to other parallel architectures.

Dataflow control strategies allow to describe parallelism implicitly. One of the first such computational models is the Dennis model [7]. It formed the basis of a number of specialized data flow processors with different architectures. There are languages that use dataflow control and are released for various architectures. For example: Sisal [8], Colamo [9], LuNA [10]. A number of these programming systems combine data flow management with a functional style. However, for many computational models it is problematic to talk about architectural independence, which is often associated with the orientation of programming languages and methods for their transformation to certain architectural solutions. Most of them have not fully developed the concept of unlimited parallelism. Also, dataflow control is often combined with the use of explicit management or with the need to manage limited resources.

Along with the functional approach and dataflow control, some problems linked with an architecture-independent representation of parallelism can be solved using special data structures that not only contain data, but also support their various parallel behavior. The differences in the behavior of these data structures determine the approaches to the different organization of parallelism. Encapsulation of the behavior of these data structures inside of specialized dynamically generated and synchronized objects allows you to remove from programs the explicit control of calculations, which is usually used in imperative programming languages, replacing it by interaction with functions and other data structures with dataflow control. The use of parallel recursion allows us to consider the program as a description of activities performed in unlimited computing resources, formed implicitly as needed. Such an approach at the programming language level allows using it as an architecture-independent language. At the same time, this imposes special requirements on the transformation of programs from such languages into architecture-dependent programs.

The use of special data structures was implemented in the Pifagor functional-dataflow parallel programming language, which is based on the functional-dataflow parallel computing model (FDPCM) [11, 12]. These structures are represented as data lists, parallel lists, delayed lists, asynchronous lists. Each type of list defines its own methods for grouping data and dataflow control. The disadvantages of the proposed constructions and the language, include the dynamic typing of atomic types, as well as the dynamic formation of lists in the calculations process. It does not allow an effective output representation to be formed during program compilation.

The need to use static typing was confirmed during the implementation of a number of projects on the transformation of functional-dataflow parallel programs into real architectures:

- when converting to FPGA topology [13];
- when transforming into a statically typed imperative programming language [14].

To obtain the required solutions, it was necessary to use additional descriptions into the intermediate representation generated by the Pifagor compiler that define the types of processed data, and to limit the semantics of lists representing parallel structures.

Thus, to solve the problem of efficient transformation of an architecture-independent parallel program into a program for a real target architecture, it is necessary to jointly use approaches that do not solve the problem separately:

1. dataflow control;
2. functional programming paradigm;
3. special data structures designed to represent different types of parallelism;
4. static data typing.

Their joint application will be reflected both in the parallel computing model and in the tools created on its basis. Orientation towards architectural independence leads to the formation of a domain-specific parallel computing model and a domain-specific architecture-independent parallel programming language with a specific set of data structures and their semantics.

2. Statically Typed Functionally-dataflow Parallel Computing Model

The application of the considered approaches makes it possible to form a statically typed functionally-dataflow parallel computing model (STFDPCM). Borrowing many of the ideas from the previously proposed FDPCM [11, 12], the STFDPCM is oriented towards the use of a static type system and fixed dimensions of data objects. This, in turn, leads to a change in the semantics of program-forming operators. The axioms of the model and its transformation algebra are also changed. They get oriented to a more efficient transformation at compile time. At the same time, the main characteristics of the model that determine the specifics of functional-dataflow parallel programming remain unchanged:

- calculations take place inside unlimited resources, which allows you to implicitly describe parallelism without resource conflicts;
- calculations are managed using dataflow control;
- the choice of operations and axioms that define the basic set of functions is focused on a visual textual representation of the information graph of the program during its subsequent description using a programming language;
- the computational model defines the general structure of a functional-dataflow parallel program without reference to operational semantics, which can be additionally defined, thereby defining the specifics of a particular language of functional-dataflow parallel programming.

The model is given by the triple:

$$\mathbf{M} = (\mathbf{G}, \mathbf{P}, \mathbf{S_0}),$$

where **G** is an acyclic oriented graph that defines the information structure of the program (its informational graph), **P** is a set of rules that determine the dynamics of the model's operation (the mechanism for generating markup), $\mathbf{S_0}$ – initial labeling of informational graph arcs, on which data has already been generated that determines the execution dynamics.

Informational graph of the program:

$$\mathbf{G} = (\mathbf{V}, \mathbf{A}),$$

where **V** is a set of vertices that define operators, and **A** is a set of arcs that define ways of information transfer from data source operators to receiver operators. Each source operator can be associated with one or more receiver operators. The receiver operator can have multiple inputs, each of which can be associated with the output of only one arc. The information transmitted along the arcs can be of any acceptable type, determined by the characteristics of the data objects used in the STFDPCM.

The vertices of the graph corresponding to the operators provide information transformations of data and their structuring in various ways. There are the following types of vertices:

- interpretation operators (interpreters);
- copy operators (denotations);
- operators for grouping to data structures (data objects);
- delaying computations operator (delay).

The use in model of static typing instead of dynamic typing imposes its own limitations, but, on the other hand, it provides additional opportunities for transforming functionally streaming parallel programs into programs for real architectures.

2.1. Interpreters

Interpreters are designed to perform functional transformations. Each such operator has two inputs, one of which receives a value perceived as a function of **F** (functional input), and the other (data input) receives a value that is an argument of **X**. The focus of model on compile-time analysis has led to two varieties of interpreters: single-argument operator and group-argument operator.

A single-argument interpretation operator, denoted in textual representation, as in FD-PCM [11], by "`:`" (postfix form) or "`^`" (prefix form), is designed to define ordinary functions that take an argument as a whole. The group-argument interpretation operator is used to set calculations on each element of a group data object, generating at the output a similar data object with elements whose type corresponds to the type of the result of the function being executed. Denoted by the double character "`::`" for postfix or "`^^`" for prefix forms, respectively.

Using different interpreters allows to unambiguously apply a function with the same name depending on the context. For example, the subtraction function "`-`" on the argument `(10,-3)`, perceived it as a vector consisting of two integers, generates the following results:

- $(\mathbf{10}, -\mathbf{3}) : - \Rightarrow \mathbf{13}$ – subtraction function of second number from first number;
- $(\mathbf{10}, -\mathbf{3}) :: - \Rightarrow (-\mathbf{10}, \mathbf{3})$ – sign change group function.

The division of the interpretation operator into single-argument and group-argument allows you to introduce a flexible set of additional functions for lists of various structures, while providing more diverse processing of asynchronously incoming data.

2.2. Copy Operator

Each vertex of the graph **G** admits the presence of several output arcs, along which the same value is transmitted to other vertices. This set of arcs can also be thought of as a single **copy operator** that transfers data from its single input to multiple outputs. In general, a copy operator can be combined with the preceding operator from which its output arc emerges. A chain of copy statements is also possible, which can be thought of as a single statement. In textual form, it is determined by naming the value that determines the output of the vertex, and further using the introduced designation at the required points of the informational graph. Both postfix naming of the propagated object in the form: "`value >> name`" and its prefix equivalent, which looks like: "`name << value`" are used. For example:

```
y << F^x;
(x,y):+ >> c;
```

The type of the denotation is the same as the type of the result of the preceding computations and is determined at compile time.

2.3. Data Objects

Data objects are designed to represent various options for grouping data into structures that have certain properties and behavior. It is them that define various options for representing parallelism and ways to determine the dataflow control during calculations. Data objects include:

- a constant operator (constant) that ensures the use of immutable values;
- synchronous grouping operator or **join** operator, designed to collect all incoming data together before their subsequent issuance as a result;
- operator of asynchronous grouping or **swarm**, performing ordering by input number, but not synchronizing data coming to different inputs;
- an asynchronous sequence operator or **stream**, that arranges data according to the time it arrived and produces as result the first arrived value.

Each kind of objects has its own behavior in the course of the formation of incoming data, exposing the result obtained to its output, transmitted along the arcs to the vertexes that receive information.

2.3.1. Constant

Constant operator or **constant** defines a node that stores a constant value and is always ready to execute. This operator has no input. The output is initially set to markup that defines the prescribed value. The set of constant operators of the informational graph form the internal initial markup of the computation model. In textual representation, a constant operator is given by a value of the corresponding type. The type of the constant is assumed to be known at compile time. Constants include data of various basic types, for example: integers, booleans, signals, atomic functions, as well as functions that are formed when writing a program. Functions are referred to as constants because their descriptions are fixed and are constructions that are immediately ready for execution. Constant examples:

- **10** – integer constant;
- **true** – boolean constant;
- **!** – signal constant (denoted by symbol "!");
- **+** – atomic function plus;
- **min** – the function name which developed for finding the minimum of two elements.

2.3.2. Join

Join has multiple inputs and one output. It provides structuring, ordering and synchronization of data coming from various sources along input arcs. The types of incoming elements must be known at compile time. The order of elements is determined by the numbers of inputs, each of which corresponds to a natural number in the range from $\mathbf{0}$ to $\mathbf{N-1}$, where $\mathbf{N}$ is the length of the generated data set. A connector signals ready when it receives all input. In text form, the operator is specified by delimiting the list elements with parentheses "(" and ")". For example:

```
(x0, x1, x2, x3)
```

The numbering of elements starts from zero and is set implicitly in accordance with their order from left to right. The element type of the join must be known at compile time and determines one of the possible interpretations: a vector or a tuple.

A join as **vector** is intended for grouping elements of the same type. This allows you to access its elements by index, as well as perform bulk operations on all elements.

The join as **tuple** also has multiple inputs and one output. It provides structuring, ordering and synchronization of heterogeneous data coming along arcs from various sources. The types of incoming data must be known at compile time. Elements are accessed by an index specified by a constant, which allows the type of the output value to be determined at compile time. In text form, similarly to a vector, it is specified by limiting the elements of the list with parentheses "(" and ")". Tuples can be used as arguments to functions, each of which, in its description, uniquely maps the element type to its location in the tuple.

2.3.3. Stream

The concept of **stream** extends the idea of the previously proposed asynchronous list [12]. The basic idea associated with asynchronous data arrival is preserved. However, all elements are assumed to be of the same type, which in turn cannot be a stream or a swarm. This is quite consistent with the concepts of universal statically typed languages. A stream can be considered as an object (Fig. 1), the main characteristics of which are:

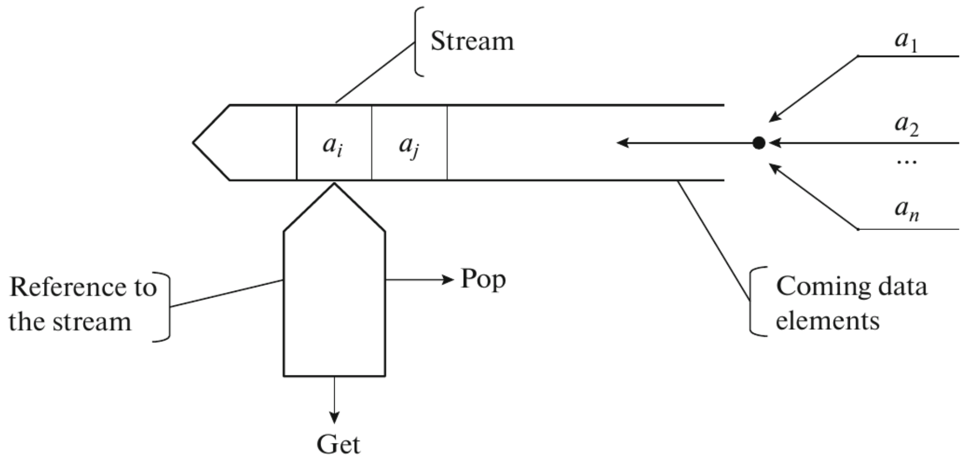


Figure 1. General scheme of a stream

- when at least one ready data element appears in the stream, it generates a signal informing about its readiness;
- the ready element can be read from the stream for processing;
- if, during processing of an element selected from the stream, new data items enter it, they can also be asynchronously selected from the stream in order of arrival and processed in parallel;
- parallel processed elements of the stream may arrive after processing in another stream, the type of which is determined by the type of the result of the function (in this case, the order of their arrival may differ from the initial one, depending on the processing time);
- we can check the stream for end of data in it and terminate the work if it is true.

In text form, grouping into a stream is specified by delimiting its elements with the symbols "`<[`" on the left and the closing square bracket "`]`" on the right. For example:

```
<[x0, x1, x2, x3]
```

The same type of stream elements is explained by the fact that if they appear randomly in time, it is impossible to determine the type of the current value at compile time.

2.3.4. Swarm

Swarm, unlike **join**, groups independent data. The arrival of each element in the swarm is accompanied by the issuance of a ready signal, informing the nodes of the informational graph about this event, receiving information from it. This allows you to quickly and asynchronously respond to changes in the state of the swarm.

In text form, grouping into a swarm is specified by limiting its elements with square brackets "`[`" and "`]`". For example:

```
[x0, x1, x2, x3]
```

Each element of the swarm is formed independently and is ready for execution when it appears. Like the connector, swarm elements can be of the same type, forming a vector, or of different types, forming a tuple.

A swarm, like a stream, allows you to process incoming data one element at a time (Fig. 2). This is possible due to the fact that the arrival of elements in the swarm can occur non-

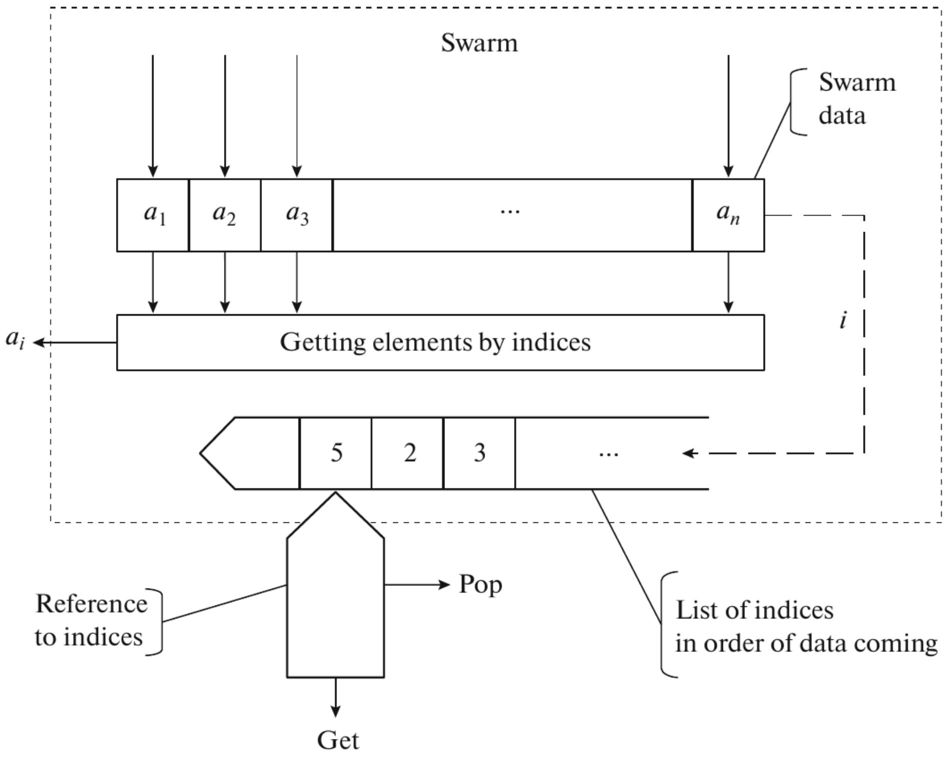


Figure 2. General scheme of a swarm and its reference

simultaneously and asynchronously. Therefore, the indexes of the elements can line up in the internal stream in the order in which they were received. Sequential selection of indices from this stream allows accessing swarm elements at the moment they arrive.

2.4. Delay

Delay operator or delay is specified by a vertex containing a valid informational subgraph that includes several input arcs and one output arc. The input arcs determine the arrival of arguments, and the output specifies the result issued from the subgraph. A specific feature of this grouping is that vertices bound by the delay operator cannot be executed, even if all arguments are present at their inputs. Their activation is possible only when the delay is removed (opening the contour), when the limited subgraph becomes part of the entire computed graph.

Initially, the delayed subgraph creates on its only output a constant markup, which is the image ("icon") of this subgraph. This markup propagates along the arcs of the graph from one operator to another, multiplying, entering various data objects and getting out of them until it arrives at one of the inputs of the interpretation operator. As soon as the delay operator becomes one of the arguments of the interpretation operator, instead of the "icon" the delayed subgraph is substituted with the input connections preserved. The contour of the delay operator encircling the subgraph is "removed" in this case, and the activated operators are executed. As a result, the resulting labeling is again formed on the output arc of the expanded subgraph, which is one of the arguments of the interpretation operator that expanded the delayed subgraph. This procedure is called delayed subgraph expansion.

In text form, the delay operator is specified by enclosing other operators with curly braces "{" and "}". For example:

```
{(a,b):+}
```

If it is necessary to form several independent arguments within the delay, then they are grouped into a swarm, which is initiated upon revelation:

```
{[x0, x1, x2, x3]}
```

The presence of a delay allows you to postpone the start of some calculations or not start them at all. It is necessary when organizing selective data processing. In addition, this operator, if necessary, can be used as brackets that change the priority of operator execution. To do this, it can be directly provided as one of the arguments to the interpretation operator.

3. Relationship between Grouping and Interpretation Operators

Calculations are formed when data arrives at interpretation operators. In this case, a calculation result is formed, the type of which depends both on the types of the argument and function, and on the type of the interpretation operation. Various combinations of these three components form the operational semantics of the computational model, and also determine options for equivalent transformations in accordance with the algebra of the model.

3.1. Using a Single-argument Interpretation Operator

A single-argument interpretation operator in most cases defines the traditional functional transformations of its arguments into a result. The following combinations of relationships between arguments are allowed:

- scalar – scalar;
- join – scalar;
- swarm – scalar;
- stream – scalar;
- scalar – join;
- join – join;
- swarm – join;
- stream – join;
- scalar – swarm;
- join – swarm;
- swarm – swarm;
- stream – swarm.

In these relations the first argument acts as the data to be processed, and the second defines the function that processes this data. Note that stream cannot act as a function. An argument is a scalar if it has a predefined base (atomic) type or is a constant of one of the base types.

3.1.1. Relation "scalar – scalar"

This relation is perceived by a single-argument interpretation operator as a traditional function of one argument. That is, the data being processed are constants or computed values of the underlying type. The generated result is determined by the semantics of the data interpreted as a function. The function can be either predefined or developed by the programmer. For example:

$$\mathbf{x} : - \equiv -\mathbf{x}$$
$$\mathbf{x} : \mathbf{sin} \equiv \mathbf{sin}(\mathbf{x})$$

3.1.2. Relation "join – scalar"

The relation is almost the same as executing a function from several arguments, the values of which are defined as elements of the join:

$$(\mathbf{x}, \mathbf{y}) : + \equiv +(\mathbf{x}, \mathbf{y}) \equiv \mathbf{x} + \mathbf{y}$$
$$(\mathbf{x}, \mathbf{y}) : \mathbf{min} \equiv \mathbf{min}(\mathbf{x}, \mathbf{y})$$

3.1.3. Relation "swarm – scalar"

The relationship allows to form an asynchronous flow of arguments to the function that displays them. It is assumed that when any element appears in the swarm, the interpretation operator will launch a function that processes the swarm, which will perform a partial calculation. That is, in this case, there is no preliminary synchronization of arguments before calling the function. The situation is similar in many ways to using of the **inline-function**. It is assumed that such calculations are possible when input parameter of function is described as a swarm. Example:

$$[\mathbf{x}, \mathbf{y}] : \mathbf{min}$$

3.1.4. Relation "stream – scalar"

The relation determines the transfer to the input of a function the stream, the readiness of which is determined by the appearance of any first element. The analysis of data readiness for subsequent elements of the incoming stream is formed within the function that processes this stream.

It should also be noted that, in the general case, the number of incoming elements is not defined for streams. Therefore, along with the value generated in the stream, a ready flag is also generated, which is a boolean value. As a result, a join of the following format is formed at the output of the stream:

$$(\mathbf{value}, \mathbf{flag})$$

Therefore, before using the value at the beginning, it is necessary to check the truth of the flag, indicating that the required values from the stream are still coming. The last value that signals the end of data is the **false** flag. However, the data itself is no longer defined.

3.1.5. Relation "data – join"

The data can be: scalar, join, swarm, stream. The join as a functional argument specifies the simultaneous execution of all the functions over the processed argument. In this case, parallelism is implemented with many independent command streams over one data set. For any input data $\mathbf{X}$ and a list of functions $\mathbf{F} = (\mathbf{F_0}, \mathbf{F_1}, ..., \mathbf{F_{n-1}})$ during interpretation, an equivalent transformation into a set of parallel executable statements is performed:

$$\mathbf{X} : (\mathbf{F_0}, \mathbf{F_1}, ..., \mathbf{F_{n-1}}) \equiv (\mathbf{X} : \mathbf{F_0}, \mathbf{X} : \mathbf{F_1}, ..., \mathbf{X} : \mathbf{F_{n-1}})$$

The output join is formed as result. For example, the simultaneous execution of addition, subtraction, multiplication, and division operations on a single data argument can be described as follows:

```
    (x,y):(+, -, *, /) ⇒ ((x,y):+, (x,y):-, (x,y):*, (x,y):/)
```

3.1.6. Relation "data – swarm"

The options for using a swarm as a set of functions are similar in many ways to using a join in that role. The difference is that its elements can process the input argument without common synchronization. Therefore, the appearance of results of the interpretation operator that implements this relation can be arbitrary with the formation of a new swarm as result. The input data $\mathbf{X}$ and the list of functions $\mathbf{F} = [\mathbf{F_0}, \mathbf{F_1}, ..., \mathbf{F_{n-1}}]$ are transformed into a set of parallel operators:

$$\mathbf{X} : [\mathbf{F_0}, \mathbf{F_1}, ..., \mathbf{F_{n-1}}] \equiv [\mathbf{X} : \mathbf{F_0}, \mathbf{X} : \mathbf{F_1}, ..., \mathbf{X} : \mathbf{F_{n-1}}]$$

As result a swarm is formed. Performing addition, subtraction, multiplication, and division above the same data argument at parallel would look like this:

```
    (x,y):[+, -, *, /] ⇒ [(x,y):+, (x,y):-, (x,y):*, (x,y):/]
```

3.2. Using the Group-argument Interpretation Operator

The group-argument interpretation operator is focused on the processing of group data objects by one function. Its main task is the simultaneous processing of elements located in data objects. In this case, the final calculations are formed through single-argument interpretation operators, reduction to which is carried out by applying equivalent transformations. The following combinations of relations are allowed for the interpretation group operator:

- join – scalar;
- swarm – scalar;
- stream – scalar.

More complex combinations of various data objects lead to the formation of multidimensional structures and are not yet considered at the level of the computation model.

3.2.1. Relation "join – scalar"

In this relation, the join is considered as a vector of elements of the same type, each of which is applied to the same function, acting as a scalar. Let $\mathbf{X} \equiv (\mathbf{x_0}, \mathbf{x_1}, ..., \mathbf{x_{n-1}})$ be a join data object that defines the vector of processed data, $\mathbf{f}$ be a function. Then, taking into account the subsequent equivalent transformations, the group interpretation operator for this relation can be represented as follows:

$$\mathbf{X} :: \mathbf{f} \equiv (\mathbf{x_0}, \mathbf{x_1}, ..., \mathbf{x_{n-1}}) :: \mathbf{f} \equiv (\mathbf{x_0} : \mathbf{f}, \mathbf{x_1} : \mathbf{f}, ..., \mathbf{x_{n-1}} : \mathbf{f})$$

3.2.2. Relation "swarm – scalar"

This relationship is largely similar to the previous one. The difference lies in the fact that the execution of functions on individual elements begins immediately upon receipt of these elements. A swarm is also asynchronously formed as a result of the execution of functions.

The swarm is defined as follows: $\mathbf{X} \equiv [\mathbf{x_0}, \mathbf{x_1}, ..., \mathbf{x_{n-1}}]$. A group operation on a swarm using the $\mathbf{f}$ function is defined by the following equivalent transformation:

$$\mathbf{X} :: \mathbf{f} \equiv [\mathbf{x_0}, \mathbf{x_1}, ..., \mathbf{x_{n-1}}] :: \mathbf{f} \equiv [\mathbf{x_0} : \mathbf{f}, \mathbf{x_1} : \mathbf{f}, ..., \mathbf{x_{n-1}} : \mathbf{f}]$$

3.2.3. Relation "stream – scalar"

Group interpretation of the relationship is carried out by analogy with the previous ones. That is, the function is executed on each element of the stream as long as the stream receives elements from various sources. As a result, a new stream is formed at the output. It should be noted that the order of the results in the new stream may not match the order of the original data. This is because even when the same function is executed, the calculation time may differ for various reasons. The specificity of the stream is also that the number of processed elements may be unknown in advance, and the completion of the arrival of elements is determined automatically by reading the end-of-stream marker.

$$\mathbf{X} :: \mathbf{f} \equiv < [\mathbf{x_0}, \mathbf{x_1}, ..., \mathbf{x_{n-1}}] :: \mathbf{f} \equiv\ < [\mathbf{x_0} : \mathbf{f}, \mathbf{x_1} : \mathbf{f}, ..., \mathbf{x_{n-1}} : \mathbf{f}]$$

Conclusion

The key idea of the approach is the initial attempt to abandon traditional architectural solutions and consider parallel computing as an architecture-independent domain specific area. This idea largely coincides with the views of J. Backus on whether programming can be liberated from von Neuman style [15]. The main difference from the previously presented works [2, 11–14] on the functional-dataflow model of parallel computing is the explicit separation of data objects from its program-forming operators and the adaptation of their semantics to a more complete analysis at the compilation stage. This should ensure efficient transformation into programs for various architectures of real parallel computing systems.

The presented base features of the statically typed model of functional-dataflow parallel computing formed the basis of the developed statically typed language of functionally-dataflow parallel programming Smile [16]. The proposed constructions make it possible to describe not only static data structures, but also to form various options for the dynamics of their behavior, which provides various additional possibilities for representing parallelism in programs. Options for representing dynamically changing parallelism appear, what depends on the relationship between the intensity of data arrival and the time of their processing. At the same time, along with data parallelism, their dynamic pipelining is possible through the use of streams [17]. Accounting for the use of static typing at the level of the computational model makes it possible to build a more efficient basis for subsequent transformations of architecture-independent parallel programs.

References

1. Steinberg, B.Ya., Steinberg, O.B.: Program transformations as the base for optimizing parallelizing compilers. Program Systems: Theory and Applications 12:1(48), 21–113 (2021).

https://doi.org/10.25209/2079-3316-2021-12-1-21-113 (in Russian)

2. Legalov, A.I., Vasilyev, V.S., Matkovskii, I.V., Ushakova, M.S.: A Toolkit for the Development of Data-Driven Functional Parallel Programmes. In: Sokolinsky, L., Zymbler, M. (eds) Parallel Computational Technologies. PCT 2018. Communications in Computer and Information Science, vol. 910, pp. 16–30. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99673-8_2

3. Pierce, B.C.: Types and Programming Languages. The MIT Press (2002)

4. Legalov, A.I.: On the control of computations in parallel systems and programming languages. Scientific Bulletin of NSTU 3(18), 63–72 (2004) (in Russian)

5. Okasaki, C.: Purely Functional Data Structures. Cambridge University Press (1998)

6. Charpentier, M.: Functional and Concurrent Programming: Core Concepts and Features. Addison-Wesley (2022). 528 p.

7. Dennis, J.B., Fosseen, J.B., Linderman, J.P.: Data flow schemas. In: Ershov, A., Nepomniaschy, V. (eds) International Symposium on Theoretical Programming. Lecture Notes in Computer Science, vol. 5. Springer (1974). https://doi.org/10.1007/3-540-06720-5_15

8. Kasyanov, V.: Sisal 3.2: functional language for scientific parallel programming. Enterp. Inf. Syst. 7(2), 227–236 (2013). https://doi.org/10.1080/17517575.2012.744854

9. Levin, I., Dordopulo, A., Gudkov, V., *et al.*: Software Development Tools for FPGA-Based Reconfigurable Systems Programming. In: Voevodin, Vl., Sobolev, S. (eds) Supercomputing. RuSCDays 2019. Communications in Computer and Information Science, vol. 1129. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36592-9_51

10. Malyshkin, V., Perepelkin, V.: The PIC Implementation in LuNA System of Fragmented Programming. The Journal of Supercomputing 69(1), 89–97 (2014). https://doi.org/10.1007/s11227-014-1216-8

11. Legalov, A.I.: Functional language for creating architecturally independent parallel programs. Vychislit. Tekhnol. 10(1), 71–89 (2005) (in Russian)

12. Legalov, A.I., Redkin, A.V., Matkovsky, I.V.: Functional-dataflow parallel programming with asynchronously incoming data. In: Parallel computing technologies (PaVT'2009): Proceedings of the International Scientific Conference, Nizhny Novgorod, March 30 – April 3, 2009, pp. 573–578. Chelyabinsk, Ed. SUSU (2009) (in Russian)

13. Romanova, D.S., Nepomnyashchiy, O.V., Ryzhenko, I.N., *et al.*: Parallelism reduction method in the high-level VLSI synthesis implementation. Trudy ISP RAN/Proc. ISP RAS 34(1), 59–72 (2022). https://doi.org/10.15514/ISPRAS-2022-34(1)-5

14. Vasilev, V.S., Legalov, A.I., Zykov, S.V.: Transformation of Functional Dataflow Parallel Programs into Imperative Programs / Automatic Control and Computer Sciences 56(7), 815–827 (2021). https://doi.org/10.3103/S0146411622070239

15. Backus, J.: Can programming be liberated from von Neuman style? A functional stile and its algebra of programs. CACM 21(8), 613–641 (1978). https://doi.org/10.1145/359576.359579

16. Legalov, A.I., Legalov, I.A., Matkovskii, I.V.: Specifics of Semantics of a Statically Typed Language of Functional and Dataflow Parallel Programming - Scientific Services & Internet 2019. In: CEUR Workshop Proceedings, vol. 2543, pp. 274–284. https://doi.org/10.20948/abrau-2019-08

17. Legalov, A.I., Matkovskii. I.V., Ushakova, M.S., Romanova, D.S.: Dynamically Changing Parallelism with Asynchronous Sequential Data Flows. Automatic Control and Computer Sciences 55(7), 636–646 (2021). https://doi.org/10.3103/S0146411621070105

DOI: 10.14529/jsfi230204

Multipurpose Reconfigurable Supercomputer with Immersion Cooling

Ilya I. Levin[1], *Aleksandr M. Fedorov*[1], *Yuriy I. Doronchenko*[1], *Maksim K. Raskladkin*[1]

In the paper we consider a promising universal reconfigurable supercomputer, the computing nodes of which are reconfigurable computing device Arcturus. It was developed at the Supercomputers and Neurocomputers Research Center (Taganrog) and based on modern Xilinx FPGAs of UltraScale+ family of HBM-series. The purpose is to achieve the highest computational layout density, to ensure balanced power supply and cooling, as well as the implementation of powerful data exchange configuration. The supercomputer can have up to 1.5 thousand FPGAs of a single computing field. It has extensive information exchange capabilities between FPGAs within the device and between devices to solve tightly coupled problems. Differential lines with multi-gigabit transceivers connected to them are used as the main connections between FPGAs. It provides exchange at the velocity up to 25 Gbit/s. Information interaction between the RCD is performed through optical channels with the capacity up to 4.5 Tbit/s. Immersion technology is used for cooling components of computing system. It provides the removal of the total heat output up to 20 kW. The developed power supply configuration is based on the input constant voltage of 380 V and provides stable power supply to the components. Owning to the implementation of time-consuming algorithms for various scientific and technical problems with the high real performance, it is possible to widespread use of the Arcturus supercomputer. Scaling of computing nodes will allow designing an entire computing circuit of a supercomputer with the performance up to several tens of Petaflops.

Keywords: supercomputers, reconfigurable computing systems, computing performance, immersion cooling systems, computing energy efficiency, computational density, highly connected problems.

Introduction

The solution of modern time-consuming scientific and technical problems of various subject areas requires the design of computing systems not only with overall high chip performance, but also with a variety of high-capacity channels for direct information exchange between computing nodes. Solving such tightly coupled problems using calculators based on universal processors is extremely inefficient. This is because traditional supercomputer architectures do not correspond to the structure of implemented algorithms. Therefore, they provide the high computing performance for very small class of problems. In addition, the processor performance growth has now stopped due to the approach to the technological limit of production.

In this regard, it seems most appropriate to use reconfigurable computing systems (RCS) to adapt their architecture for solving problem structure [1, 2]. The RCS concept is the maximum structural implementation of the task information graph on the supercomputer hardware resource. Such implementation where the algorithm operation is assigned to each operating vertex of the graph, arcs between vertices determine the data transfer between operations, and information vertices of the graph are implemented by corresponding memory channels. The structural implementation of calculations ensures the organization of pipelined, fastest data processing [3].

Field-programmable gate arrays (FPGAs) as the element base of reconfigurable systems, in contrast to the element base of multiprocessor computing systems, continues to support the

[1]Supercomputers and Neurocomputers Research Center, Taganrog, Russia

RCS development by maintaining the growth rate of hardware resource and operating clock frequency. The reconfiguration possibility allows the user to create virtual specialized computers within the same architecture, adequate to the structure of each solved application problem. However, the main problem is the organization of the commutation system, which should provide a wide capacity, but not require large hardware costs and time delays. Existing reconfigurable supercomputers are characterized by powerful information exchanges only within the basic functionally complete nodes – calculating blades [4]. The bandwidth between the boards of computing modules is significantly lower. This limits the possibilities of efficient parallelization of calculations in application programs.

Currently, the employees of the Supercomputers and Neurocomputers Research Center are designing the experimental sample of a prospective universal reconfigurable computing device (RCD) Arcturus based on FPGA, which is a development of the commercially produced RCB Neckar [4]. A number of breakthrough technical solutions have been implemented in the new computer. Owning to them, it is possible to implement large graphs of complex problems in an entire computing circuit and perform calculations without interruptions due to the powerful information exchange system. At the same time, it has provided the necessary level of power supply and cooling.

The current level of energy consumption of chips does not allow obtaining effective solutions using the air cooling [5]. As a rule, the combined types of cooling, mainly liquid, are used at the development of new computing complexes. The most effective approach is the immersion technology with direct immersion of electronic components in a coolant [6]. It began to actively develop and received its implementation in computer technology samples. This technology ensures the absence of complex structural elements. This determines the highest layout density, as well as the use of low-cost coolants and guarantees the absence of critical leaks. However, constructs designed by various manufacturers [7, 8] for liquid cooling have a very low layout density of computing elements. As a rule, they are intended only for devices based on CPU or GPU. An original immersion cooling system has been developed.

The article is organized as follows. Section 1 presents the architecture of supercomputer, allowing increasing the performance efficiency at solving widespread tasks. In Section 2, the RCD Arcturus structure and its design features are described. Section 3 describes the organization of information interaction between RCB, as well as the features of the use of multi-gigabit transceiver technology implemented in them. Section 4 is devoted to the RCD Arcturus, which provides to problem solutions with special memory requirements. Section 5 describes the computing resources of the developed RCB. Section 6 is devoted to the description of the original motherboard, designed and manufactured for loading FPGAs and controlling the computing process in the RCB. Section 7 describes the design components of the RCB cooling system and the features of implemented immersion liquid cooling system. In Section 8, the real performance evaluation of the RCD Arcturus was performed using two floating-point tasks. In conclusion, the achieved results of the development of the promising RCD Arcturus are listed.

1. Supercomputer Architecture

For solving modern problems, it is necessary not only to have a large number of computing chips and memory, but also rapid data exchange between components of the computing system. It is known that the real performance is rapidly falling in tightly coupled problems due to the significant amount of transfers. For example, this class of problems includes the currently widespread tasks of machine learning, simulation of complex physical processes and technological devices, problems of the Earth remote sensing and digital signal processing. To increase the efficiency of solving such problems, a supercomputer architecture is proposed to organize a commutation system with up to 1.5 thousand chips combined in a single computing circuit (Fig. 1). An architecture feature is the direct information communication between the chips in the horizontal and vertical directions, as well as ring connections in each horizontal layer.

The implementation of this architecture on a single printed circuit board (PCB) is not possible. The overall dimensions are too large. The components' installation is almost impossible. The cost of one board is prohibitively high; the maintainability is low. It is necessary to decompose into separate modules for implementation each row of chips on a separate PCB. The set of blades obtained in this way are placed vertically and ensure the interaction using the cross-board (Fig. 2). The horizontal placement of boards is impractical, as this will increase the depth, and lead to the impossibility of chips' cooling.

Besides the computing blades, the power blades and control module blade must be provided for power supply, monitoring and control. The power module can provide the power to several calculating blades (for example, four). Thus, it is possible to form an element of a regular structure – a processing unit (Fig. 2). Processing units are scaled in a single construct – a computing device. The number of such units in the computing device can be equaled to four. There can be one control module blade.

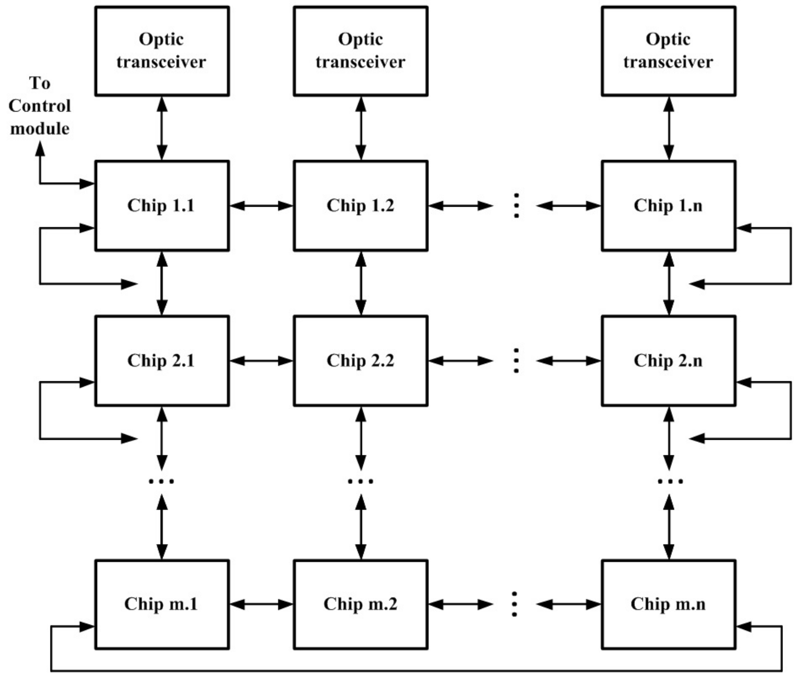


Figure 1. Supercomputer architecture

The decomposition of the proposed architecture leads to the rupture of vertical connections. The connections can be restored using the high-speed optical channels. Using the optical transceivers, a set of computing devices is combined with each other into a computing rack (Fig. 3). Therefore, the supercomputer can consist of several computing racks, in which computing devices jointly solve a problem.

The architecture shown in Fig. 1 corresponds to the "tor" architecture that does not contain vertical circuit information connections. Their implementation in general is impossible. Since the

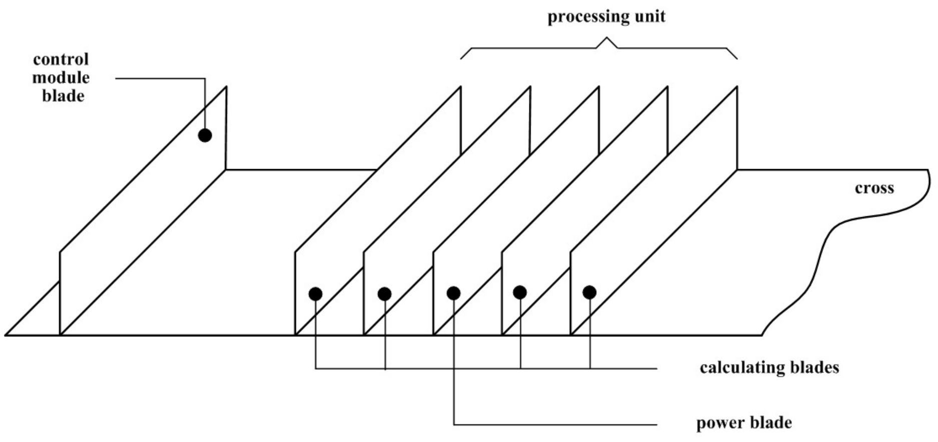


Figure 2. Blades placement

supercomputer is based on the modular expandability principle [9], and the number of processing units and computing devices is not known in each case.

At implementation of 6 chips on calculating blades, 4 processing units in the computing device will provide 96 chips. In the case of 16 computing devices in one computing rack, there will be more than 1.5 thousand chips.

This architecture can be used with ASIC, eASIC chips. The use of FPGA is most effective. Reconfigurable FPGAs allow solving many problems, which ensures the supercomputer versatility. In this case, the processing units combined in one construct called the RCD.

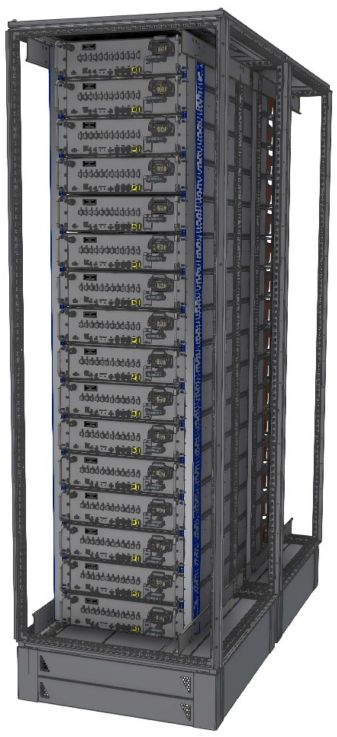


Figure 3. Computing rack

To solve time-consuming problems on the Arcturus supercomputer, it is necessary to use the paradigm of structural organization of calculations with parallelization by iterations and layers [10]. This approach provides the linear increasing of system performance at the linear increasing in hardware resource. This is not available to any modern computing system, based on processors and graphics accelerators. This is because the number of information exchanges is often comparable to the number of performed operations in time-consuming problems. Accordingly, the data exchange time between computing nodes has a greater impact on the problem solution velocity with such connectivity in traditional systems. In some cases, when the number of information exchanges in problems is close to the number of performed operations, the data parallelization schemes used in traditional systems are critical to the RAM, the access speed and the data transfer rate in interfaces between computing nodes. There is a "bottleneck problem", and the data exchange time turns out to be longer than the calculation time. Therefore, the system performance is a significant decrease at increasing the number of computing nodes.

In contrast to traditional approaches, the communication is provided by a spatial commutation matrix at the structural organization of calculations with parallelization by iterations and layers between the functional system nodes. The problem of memory access speed is solved by using distributed memory technology and the independent multi-channel access.

The used approach has been repeatedly tested and has proven itself well at solving many time-consuming, tightly coupled problems, such as problems in the field of bioinformatics, artificial intelligence, molecular modeling, simulation of complex physical processes, bit processing, geophysics, the Earth remote sensing and many others.

2. RCD Arcturus Structure

The supercomputer is based on RCD. The main structure features of the developed RCD Arcturus were considered. The RCD Arcturus design has the standard 3U 19" dimension and contains 16 calculating blades (CBs) vertically arranged on a cross-board with 6 Xilinx UltraScale+ XCVU37P FPGAs in each. The control module blade based on the Intel Skylake processor is used for general management, configuration and control of computing process.

The power blade DC/DC has been developed for power supply of the processing unit (four calculating blades). It converted performing conversions from the input power supply of 380 V DC to 12 V. The maximum power consumption of the RCD Arcturus is 20 kW.

These electronic blades are placed on a single cross-board (Fig. 4), which provides intermodule information and control interaction.

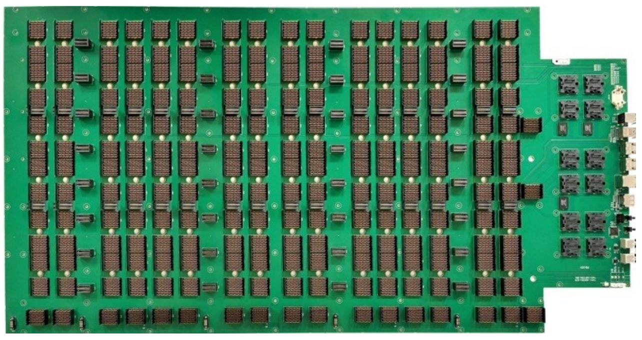


Figure 4. Cross-board

Cooling of the loaded electronic RCD components is provided by the immersion cooling system based on a dielectric coolant [11, 12]. The coolant has high electrical strength and thermal conductivity, as well as the highest possible heat capacity at low viscosity.

The construct consists of the main computing section with blades, filled by the coolant, and an additional dry section in front, which located user interfaces, connectors and optical modules. The heat exchanger for heat energy transferring from the first (coolant) to the second (water) cooling circuit is located behind the main section. The unique cross-board provides the high-frequency data transfer between blades. The components soldering is performed on an automatic line. The PCB dimensions are large and equal 845×436 mm. Therefore, the assembly of components and pressing of connectors is difficult at soldering paste application technology and pressing. The cross-board provides the sealed transition between the environments (coolant – air). The external dry-area of the cross-board implements the display output, control buttons, as well as interfaces for connecting a keyboard, mouse, display, Ethernet and optical information channels used for interdevice information exchange.

The 3D model of the RCD Arcturus is shown in Fig. 5. The RCD Arcturus with the open lid is shown in Fig. 6.

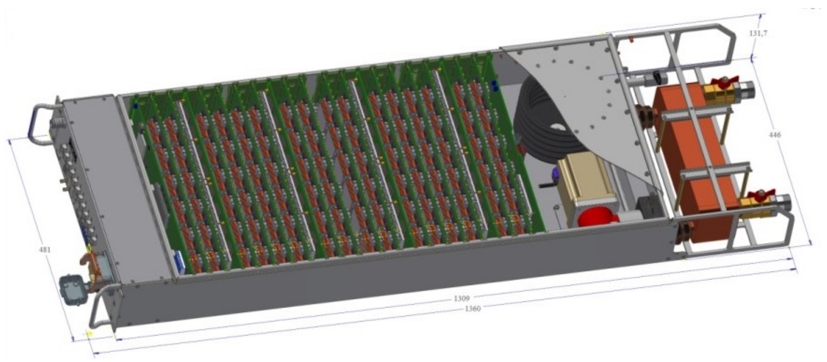


Figure 5. 3D model of the RCD Arcturus

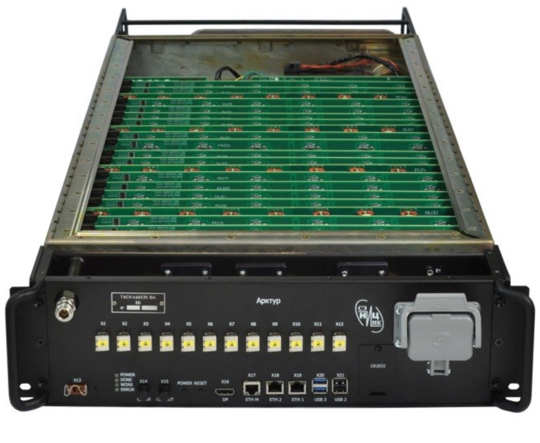


Figure 6. RCD Arcturus

3. Data Exchange Configuration

The most important feature of the promising reconfigurable supercomputer is the support of unique computing power by the extensive capabilities of data exchange configuration. It represents multiple communication channels between FPGAs within the CB, as well as between FPGAs of neighboring CBs [13].The structure of information links has been clarified at operating with the CB topology and designing technical solutions. Communication between FPGAs is provided via differential lines using multi-gigabit transceivers (MGT), integrated into the FPGA. Auxiliary connections are differential LVDS lines connected to the FPGA HP-banks (Fig. 7). 24 differential lines with data transfer rates up to 24 Gbit/s within the CB and 24 differential lines with data transfer rates up to 16 Gbit/s between the CBs are implemented between the pair of FPGAs. The total capacity of CB communication channels is 15.6 Tbit/s, including between CBs is 9 Tbit/s. Therefore, the RCD implements a universal orthogonal high-speed information communication system between computing FPGAs (Fig. 8).

At creation the computing complexes, the information interaction between RCD is performed through optical channels. Amphenol multichannel optical transceivers are installed on the cross-board and connected to the first calculating blade via the split connection, providing the capacity up to 4.5 Tbit/s (Fig. 8, 9).

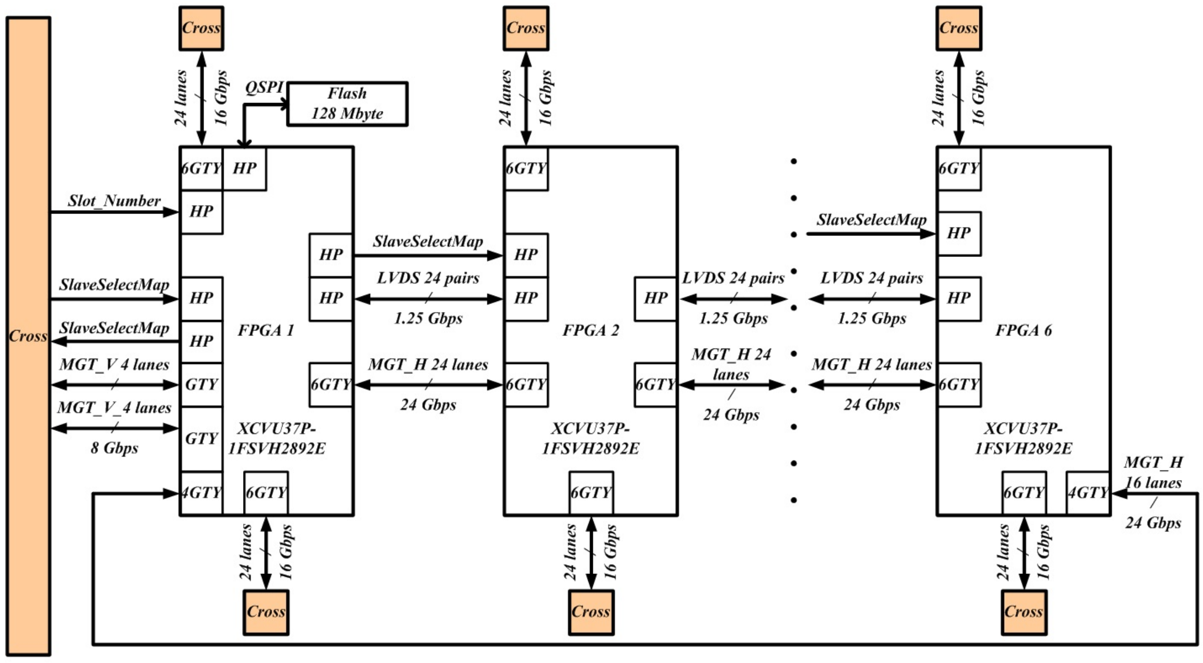


Figure 7. CB communication structure

Separately note the application features of the multi-gigabit transceiver technology. The part of the XCVU37P FPGA architecture is 96 high-speed (up to 25.785 Gbit/s) transceivers. For high data transfer rate, it is necessary to use special topological solutions at PCB designing: the limit on the line length, on the amount and various versions of transition holes, rounding corners, the use of a special PCB material that provides a low dielectric loss coefficient, etc.

However, the use of only topological solutions is not enough for high data transfer rates close to the maximum possible. The Xilinx FPGA manufacturer provides a special procedure for transceiver configuration to improve the transmitted data integrity. This setup of communication lines is a very difficult operation. It includes the searching procedure of transmitter optimal parameters and the receiver operating modes, as well as the margin estimation using an eye diagram and comparing the obtained value with a test mask. The setup procedure must be repeated for each high-speed line. The complete search for one communication line of all possible combinations of receiver and transmitter parameter values is about 100 thousand options. Obviously, manual configuration requires an unacceptably long time.

An original algorithm of progressive setup by the sequential approximation method have been developed by specialists of the Supercomputers and Neurocomputers Research Center.

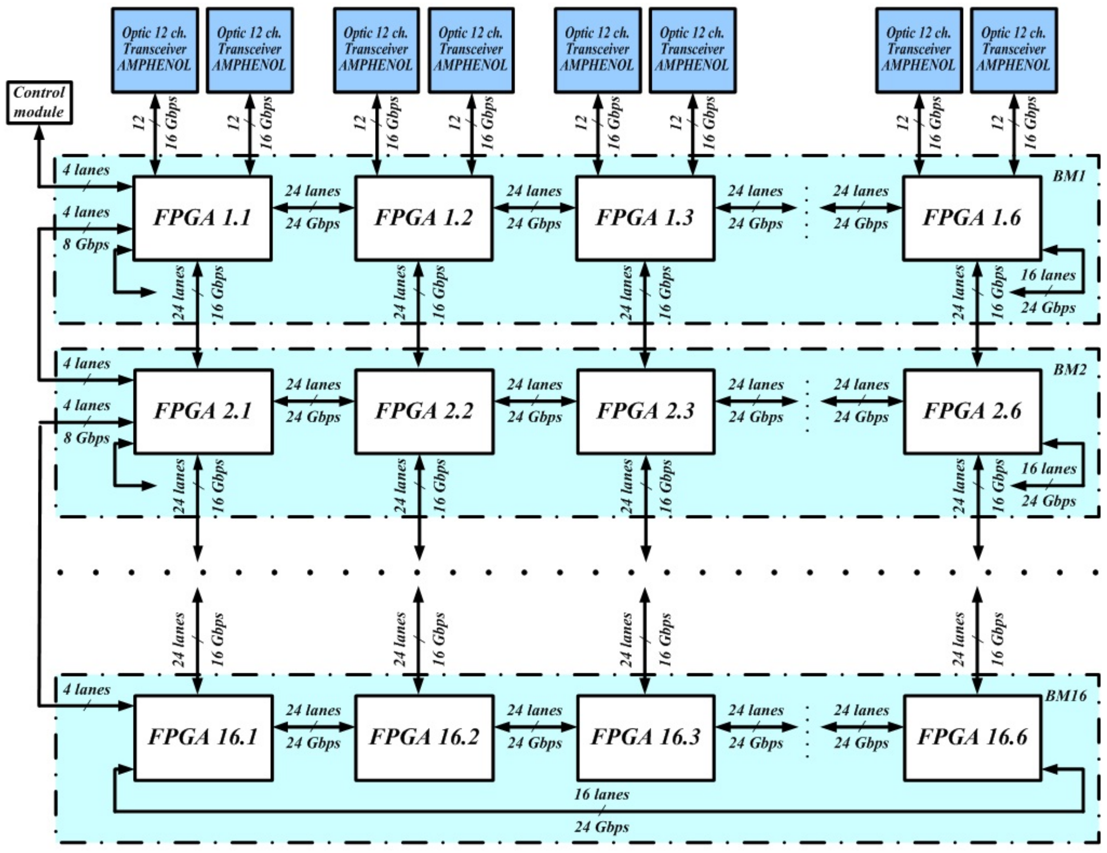


Figure 8. RCD communication structure

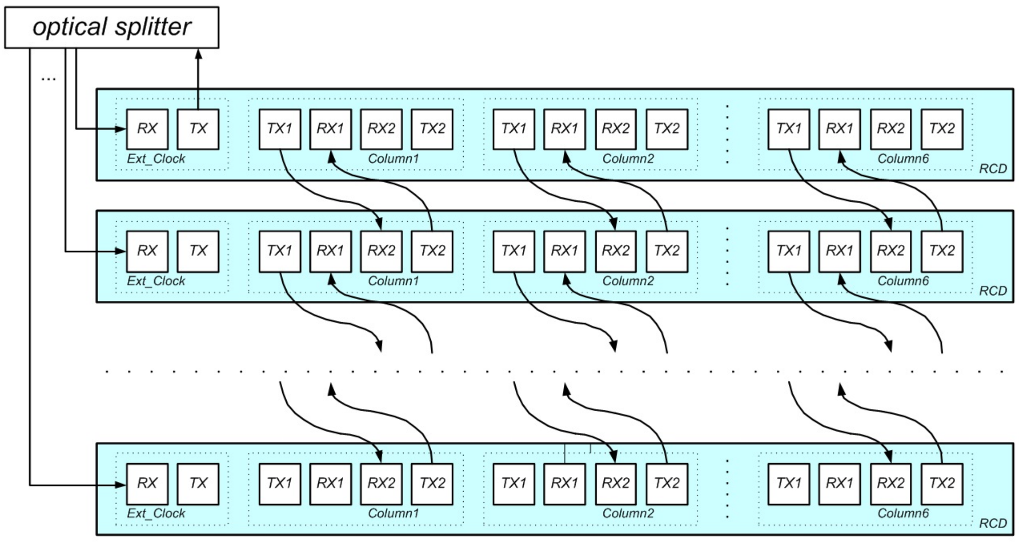


Figure 9. RCD interaction

Depending on the number of combinations, it is possible to reduce in 16 times for testing by iteratively searching the best set of settings, with the gradual decreasing in the range of considered parameters.

The algorithm consists of three iterations. At the first iteration, verification combinations are formed by sampling the transceivers settings every five possible values. Reference values are selected after receiving and measuring the eye diagrams of each combination – a set of settings gives the best result. At the next iteration, the parameters are sampled in steps of 3;

the search range around the selected reference values is narrowing. The following combinations are formed, the verification and adjustment of which gives a new set of reference values. The last iteration is an exact setup with the selection of each parameter from the required neighborhood of reference values. As a result, the number of checking combinations is significantly reduced using the progressive tuning method.

The progressive setup algorithm, as well as the developed functions for quality evaluation the of the eye diagram, logging the setup process and creating reports on the obtained results are implemented as a set of Tcl scripts, executed in the Xilinx Vivado.

The use of developed scenarios makes it possible to implement a fully automatic process of transceiver individual configuration. It does not require the participation of a highly qualified specialist and ensures the highest possible quality and transmission speed for each communication channel between FPGAs.

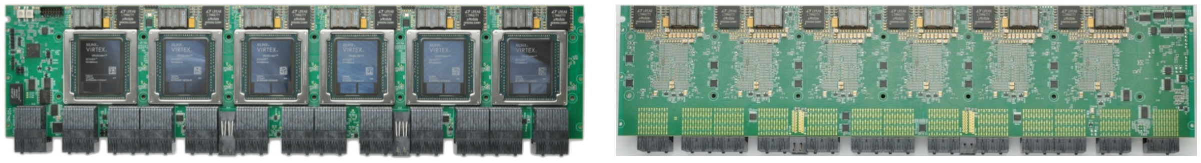
4. Calculating Blade

The RCD Arcturus should provide the problem solutions with special memory requirements. A typical technical solution used earlier was the placement of static or dynamic memory chips connected with computing FPGAs on the calculating blade [4]. Xilinx, as part of the UltraScale+ family, has released a special HBM line (High Bandwidth Memory). It includes the FPGA with a new hardware resource within the same package – the "HBM2" as integrated DDR. The HBM2 memory has a capacity up to 16 GB and can provide multi-channel access. This technology will significantly expand the possibilities for rational memory use in applications for solving the learning problems and mathematical physics.

The XCVU37P chip, used in the RCD calculating blade, has 8 GB HBM with the peak capacity up to 460 GB/s. For connection to the HBM controller, 256-bit API interfaces are used, the total number of which is 32. Each interface can either operate with its own dedicated 256 MB address space, or access any HBM address via the special switch (32×32 AXI crossbar switch). Experimental researches have shown the exchange possibility at the 12 GB/s speed on each channel.

The dimensions of the UltraScale+ FPGA case are 5 mm larger than the dimensions of the UltraScale FPGA case in previous RCD, for example, the RCD Skat [14], the RCD Neckar [4]. The dimensions of the PCB calculating blade are limited. Therefore, it is impossible to place, as usual, 8 FPGAs. Here 6 FPGAs are implemented. To preserve the computational density of the product, the number of processing units has been increased from 3 to 4.

Currently, the configuration work of the pilot calculating blade is being completed. The photo of CB is shown in Fig. 10.



(a) top view (b) bottom view

Figure 10. Arcturus calculating blade

All 136 differential MGT communication lines between FPGAs have the capacity of 6.6 Tbit/s. 288 differential MGT lines for the cross-board communication have the capacity of 9 Tbit/s. They are configured and tested using the progressive setup algorithm.

Experimental researches have confirmed the possibility of achieving the required error-free data transfer rates between FPGAs in different directions (Tab. 1).

Table 1. Error-free data transfer rates

Direction	Data rate, Gbit/sec	Number of MGT lines	Test duration	Transmitted bits	Bit error rate
FPGA – FPGA	25.6	192	96	$1.7 \cdot 10^{18}$	$0.6 \cdot 10^{-18}$
FPGA – Impel – FPGA	16	288	24	$0.8 \cdot 10^{18}$	$1.2 \cdot 10^{-18}$
FPGA – Impel – Amphenol – Impel – FPGA	16	144	24	$0.2 \cdot 10^{18}$	$5.0 \cdot 10^{-18}$

5. Power Supply, Power Blade

The new UltraScale+ chips computing resource is approximately twice the FPGA resource of the UltraScale previous generation. According to research, the maximum consumption of one FPGA is 160–180 W taking into account the increase in energy efficiency. It has required a radical redesign of the secondary power blade configuration of the calculating blade.

The contradictory requirements for the power supply system include: the high power; the maximum efficiency to minimize parasitic heat generation; the minimal overall dimensions to accommodate the required number of FPGAs on the PCB of the calculating blade. There are no existing solutions that provide the required current and voltage and are placed in one chip of the power blade. An original multiphase power blade system has been developed. Several chips are combined to obtain the required power. The difficulties of synchronization during parallel activation, interaction of radio-electronic components, PCB topological tracing, correct allocation of power polygons, etc. have been overcome. It was necessary to define a certain balance in the PCB between the number of FPGA power layers and the number of signal circuit layers. In addition, the problems of uniform current distribution from each source near the FPGA and ensuring no more than 80A current flows on each individual source FPGA section were solved.

The RCD power configuration ensures the unit's operability in the input voltage range from 350–400 V DC. The power blade is equipped with the soft start system, which ensures a smooth increase in voltage at the converter input and eliminates commutation interference.

The power configuration is started stepwise. The control module blade (CMB) is first started at the device beginning. In this, the primary converter generates the intermediate voltage of 12 V. Further, under the control of a microcontroller (MC), the following cascades are powered: FPGA on CMB, INTEL, cooling system, soft start system, pump motor and others. Information interaction between microcontrollers is implemented via the CAN bus.

To provide the power to the CBs FPGA, the primary conversion is performed on separate boards power blades (PB). The CB power system consists of two identical nodes, each of which receives 11–12.5 V from its converter located on the PB. Each node provides the power to three FPGAs. The power blade is shown in Fig. 11.

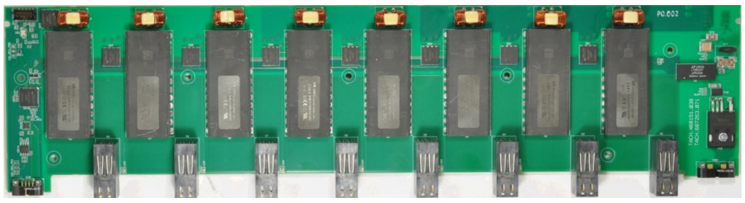


Figure 11. Power blade

The monitoring of voltages, currents and temperatures is the important node of the power blade configuration. A separate microcontroller is used to control and monitor the FPGA power. Data from all FPGA power microcontrollers are fed into the leading CB power microcontroller. Such distributed monitoring scheme makes it possible to increase the power configuration response rate to possible crashes and failures, as well as automatically disable only the failed FPGA, and not the entire calculating blade or RCD. At the same time, the communication system allows to redistribute the monitoring data flow bypassing the disabled FPGA. Note that in such cases it is possible to redistribute the user data flow bypassing the disabled FPGA, if this is available in the FPGA configuration created by the user.

6. Control Module Blade

An original motherboard (control module blade, CMB) was developed and manufactured to FPGA load and control the RCD computing process (Fig. 12). It is based on the Intel SkylakeCore I5-6300U processor. The CMB ensures the RCB functional completeness. It is realized as a separate board. An original basic input-output system (BIOS) has been developed for CMB. It allows using all capabilities of the Intel Skylake processor and external peripheral equipment. A radiator by the original design is used for heat removal from the processor.

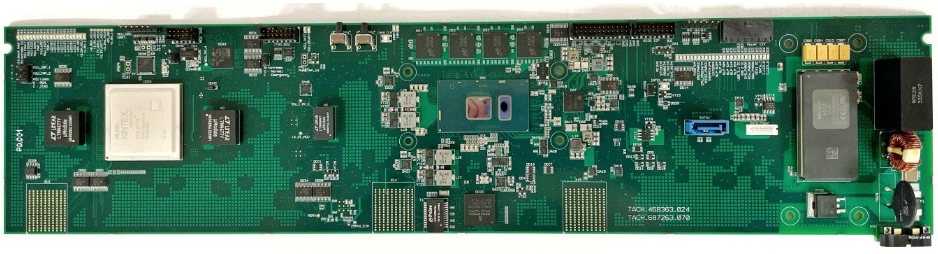


Figure 12. Control module blade

The main CMB nodes are the Intel Core i5-6300U processor and the Kintex Ultrascale FPGA. The FPGA CMB performs the controller function and provides interaction between the processor and the first FPGA in the first calculating blade. Communication between the CMB and CB is performed via the cross-board. The CMB also provides hardware monitoring of sensors, pump parameters and other data with using a microcontroller.

7. Cooling System

The RCD case is a sealed container with a coolant, in which electronic modules are immersed. The cooling system is functioning as follows. The pump circulates the coolant in the RCD along the following closed circuit: the heated coolant enters the lamellar heat exchanger and cools there. Then, the coolant re-enters the RCD case under the cross-board under the necessary pressure, with the help of special flow direction setters flows through heatsinks of cooled electronic components, which heats up itself and re-enters the heat exchanger (Fig. 13).

Figure 13. Heat exchanger

The heat exchanger is connected to the secondary cooling circuit through fittings. It was designed for coolant usually with water (Fig. 14).

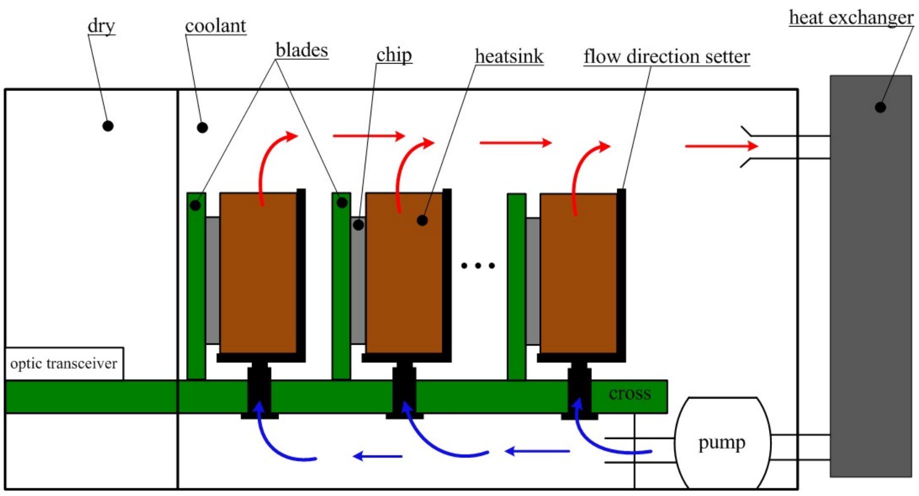


Figure 14. Cooling

The effectiveness of the immersion liquid cooling system is determined by the effectiveness of technical solutions for implementation each component: the used coolant, the construction and parameters of the used FPGA radiators, pumping equipment and heat exchangers.

The coolant must have the best electrical strength, high thermal conductivity, the highest possible heat capacity at low viscosity. The radiator must provide the maximum heat removal surface, the refrigerant circulation through the radiator, the refrigerant flow turbulence in the radiator, manufacturability. The used thermal interface must not degrade and be washed out by the refrigerant, have a consistently high thermal conductivity coefficient. The heat exchanger must provide the high heat transfer coefficient between the main and secondary cooling circuit. The pump with necessary performance is used for refrigerant circulation in the RCD volume.

This complex problem of RCD Arcturus cooling subsystem implementation provides the removal of thermal power up to 20 kW. It has been successfully solved and tested on the number of layouts. First, a new radiator construction was designed. Due to it, the effective heat exchange surface area was increased 2.5 times, compared to the previous design, at slight increasing of radiator dimensions. This became possible by reducing the thickness of ribs (thereby increasing the number of channels by 1.8 times) and increasing the number of grooves in the channels from 252 to 792; the grooves increase the heat removal area and form a special flow pseudo-turbulence (Fig. 15). Therefore, the heat removal capabilities have increased from 80 to 160 W.

The following important problems have been solved during the cooling system implementation:

– a submersible pump has been developed (Fig. 16) with the increased capacity up to 60 l/min, which makes it possible to abandon the cold air forced circulation in computing racks. It significantly simplifies the maintenance, and increases the reliability by reducing the number of open hydraulic connections;
– the cooling efficiency was improved using new heat exchanger (Fig. 12);
– a compensator of coolant volumetric expansion has been developed in the form of a corrugated rubber sleeve. The contact of the coolant with atmospheric air is completely excluded, and potential leaks are minimized;
– significant actions have been performed to change the RCD case and lid design. Due to it, the construction rigidity was increased, the product manufacturability was ensured, the assembly and product repair were simplified, the possibility of leaks was minimized, the construction weight was reduced.

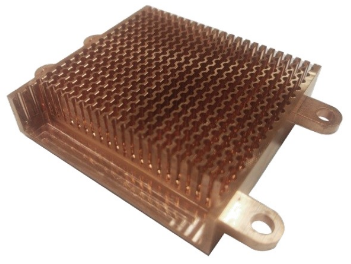


Figure 15. Heatsink

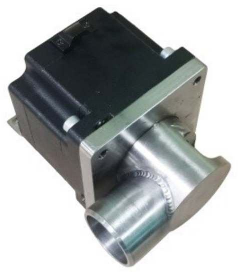


Figure 16. Submersible pump

Note that the distinctive feature of the used FPGA is its lidless-design (without the lid). The radiator installation technology directly on the FPGA crystal using the modern PSG graphite thermal interface by Panasonic with the thickness of 0.1 mm was proposed. Researches have confirmed the advantage of this technology in comparison with the radiator installation on the FPGA lid.

Currently, preparations are underway to test the developed cooling system on the manufactured experimental RCD sample. According to preliminary estimates, the thermal resistance of the “FPGA – coolant” transition is 0.27 C/W, then the temperature drop of the “FPGA – coolant” will be 43C at the average load of 160 W. When the water temperature in the second cooling circuit is 16C, the temperature on the FPGA will be 59C.

Therefore, the developed cooling system makes it possible to divert up to 20 kW of thermal power in the 3U construction. The RCD curb weight is about 150 kg. For comparison, one of the most effective modern solutions with the internal closed loop of liquid cooling is the specialized for sparse computing Cerebras CS-2 system [15] with the similar power consumption of up to 23 kW with the weight of 300 kg and placed in the 15U construction.

8. Results of Problems Implementation

The real performance of the RCD Arcturus was evaluated using two floating-point problems.

The first problem is to estimate the achievable real performance, close to the RCD peak performance. Its algorithm consists mainly of computational operations. Its implementation almost completely involves the FPGA computing resource. For this, a conveyor computing structure was used that implements the filter function with a finite impulse response (FIR) with symmetric coefficients. The computational pipeline is a series of connected filtration cascades. The performance of 150 Tflops (single precision) can be achieved at implementation of this test problem of digital data processing. Therefore, the RCD construction of 16 RCD Arcturus will provide performance of 2.4 Pflops (single precision) at solving this problem. This assessment confirms the RCD application prospects to solve time-consuming problems.

The second problem is the algorithmic basis of the LINPACK computer performance test. The SLAE solution was performed using the pipeline computing structure for implementation of the LU-decomposition function with the matrix traversal by columns. The performance of 30 Tflops (double precision) can be achieved at implementation of this test problem. The performance will be 480 Tflops (double precision) in RCD construction based on 16 RCD Arcturus at solving the LU-decomposition problem. Obviously, this assessment cannot be a characteristic for comparison with other computing architectures. However, it confirms the RCD feasibility of test problems for traditional architectures and provides the performance level.

The advantage of the Arcturus supercomputer at solving deep machine learning problems is the performance linear increase. It is fundamentally impossible for graphics accelerators (such as Tesla A100). Analysis of the Nvidia official website data [16] shows that the increasing of Tesla A100 accelerators leads to decreasing of their specific performance at the problem of the ResNet-50 v1.5 neural network training. The Arcturus is 2-6 times faster than Tesla A100 at scaling. Besides, the analysis showed that the RCD Arcturus in terms of performance/power consumption is twice as efficient as the most modern Nvidia DGX system for problems of training neural networks of image classifiers and neural networks of natural language processing at the same performance.

Compared to the Dell R7910 RACK [17], the Arcturus RCD provides acceleration more than 610 times, and efficiency in terms of performance/power consumption is 31 times higher at solving complex problems of mathematical physics.

Compared to the SuperServer 2028U-TR4T+ server [18] manufactured by Supermicro Computer based on Intel Xeon CPU E5-2699A v4, the Arcturus RCD provides acceleration more than 114 times, and efficiency in terms of performance/power consumption is 16 times higher at solving the Earth remote sensing problems.

Conclusion

The promising RCD Arcturus, developed at the Supercomputers and Neurocomputers Research Center, has 96 FPGAs of the high integration degree in its construction (3U 19"). It includes a unique layout density of the computing resource – more than 270 million logic cells in total.

At the same time, appropriate power supply and immersion cooling of the product elements are provided at solving time-consuming problems. Ensuring the required RCD characteristics in the given design required a significant complication of the PCB topology and manufacturing technology of its components.

The RCD Arcturus production is scheduled to begin in 2024. Currently, the implementation of model applied problems in the field of high-speed machine learning, the Earth remote sensing, and mathematical physics problems are being prepared at the RCD.

Note that reconfigurable supercomputers require the appropriate engineering infrastructure. Their operation requires connection by means of supply and return manifolds through fittings (or balancing valves) and flexible pipelines to the secondary cooling circuit, as well as to the power supply source and to the network hub.

The Arcturus RCD features are the wide possibilities of information exchange within the device and between devices for solving highly related problems. The achieved characteristics ensure the versatility of the RCD architecture for a wide class of problems and various fields of application. Combining many devices into a single computing circuit will allow creating collective use centers for solving time-consuming problems, world-class computing complexes with the performance of up to several dozen Petaflops.

Acknowledgements

The research is partially funded by the Ministry of Science and Higher Education of the Russian Federation as part of state assignments "Development of a Multi-Agent Resource Manager for a Heterogeneous Supercomputer Platform Using Machine Learning and Artificial Intelligence" (topic FSEG-2022-0001).

References

1. Kalyaev, A.V., Levin, I.I.: Modular-stackable multiprocessor systems with structural and procedural organization of calculations. Janus-K Publishing house, Moscow (2003). 380 p.

2. Kalyaev, I.A., Levin, I.I.: Reconfigurable computing systems based on FPGAs. SSC RAS Publishing House, Rostov-on-Don (2022). 475 p.

3. Kalyaev, I.A., Levin, I.I.: Reconfigurable multiconveyor computing systems for solving streaming problems. Information technologies and computing systems 2, 12–22 (2011)

4. Levin, I.I, Dordopulo, A.I., Fedorov, A.M., Doronchenko, Yu.I.: Development of technology for constructing reconfigurable computing systems with liquid cooling. Supercomputer Technologies. 5th All-Russian Scientific and Technical Conference. Materials, vol. 1, pp. 184–187. SFedU Publishing House, Taganrog (2018)

5. Kalyaev, I.A., Levin, I.I.: Reconfigurable computing systems with high real performance. In: International Scientific Conference, Parallel Computational Technologies, PaCT-2009. Proceedings, 186 p. SUSU Publishing House, Chelyabinsk (2009)

6. BiXBiT immersion cooling (2018). `https://bixbit.io/assets/docs/ru/investment_project.pdf`, accessed: 2023-04-19

7. Abramov, S., Amelkin, S., Klyuev, L., Chichkovsky, A.: Liquid cooling of computing complexes. Radioelectronic technologies 5, 79–82 (2017)

8. Klyuev, L., Hrebtovsky, I.: Experience in building HPC clusters of IMMERS using immersion liquid cooling (2015). `http://www2.sscc.ru/Seminars/paper/2015/2015-05-14_IMMERS.pdf`, accessed: 2023-04-19

9. Besedin, I.V., Dmitrenko, N.N., Kalyaev, I.A., *et al.*: A family of basic modules for building reconfigurable computing systems with a structural and procedural organization of computing. In: Scientific service on the Internet. Proceedings of the All-Russian Conference. pp. 47–49. MSU, RSU, IVT RAS (2006).

10. Levin, I.I., Pelipets, A.V.: Effective implementation of the parallelization on reconfigurable computer systems. Vestnik komp'iuternykh i informatsionnykh tekhnologii (Herald of computer and information technologies), 11–16 (2018). https://doi.org/10.14489/vkit.2018.08.pp.011-016

11. Levin, I.I., Dordopulo, A.I., Fedorov, A.M., Gulenok, A.A.: Reconfigurable computer based on Virtex UltraScale+ FPGAs with immersion cooling system. In: International Scientific Conference, Parallel Computational Technologies, PaCT-2017. Proceedings. pp. 27–41. SUSU Publishing Center, Chelyabinsk (2017)

12. Kalyaev, I.A., Dordopulo, A.I., Levin, I.I., Fedorov, A.M.: Evolution domestic of multichip reconfigurable computer systems: from air to liquid cooling: from air to liquid cooling. Trudy SPIIRAN 1(50), 5–31 (2017). https://doi.org/10.15622/sp.50.1

13. Levin, I.I., Doronchenko, Yu.I., Dordopulo, A.I., Levina, M.G.: High-performance reconfigurable computing system based on FPGA XILINX ULTRASCALE+. In: International Conference, XXII Kharitonov thematic scientific readings. Collection of scientific papers. pp. 284–296. FSUE "RFNC-VNIIEF", Sarov (2022).

14. Levin, I., Dordopulo, A., Fedorov, A., Doronchenko, Yu.: High-Performance Reconfigurable Computer Systems with Immersion Cooling. In: Sokolinsky, L., Zymbler, M. (eds.) Parallel Computational Technologies. PCT 2018. Communications in Computer and Information Science, vol. 910, pp. 62–76. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99673-8_5

15. Cerebras. https://www.cerebras.net/product-system/, accessed: 2023-04-20

16. NVIDIA Data Center Deep Learning Product Performance (2023). https://developer.nvidia.com/deep-learning-performance-training-inference, accessed: 2023-04-20

17. Dell Precision Rack 7910 Owner's Manual. https://www.karma-group.ru/upload/iblock/d16/precision-r7910-manual.62BF24D6F1B748C9BB6DA43755132D49.pdf, accessed: 2023-04-21

18. SPEC CPU2017 Floating Point Rate Results (2017). https://spec.org/cpu2017/results/res2017q3/cpu2017-20170828-00069.pdf, accessed: 2023-04-21

DOI: 10.14529/jsfi230205

Elements of a Digital Photonic Computer

***Dmitriy A. Sorokin*[1], *Aleksey V. Kasarkin*[1], *Aleksandr V. Podoprigora*[1]**

The paper covers a variant of architecture development of digital photonic computers. Along with quantum computers, they are one of the possible ways to overcome the crisis of computing performance. The data processing implementation in digital photonic computers at terahertz frequencies potentially provides the performance exceeding by two or more decimal orders of magnitude the performance of the most modern computing systems. The elements of digital photonic computer architecture described in the paper are focused on solving a wide class of computationally time-consuming problems in the paradigm of structural calculations. The problems of ensuring the performance and accuracy at solving problems on the developed digital photonic computer, as the processing rate should correspond to the data receipt rate, are considered. The synchronization and switching subsystem was designed and analyzed by the authors. At the synthesis (programming) stage, it forms a computational structure and provides both static and dynamic coordination of data flows in calculations.

Keywords: digital photonic computer, architecture of DPC, data flow synchronization, paradigm of structural calculations.

Introduction

The successful scientific researches and technical developments are closely related to the availability of high-performance computing systems, as well as the possibility of timely increasing the speed and quality of solving time-consuming problems. However, the development of high-performance systems element base is slowing down [1]. In modern microelectronics, this is due to the achievement of physical limits for increasing clock frequencies and the integration degree [2], and in actively promoted quantum computing [3] – technological problems of system isolating the from "white noise", poor repeatability and accuracy of experiments [4, 5].

A possible way to overcome the crisis of computer technology performance may be digital photonic computer (DPCs). These are devices, in which calculations are performed using a light flux emitted by a laser. This is similar to the electric current generated by a generator in modern microelectronics. At the same time, to ensure high performance and accuracy of calculations at solving the problems, it is advisable to develop a fully digital photonic computer. In it, the data processing is performed by photonic logic gates, such as NOT, AND, OR [6, 7], as well as triggers and functional devices, based on it. In addition, it is important to choose such architecture of DPC, which, in the absence of immediate prospects for creating photonic memory [8–11], eliminates the traditional "bottleneck" problems, and will ensure the correspondence of data transmission frequencies and the transformation speed.

This paper is devoted to the description of an architecture of DPC proposed by the authors and one of its main components – the switching and synchronization subsystem between functional devices, as well as communication system between DPCs and external data sources and receivers. The first section contains a brief overview of modern achievements in photonic calculators and an assessment of the possibility of their application at solving time-consuming problems. The second section contains a brief description of the DPC proposed architecture with the structural organization of calculations. The development principles of the synchronization and data flows switching subsystem are described. The third section presents a detailed analysis

[1]Supercomputers and Neurocomputers Research Center, Taganrog, Russia

of the main component of the synchronization and data flows switching subsystem – the operand block. Its structure, operation modes are considered. The fourth section gives the estimation of the DPC expected efficiency at solving problems of mathematical physics. The obtained results were analyzed in conclusion.

1. Review of Modern Researches on Photonic Calculators

Interest to the computing system design, in which the information is transmitted by a light stream, originated in the late 50s and early 60s of the last century. The prospect of solving complex problems with near-light speed caused significant progress at the creation an appropriate element base. This led to the appearance of a separate class of devices – optical correlators [12]. The operation principle of these calculators is based on the comparison of signals using the Fourier lens [13]. The data processing is performed in an analog format, which has all the advantages and disadvantages of analog machines. Therefore, since the late 80s and early 90s of the last century, it is necessary to integrate the correlators and digital computers. For example, the Bell Labs presented the first layout of an optical computer in 1990 [14]. The OptiComp introduced a DOC II 32-bit general purpose optical computer in 1991 [15]. In 2015, the ORNL laboratory conducted a number of researches to assess of the speed of solving the Fast Fourier Transform (FFT) problem on the EnLight Alpha computing system. It was based on the EnLight 256 optical processor, compared to the computing system based on 2 Intel Xeon 2 GHZ processors. The conducted researches have shown more than 13,000 times the acceleration in time at problem solution achieved on EnLight Alpha. However, as ORNL researchers note, the calculation speed is inversely dependent on accuracy [16].

Such computers are hybrid systems. In it, an analog converter performs the main calculations, and the data preparation, transmission and storage functions are performed by traditional microelectronic components. Currently, the development of optoelectronic hybrids continues in two directions: by increasing the computational characteristics of optical correlators by growing the resolution and performance of modulation tools, or by using invariant correlators in combination with data mining methods [12]. These technologies make it possible to process information with the bandwidth of tens of gigabits per second and obtain a satisfactory quality solution of image analysis problems. However, the optical correlators and systems based on them are not applicable for many modern time-consuming problems from such fields as gas dynamics and molecular dynamics, plasma physics and inertial fusion, and many others. It is required to perform processing in high-precision data representation formats. Only fully DPCs can provide high-speed computing at solving specified problems with the accuracy of the IEEE 754 standard level or similar.

Currently, the researches have being conducted largely on logic elements that perform operations on light pulses and provide a fully functional basis for a digital computer in Russia [17, 18] and abroad [19–21]. At the same time, few papers have been devoted to the problem of choosing the prospective DPC architecture and methods of organization of calculations [22, 23].

The structure and principles of the DPC implementation are considered in [22], which correspond to the reduction calculation model. The reduction model forms a flow graph of data processing by recursive analysis of the algorithm for problem solution. The sequence of graph operations is dynamically mapped to the system computing resource. Therefore, data is constantly transmitted over the switching network from one functional device (FD) of the system to another. RAM is not required to store intermediate calculations.

In this case, one FD either performs the requested operation, or stores one valid data. The search for exchange paths between FDs, implementing the graph operations, is performed constantly.

However, conflicts and, accordingly, temporary losses are inevitable at exchanging between FDs. This can only be eliminated by a fully connected switching system that requires large equipment costs. If the switching system is not fully connected, then the most of the FD will be occupied not with performing direct calculations, but with storing intermediate results and transferring them to other FD for subsequent calculations at solving the real applied problem. Computing equipment will be used inefficiently.

The paper [23] describes the architecture of DPC based on the paradigm of structural calculations [24]. In structural calculations, the FDs perform only informationally significant transformations, and pipeline data processing is performed at the rate of their entering at the FD inputs. All informationally insignificant operations, such as the exchange or data source selection, are implemented by spatial switching and synchronization. The processing results are transferred through physical channels to the following FDs, according to the task information graph [25], and are not buffered in memory. This allows minimizing the memory for storing the results of intermediate calculations and reducing the bandwidth requirements for photonic-electronic data exchange interfaces with external devices. The proposed solutions are aimed at efficient use of the available computing resource and preserving the advantage of high data processing frequency in the DPC over microelectronic devices.

2. Architecture of DPC with the Paradigm of Structural Calculations

Within the framework of the paradigm of structural calculations, the high efficiency can be achieved with the equality (consistency) of data exchange rate between all system components at solving time-consuming problems: RAM and system computing nodes, the data processing rate in the nodes and transmission between them.

The DPC data processing is assumed at the frequency about 1 THz [7]. Modern microelectronics technologies provide exchange with RAM at the frequency about 1 GHz. Therefore, the stream must be multiplexed for each channel bit between the RAM and DPC. Figure 1 shows a possible variant of such structure.

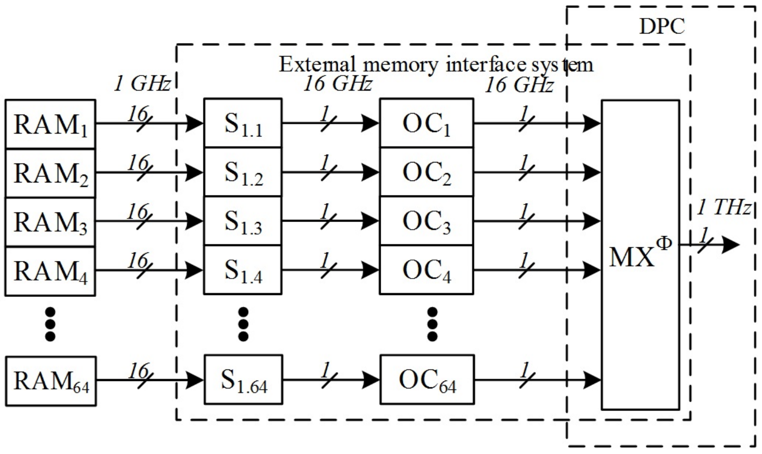


Figure 1. The proposed structure of data flow formation for DPC

RAM can be developed on SDRAM-type chips, for example. Their quantity is determined by the ratio of the required bandwidth of the digital computer channel to the bandwidth of the memory chip channel. Using DDR3 memory with 16-bit channels and the 1 GHz transmission frequency, it is necessary to use 64 chips to provide the stream about 1 Tbit/s (RAM_1–RAM_{64}).

At connection to the DPC, the electronic channels of RAM_1–RAM_{64} chips must be converted into photonic ones using the interface system with external memory. In general, such system is a set of cascades of serializers S_1–S_{64} by the MGT type, optical converters OC_1–OC_{64} by the SFP+ type and MX^{Φ} multiplexer, based on photonic logic. The similar solution can be used to develop data transmission channels of arbitrary digit capacity in DPC. The organization of output channels from the DPC into RAM will require the construction of a similar system that performs transformations in reverse order.

The transformed input data streams are processed in the DPC by a computational structure synthesized on a set of arithmetical and logical FD_1–FD_L (Fig. 2). The switching and synchronization subsystem must ensure a uniform rate of data transfer between the FD involved in processing, as well as between the DPC and external memory. Since it is proposed to solve problems in the paradigm of structural calculations, the switching and synchronization subsystem should have the high connectivity. However, the construction of the most efficient, fully connected switches involves a factorial increase in hardware costs at linear increasing of FB. This will lead to the use of most of the photonic logic for switching instead of calculations, i.e., to the violation of the basic principle of the paradigm of structural calculations: the use of most of the system hardware resource directly for calculations and a smaller part for organization of information-insignificant operations.

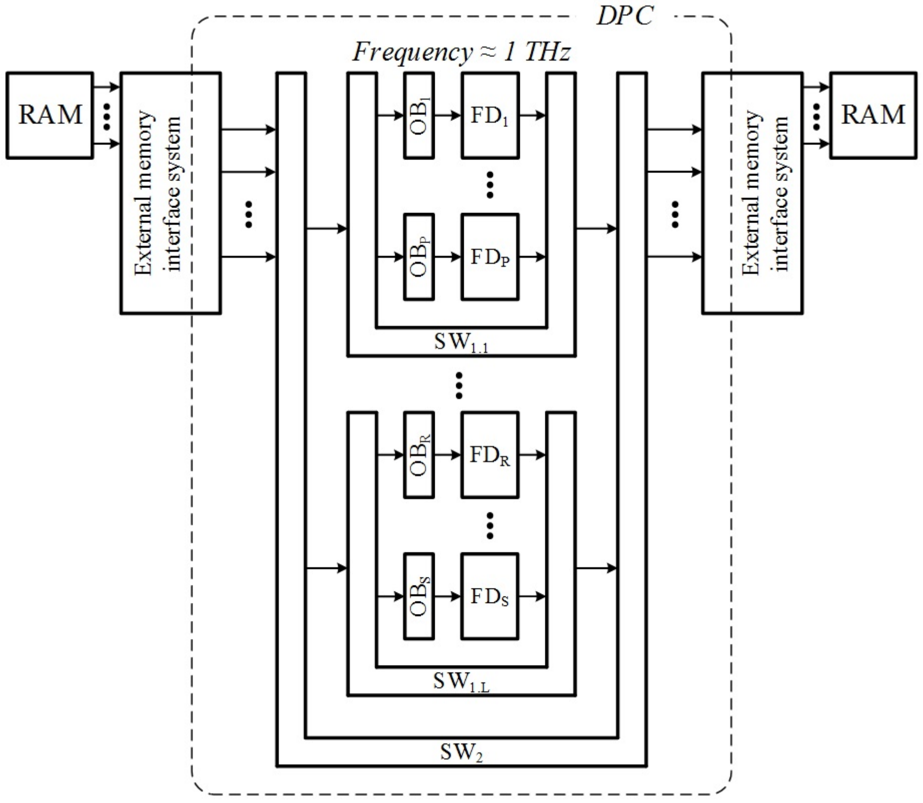

DPC
Frequency ≈ 1 THz
RAM
External memory interface system
OB1
FD1
OBP
FDP
SW1.1
OBR
FDR
OBS
FDS
SW1.L
SW2
External memory interface system
RAM

Figure 2. DPC architecture

Therefore, the switching and synchronization subsystem has a hierarchical topology in the proposed version of the DPC construction [23]. It provides more efficient use of hardware resources at solving various problems that differ by the number of involved FD and the connectivity between them. Figure 2 shows the proposed architecture of DPC.

The switching and synchronization subsystem consists of the SW static switches and blocks of dynamic switching and synchronization of operand streams (hereinafter – operand blocks, OB).

The SW static switches are devices of the "multiplexer/demultiplexer" type. Owning to it, the computational structure is synthesized on the set of FD in accordance with the task information graph at the DPC programming stage [26]. For this, the configuration parameters formed at the stage of DPC program transmission must be submitted on the control inputs of static switches. Configuration parameters cannot be changed at solving the problem.

Let us consider a variant of a computing structure, synthesized on a certain DPC fragment (Fig. 3). It implements the calculation function of a random variable dispersion

$$D = \frac{\sum_{i=1}^{n} x_i^2}{n} - \left(\frac{\sum_{i=1}^{n} x_i^2}{n}\right)^2.$$

This function is used at statistical analysis of big data, training and execution of artificial neural networks, forecasting the financial markets situation and others. Three adders, two multipliers and two dividers are necessary for construction of the pipeline computing structure that fully corresponds to the information graph of calculation D.

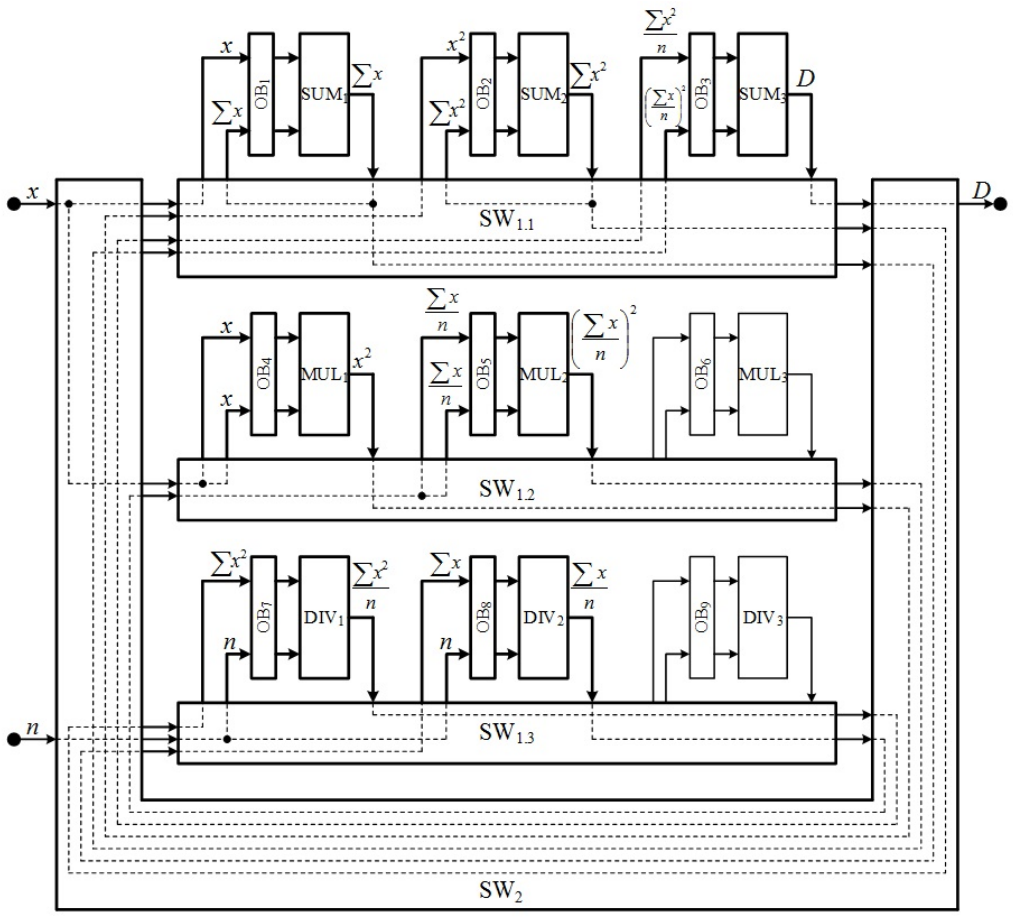


Figure 3. Computational structure of the function D

According to the Fig. 3, the selected DPC fragment contains three static switches $SW_{1.1}$–$SW_{1.3}$ of the hierarchy first level, the static switch SW_2 of the hierarchy second level, operand blocks OB_1–OB_9 and an FD array divided into three groups: SUM_1–SUM_3 adders, MUL_1–MUL_3 multipliers and DIV_1–DIV_3 dividers.

The information channels formed during the DPC programming in the static switches $SW_{1.1}$–$SW_{1.3}$ and SW_2 are shown by dotted lines.

The blocks OB_1–OB_3 are configured to operate in synchronization mode and operand streams' switching at inputs SUM_1–SUM_3. OB_4 and OB_5 are configured to operate in switching mode of operand streams at inputs MUL_1 and MUL_2. The blocks OB_7 and OB_8 are configured to operate in switching mode operand streams and the register variable n, which does not change during calculations of the current dispersion value at the inputs DIV_1 and DIV_2.

Note that the latency of various arithmetic logic FD, as well as the out of sync between different fragments of the computational structure may differ many times, in general. Therefore, it is necessary to be able to synchronize threads through external memory, both at the synthesis (programming) stage and at the DPC operation in operand blocks.

3. Data Flow Synchronization Management. Operand Block

Without violating the generality, we assume that all operations in the DPC computing structures are double. They are divided into two groups: operations with constants and threads. The necessary synchronizing component configuration is performed during the synthesis of computing structures on DPC or during data processing in OBs. The operand stream can be synchronized either with a register variable (constant), with an array of constants, or with another stream. The operand block structure OB_i is given in Fig. 4.

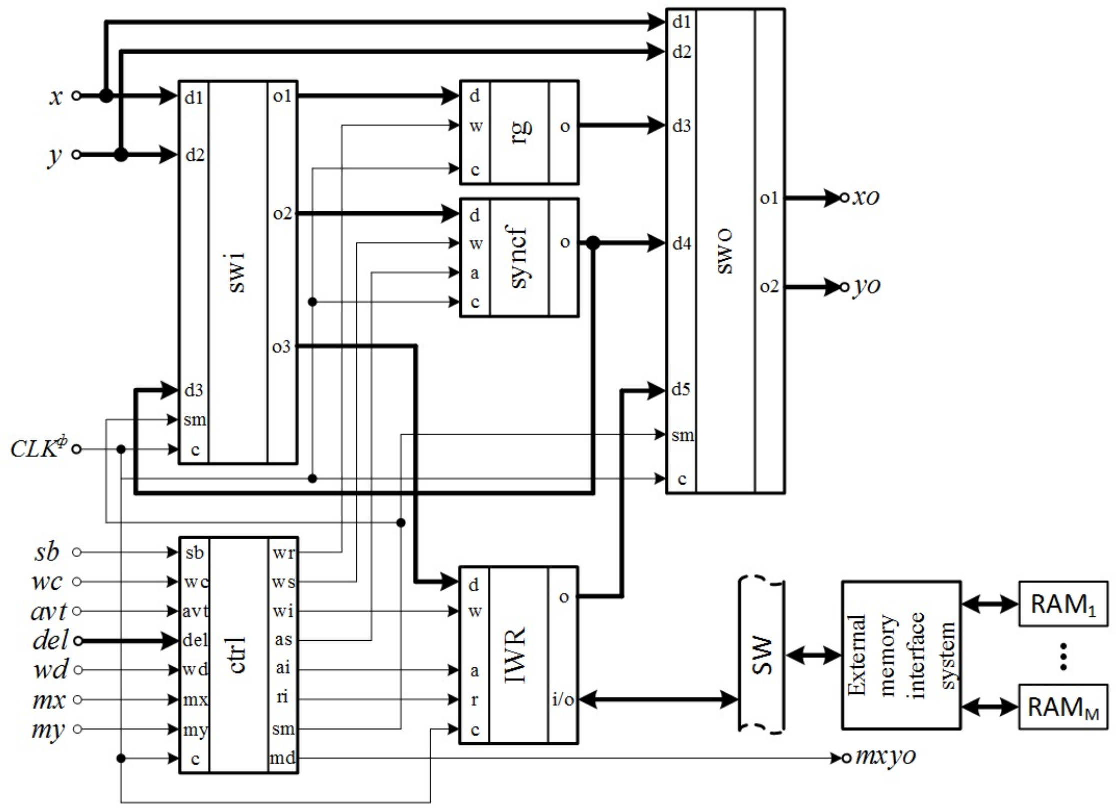


Figure 4. OB structure

The operand block consists of *swi* and *swo* dynamic switches, the *rg* variable storage register, the *syncf* operand flow synchronization block, the *IWR* external memory access interface, and the *ctrl* control block. The *ctrl* control block is designed to calculate the out of sync value, automatic or forced control of operating modes and the formation of appropriate control signals.

The *swi* and *swo* dynamic switches are also devices of the "multiplexer/demultiplexer" type. It are intended both for implementation of conditional transitions at solving problems and for synthesis of a computational structure. The parameters on their control inputs can be determined by the FD operation results or formed at the DPC program transmission.

The *rg* register is used in arithmetic and logical operations on a data stream and a variable that changes its value in no less time than the stream processing time. In this case, the variable digit capacity should not exceed the *rg* digit capacity. For example, the FP64 format of the IEEE 754 standard is used at solving mathematical physics problems. Therefore, the *rg* digit capacity must be at least 64 bits. A constant is written to this register at the DPC programming stage, or a variable is written and stored until a new entry is made at solving problem.

The *syncf* and *IWR* delay synchronizers in OBs operate according to the "shift register" principle and perform the data flows coordinating functions between the FDs. The latencies of different FD cannot coincide, and streams from different parts of the computational structure can arrive asynchronously. The *syncf* is used at operand streams matching, if the out of sync does not exceed 10^3 clock cycles. However, the out of sync value can be more than 10^3 clock cycles between fragments of computational structures. In this case, it is often necessary to use arrays of constants or intermediate data with the volume more than 10^3 bits at solving complex problems. It is necessary to use the *IWR* that provides access to external memory through the SW static switch and external memory interface system, described above. It is important to note that the number of simultaneously possible interfaces between different OBs and external memory is determined by the bandwidth of the memory channels of the entire DPC.

Taking into account the possible variety of synchronization scenarios, four main operating modes should be provided in the OB. In this case, the synchronization mode definition in the *ctrl* block can be performed at the DPC programming stage. For this, the *del* synchronization value is written in the *ctrl* block under the control of the *wd* signal, the value of which determines the operation mode. At the same time, the forced synchronization signal is set as $avt = 1$. If $avt = 0$, the determination of the operation mode will be performed automatically in accordance with the out of sync value between the mx and my markers of the x and y operand streams at solving the problem.

Operation modes 1, 2 or 3 are set according to the out of sync value between the operand streams. The mode 1 is set if the operand streams come synchronously or it is set forcibly at solving the problem. In this case, the input operand streams of the x and y are transferred directly to the outputs *o1* and *o2* of the *swu* switch. The mode 2 is set if the out of sync value is up to 10^3 clock cycles or forcibly at operating with arrays of constants with a volume up to 10^3 bits. The leading operand stream of x (or y) through the swi switch is transferred to $syncf$, and then with the lagging operands stream of y (or x) flows synchronously through the swi switch to the outputs. The mode 3 is set at the out of sync value and is more than 10^3 clock cycles or forcibly at operating with arrays of constants of more than 10^3 bits, stored in external memory. The mode 0 is set in operations with a constant. At the same time, during operation, the *rg* can be connected to the required output of the *swo* switch, or be disconnected, and the operand block can operate in modes 1–3.

The description of input signals and OB operation modes, corresponding to it, are given in Tab. 1.

Table 1. Error-free data transfer rates

Ctrl input signals						Mode number	Note
sb	*wc*	*avt*	*wd*	*mx*	*my*		
–	0	0	0	1	1	1	if Δ=0
–	0	0	0	1	1	2 or 3	mode 2, if $\Delta \leq 10^3$ mode 3, if $\Delta > 10^3$
0	1	0	0	1	0	–	rg=x
1	1	0	0	0	1	–	rg=y
0	0	0	0	1	0	0	yo=rg
1	0	0	0	0	1	0	xo=rg
–	1	0	1	0	0	–	write *del* to *ctrl*
–	0	1	0	1	1	2 or 3	mode 2, if *del*$\leq 10^3$ mode 3, if *del*$> 10^3$
0	1	1	0	1	0	–	write x to *syncf*, if *del*$\leq 10^3$ write x to *IWR*, if *del*$> 10^3$
1	1	1	0	0	1	–	write y to *syncf*, if *del*$\leq 10^3$ write y to *IWR*, if *del*$> 10^3$

At calculation the out of sync value, one clock cycle of machine time at the DPC frequency is taken as a measure unit. The calculation process of out of sync value is started and stopped by the mx or my marker of the leading operand x or y according to the formula:

$$Q_i = Q_{i-1} + 1, \quad if \quad mx_i \oplus my_i = 1,$$

where Q is a counter value; i is the clock cycle number of the counter at some time.

Fixing the desynchronization value Δ is performed according to the formula:

$$\Delta = Q_i, \quad if \quad (\overline{mx_{i-1}} \vee \overline{my_{i-1}}) \& mx_i \& my_i = 1.$$

If only one of the x or y channels starts receiving the data into the OB, the *ctrl* is in the mode 0. At the same time, the leading data from x (or y) channel is transferred to the input $x0$ (or $y0$), and a constant from *rg* is transferred to the output $y0$ (or $x0$). In general, it is not known in advance whether the second data stream will come, so the determination process of Δ value starts automatically. Until the Q value achieves the value 10^3, the input operand stream is written by *syncf* under the signal control $ws = 1$. If the Q value becomes greater than 10^3, then the signal $wi = 1$ is generated and the input operand stream from *syncf* is rewritten to external memory via *IWR*. In the case, after i clock cycles after the arrival of the operand leading stream, the lagging stream of operands begins to arrive ($mxi = myi = 1$), then the *ctrl* sets the operating mode 2 or 3 based on the value of the out of sync value $\Delta > 0$. The *ctrl* block allows forcing the mode 2 or 3. For this, the value of the forced delay is written over the *del* bus at $wc = 1$ and $wd = 1$. Then the signals $avt = 1$, $wc = 0$ and $wd = 0$ are set.

Figure 5 shows the mode and transition diagram of the *ctrl* control unit. In accordance with this diagram, the configuration and installation algorithms of synchronization modes of the input operand streams x and y are defined. According to the Fig. 5, the "Mode 0" is the base for control block. Therefore, transitions from it to any operation modes are possible.

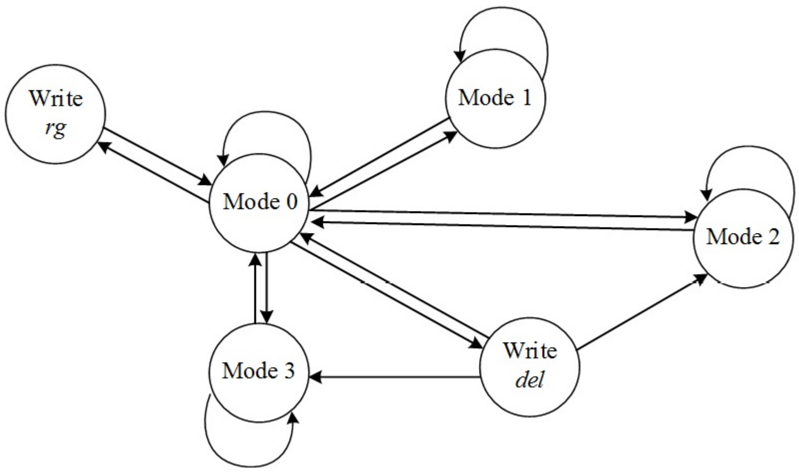


Figure 5. Diagram of transitions and the *ctrl* signals modes

Table 2 shows the basic conditions for switching the *ctrl* block to each operating mode and installation of the corresponding signals and data: *sm* mode numbers; *md* data markers; *wi*, *ws*, *wr* writting enable signals; *as* and *ai* delay depths.

Table 2. Basic conditions for transitions and the *ctrl* signals modes

The mode	Transition condition	Action
"Mode 0"	$(mx \oplus my)\&\overline{wc}\&\overline{avt}\&\overline{wd} = 1$	$sm = 0; ws = 1; md = 1$

"Write rg"	$(mx \oplus my)\&wc\&\overline{avt}\&\overline{wd} = 1$	$sm = 0; wr = 1; ws = 0; md = 0;$
		rg=x, if sb=0;
		rg=y, if sb=1;
		switch to "Mode 0"

"Mode 1"	$mx\&my\&(\Delta = 0)\&\overline{ws}\&\overline{avt}\&\overline{wd} = 1$	$sm = 1; ws = 0; md = 1$
"Mode 2"	$mx\&my\&(0 < \Delta \leq 10^3)\&\overline{ws}\&\overline{avt}\&\overline{wd} = 1$ or $mx\&my\&(0 < del \leq 10^3)\&\overline{ws}\&\overline{wd} = 1$	$sm = 2; ws = 1; wi = 0; md = 1;$ as=Δ, if avt=0; as=del, if avt=1
"Mode 3"	$mx\&my\&(\Delta > 10^3)\&\overline{ws}\&\overline{avt}\&\overline{wd} = 1$ or $mx\&my\&(del > 10^3)\&\overline{ws}\&\overline{wd} = 1$	$sm = 3; wi = 1; ws = 0; md = 1;$ $ai = \Delta$, if avt=0; ai=del, if avt=1
"Write del"	$wc\&wd\&\&\overline{avt}\&\overline{mx}\&\overline{my} = 1$	write *del* in *ctrl*

The considered modes of the *ctrl* block provide the necessary OB functional set for synthesis of computational structures or data processing on synthesized structures at solving the problems.

The construction of *rg* and *syncf* elements is of particular importance for OB implementation. These elements perform memory functions at storing constants and synchronizing threads. As noted earlier, the prospects for the photonic memory creation are very indefinite. It is possible to use only the DPC trigger elements for their implementation. At the same time, DPC triggers do not have an information storage mode. Therefore, it is necessary to use a "ring buffer" scheme to develop the delay and store register of the *rg* constant. The *rg* block diagram, implemented in the DPC basis, has the form as shown in Fig. 6.

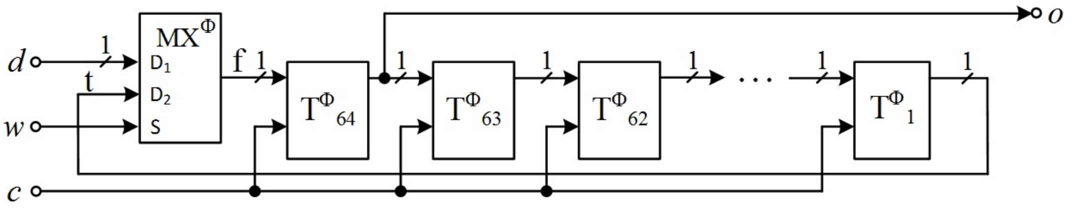


Figure 6. The *rg* block diagram

The *rg* block includes:
– T^{Φ}_{64}–T^{Φ}_{1} are DPC triggers;
– MX^{Φ} is the DPC multiplexer.

The time diagram of the *rg* operation is given in Fig. 7.

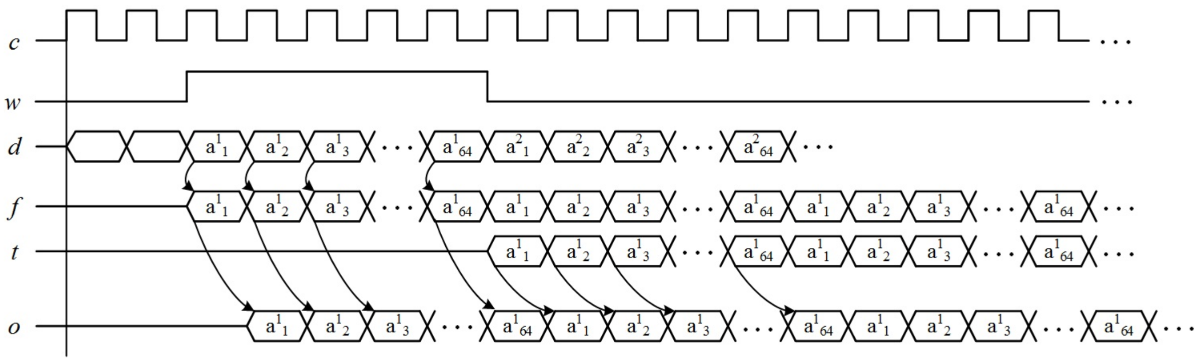


Figure 7. Time diagram of *rg* operation

When the logical “1” is applied to input w, the MX^{Φ} multiplexer output switches to bus d, and the input data is written to the T^{Φ}_{64}–T^{Φ}_{1} triggers. As soon as the logical “0” is received at the input w, the MX^{Φ} output will switch to the bus t, closing the ring connection. According to the time diagram, the output data $< a^1_1, a^1_2, a^1_3, \dots, a^1_{64} >$ appears at the output o after 1 clock cycle. If necessary, the delay can be increased without increasing hardware costs up to 64 clock cycles by switching the output bus o to the appropriate trigger.

The described approach to the construction of the n-bit data storage register assumes sequential one-bit processing. At the same time, one MX^{Φ} multiplexer and n T^{Φ} triggers are required to implement the *rg*.

Such hardware costs for the T^{Φ} resource completely coincide with the costs of $n$bit data parallel processing with any bit reduction coefficient r by digits [27]. The hardware costs of the MX^{Φ} resource are determined by the ratio $\dfrac{n}{r}$.

The computational structure of the *syncf* block included in the OB is shown in Fig. 8.

The *syncf* block allows the delay of operands by Δ clock cycles. It consists of N serially connected DPC triggers T^{Φ}_{1}–T^{Φ}_{N} and MX multiplexer. The delay value is set by transfer of the value Δ to the input a. The time diagram of the *rg* operation is shown in Fig. 9.

Three options of operand stream synchronization on the *syncf* block are shown in the diagram: 1 is the synchronization of operand streams with the difference $a = 3$ clock cycles; 2 is the synchronization of operand streams with the difference $a = 5$ clock cycles; 3 is the synchronization of operand streams with the difference $a = 6$ clock cycles.

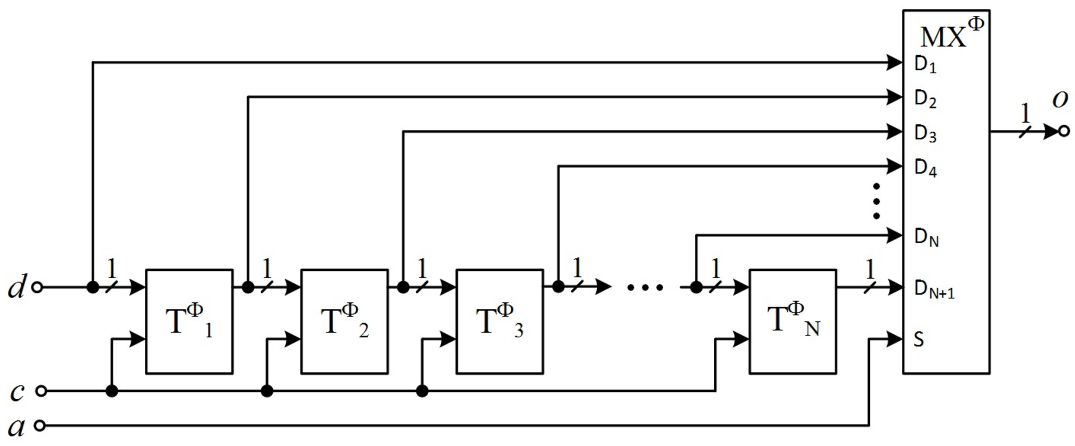


Figure 8. *Syncf* block diagram

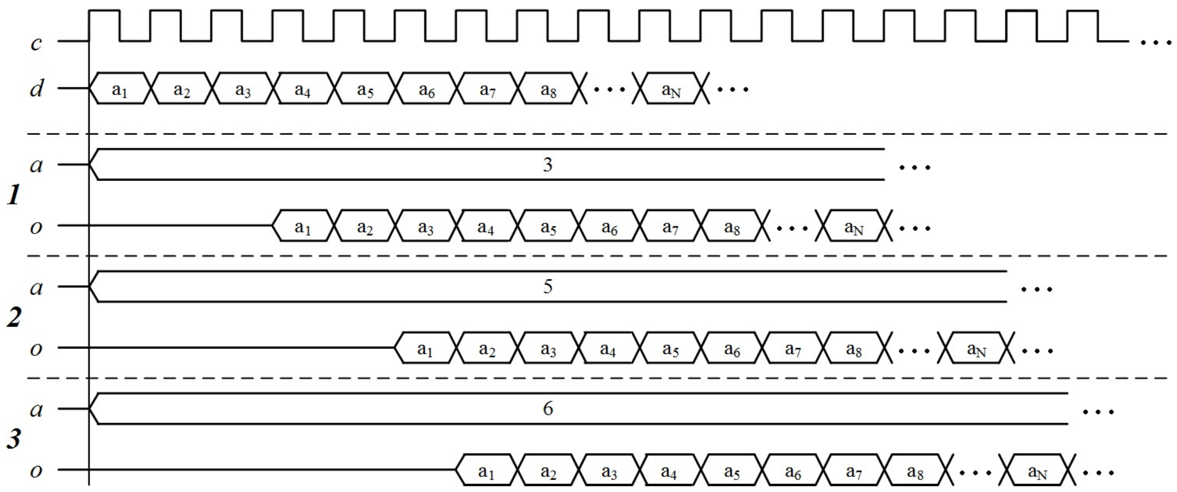


Figure 9. Time diagram of *syncf* operation

To organize data delays synchronization by Δ clock cycles during sequential one-bit processing, the Δ of T^Φ triggers and one MX^Φ multiplexer will be required. This is r times less compared to the costs at parallel data processing. The increasing of the processed data digit capacity will lead to corresponding increase in the hardware costs for *syncf* implementation.

Theoretically, the DPC data processing can be performed in a format of any digit capacity. However, taking into account the necessity to coordinate data flows between the DPC and external memory, the increase of digit capacity of data processing at solving problems on the DPC in the paradigm of structural calculations leads to the same increase in the cost of implementation of external microelectronic memory. Therefore, the DPC data processing will be present in the minimum possible single-digit format in the near future.

4. Performance Evaluation of the Proposed Solutions

The performance P of computational structure D described in the third section was evaluated according to the following formula $P = (f \cdot N)/S$, where f is the DPC operation frequency; N is the number of FD, performing arithmetic operations in the 64FP IEEE754 standard; S is the duty cycle of input data.

At $f = 1THz$, $N = 3$, $S = 64$ the theoretical performance of the computational structure will be equal to $P = 46.875$ Gflops. This corresponds to the performance of modern processors.

Theoretical researches of the DPC performance with the structural organization of calculations were performed on the DPC functional prototype made on the RCS "Tertius-2T" [28]. The computational structure is synthesized for solving five-diagonal SLAE by dimension 10^4 and the grid step of 0.01 by the Gauss-Seidel method [29] of 60 sequentially connected pipes, each of which performs one algorithm iteration. The one pipe contains five one-bit OB and five one-bit FD: four adders and one multiplier performing calculations in the 64FP standard. In addition, three one-bit static switches of the first level, one one-bit static switch of the second level and six one-bit RAM access channels are used to develop the computational structure.

Researches have shown that such computing structure on DPC, provided it operates at the 1 THz frequency, will solve the problem in about 430 μs. The computational structure of 60 pipes on one FPGA XCKU095 RCD "Tertius-2T" solves a five-diagonal matrix of about 0.15 s, and the Intel Core i5-12600K 3.4 GHz processor is about 2 s. Therefore, the DPC can provide acceleration by 340 times compared to the modern FPGAs [30], and compared to modern processors – by 4500 times, subject to an equivalent amount of hardware costs.

Conclusion

The architecture of DPC and approaches to the construction of one of its main elements – the data flow switching and synchronization subsystem are described in the paper. In the future, they will make it possible to effectively use the available DPC computing resource and maintain the performance gain over microelectronic devices due to the higher frequency of data processing.

The proposed scheme for block construction of dynamic switching and synchronization of operand flows provides basic functionality both for synthesis of computing structures on DPC and matching the rate of operand flows during solving problems of mathematical physics. Further reseaches of the proposed architecture of DPC and development of the functionality of its elements will potentially expand the class of problems efficiently solved on the DPC that require high computational accuracy.

The estimates performed by the authors show that the DPC with the data flow architecture and the structural organization of calculations at solving time-consuming problems of mathematical physics currently have the potential to provide performance exceeding by two or more decimal orders of magnitude the performance of modern computing systems.

Acknowledgements

The work was carried out within the framework of the scientific program of the National Center for Physics and Mathematics (the project "National Center for Supercomputer Research").

References

1. Chernyak, L.: Amdahl's Law and the future of multicore processors. Open systems. DBMS 04 (2009). https://www.osp.ru/os/2009/04/9288815/, accessed: 2023-05-02

2. Moore, G.: No Exponential is Forever: But "Forever" Can Be Delayed! In: International SolidState Circuits Conference (ISSCC), 2003. Session 1. Plenary 1.1. Vol. 91, no. 11, pp. 1934–1939. IEEE (2003). https://doi.org/10.1109/ISSCC.2003.1234194.

3. Benioff, P.: Quantum mechanical hamiltonian models of turing machines. Journal of Statistical Physics 29(3), 515–546 (1982). https://doi.org/10.1007/BF01342185

4. D-Wave Announces General Availability of First Quantum Computer Built for Business. https://www.dwavesys.com/company/newsroom/press-release/d-wave-announces-general-availability-of-first-quantum-computer-built-for-\business/, accessed: 2023-05-02

5. Dalzell, A.M., Harrow, A.W., Koh, D.E., Placa, R.L.L.: How many qubits are needed for quantum computational supremacy? Quantum 4, 264 (2020). https://doi.org/10.22331/q-2020-05-11-264

6. Shubin, V.V., Balashov, K.I.: Fully optical logical basis based on a micro-ring resonator. Patent No. 2677119. The Federal State Unitary Enterprise "Russian Federal Nuclear Center - All-Russian Research Institute of Experimental Physics" (FSUE RFNC-VNIIEF), Rosatom VNIEF

7. Tamer, A.: Moniem All-optical XNOR gate based on 2D photonic-crystal ring resonators. Quantum Electronics 47(2), 169 (2017). https://doi.org/10.1070/QEL16279

8. Next generation photonic memory devices are "light-written", ultrafast and energy efficient (2019). https://www.tue.nl/en/news/news-overview/10-01-2019-next-generation-photonic-memory-devices-are-light-written-ultra\fast-and-energy-efficient/, accessed: 2023-05-02

9. Using light for next-generation data storage (2018). https://phys.org/news/2018-06-next-generation-storage.html, accessed: 2023-05-02

10. Zhang, Q., Xia, Z., Cheng, Y.B., *et al.*: High-capacity optical long data memory based on enhanced Young's modulus in nanoplasmonic hybrid glass composites. Nat Commun 9, 1183 (2018). https://doi.org/10.1038/s41467-018-03589-y

11. Gordeev, A., Voitovich, V., Svyatets, G.: Promising photonic and phonon domestic technologies for terahertz microprocessors, RAM and interface with ultra-low power consumption. Modern Electronics 2(22). https://www.soel.ru/online/perspektivnye-fotonnye-i-fononnye-otechestvennye-tekhnologii-dlya-teragert\sovykh-mikroprotsessorov-o/, accessed: 2023-05-02

12. Starikov, R.S.: Optical image correlators: History and current state. In: XVI International Conference on Holography and Applied Optical Technologies, HOLOEXPO 2019. Abstracts. pp. 82–90. Bauman Moscow State Technical University, Moscow (2019)

13. Lugt, A.V.: Signal detection by complex spatial filtering. IEEE Transactions on Information Theory 10(2), 139–145 (1964). https://doi.org/10.1109/TIT.1964.1053650

14. Arsenault, H.H., Sheng, Y.: An Introduction to Optics in Computers. Volume 8 of Tutorial texts in optical engineering. SPIE Press, Washington (1992). 126 p.

15. Stone, R.V., Zeise, F.F., Guilfoylev, P.S.: DOC II 32-bit digital optical computer: optoelectronic hardware and software. In: Optical Enhancements to Computing Technology. Proceedings, vol. 1563. SPIE (1991). https://doi.org/10.1117/12.49689

16. Barhen, J., Kotas, C., Humble, T.S., *et al.*: High performance FFT on multicore processors. In: 2010 Proceedings of the Fifth International Conference on Cognitive Radio Oriented Wireless Networks and Communications, (CROWNCOM). pp. 1–6. IEEE (2010). https://doi.org/10.4108/ICST.CROWNCOM2010.9283

17. Stepanenko, S.A.: Interference logic elements. Reports of the Russian Academy of Sciences. Mathematics, Computer Science, Management Processes 493, 68–73 (2020)

18. Kuznetsova, O.V., Speransky, V.S.: Solving optical signal processing problems without optoelectronic conversion. Telecommunications and Transport. T-Comm. 8, 35–39 (2012)

19. Wu, X., Tian, J., Yang, R.: A Type of All-Optical Logic Gate Base on Graphene Surface Plasmon Polaritons. Optics Communications 403, 185–192 (2017). http://doi.org/10.1016/j.optcom.2017.07.041

20. Papaioannou, M., Plum, E., Valente, J., *et al.*: All-Optical Multichannel Logic Based on Coherent Perfect Absorption in a Plasmonic Metamaterial. APL PHOTONICS 1, 090801 (2016). https://doi.org/10.1063/1.4966269

21. Hussein, M.E., Ali, T.A., Rafab, N.H.: New Design of a Complete Set of Photonic Crystals Logic Gates. Optics Communications 411, 175–181 (2018). https://doi.org/10.1016/j.optcom.2017.11.043

22. Stepanenko, S.A.: Photonic computing machine. Principles of implementation. Parameter estimates. Reports of the Academy of Sciences 476(4), 389–394 (2017). https://doi.org/10.1134/S1064562417050234

23. Levin, I.I., Sorokin, D.A., Kasatkin, A.V.: Perspective architecture of a digital photonic computer. Izvestiya of the SFeDu. Technical sciences 6(230), 61–71 (2022). https://doi.org/10.18522/2311-3103-2022-6-61-71

24. Besedin, I.V., Dmitrenko, N.N., Kalyaev, I.A., *et al.*: A family of basic modules for building reconfigurable computing systems with a structural and procedural organization of computing. In: Scientific service on the Internet. Proceedings of the All-Russian Conference. pp. 47–49. MSU, RSU, IVT RAS (2006)

25. Kalyaev, I.A., Levin, I.I.: Reconfigurable multiconveyor computing systems for solving streaming problems. Information technologies and computing systems 2, 12–22 (2011)

26. Kalyaev, I.A., Levin, I.I., Semernikov, E.A., Shmoilov, V.I.: Reconfigurable Multipipeline Computing Structures. Nova Science Publishers, Inc., New York, USA (2012). 345 p.

27. Dordopulo, A.I., Sorokin, D.A.: Methodology for reducing hardware costs in complex systems at solving problems with significantly variable intensity of data flows. Izvestiya SFeDu. Technical sciences 4(129), 194–199 (2012)

28. Supercomputers and Neurocomputers Research Center. Tertsius-2. http://superevm.ru/index.php?page=tertsius-2, accessed: 2023-05-02

29. Shpakovsky, G.I., Verkhoturov, A.E.: Algorithm of parallel SLOUGH solution by Gauss-Seidel method. Bulletin of BSU 1(1), 44–48 (2007)

30. Kalyaev, I.A., Levin, I.I.: Reconfigurable computing systems with high real performance. In: International Scientific Conference, Parallel Computational Technologies, PaCT-2009. Proceedings. SUSU Publishing House, Chelyabinsk (2009)

DOI: 10.14529/jsfi230206

Neuromorphic Computing Based on CMOS-Integrated Memristive Arrays: Current State and Perspectives*

Alexey N. Mikhaylov*[1]*, Evgeny G. Gryaznov*[1]*, Maria N. Koryazhkina*[1]*,
Ilya A. Bordanov*[2]*, Sergey A. Shchanikov*[1,3]*, Oleg A. Telminov*[4]*,
***Victor B. Kazantsev*[1,3]**

The paper presents an analysis of current state and perspectives of high-performance computing based on the principles of information storage and processing in biological neural networks, which are enabled by the new micro- and nanoelectronics component base. Its key element is the memristor (associated with a nonlinear resistor with memory or Resistive Random Access Memory (RRAM) device), which can be implemented on the basis of different materials and nanostructures compatible with the complementary metal-oxide-semiconductor (CMOS) process and allows computing in memory. This computing paradigm is naturally implemented in neuromorphic systems using the crossbar architecture for vector-matrix multiplication, in which memristors act as synaptic weights – plastic connections between artificial neurons in fully connected neural network architectures. The general approaches to the development and creation of a new component base based on the CMOS-integrated RRAM technology, development of artificial neural networks and neuroprocessors using memristive crossbar arrays as computational cores and scalable multi-core architectures for implementing both formal and spiking neural network algorithms are discussed. Technical solutions are described that enable hardware implementation of memristive crossbars of sufficient size, as well as solutions that compensate for some of the deficiencies or fundamental limitations inherent in emerging memristor technology. The performance and energy efficiency are analyzed for the reported prototypes of such neuromorphic systems, and a significant (orders of magnitude) gain in these parameters is highlighted compared to the computing systems based on traditional component base (including neuromorphic ones). Technological maturation of a new component base and creation of memristor-based neuromorphic computing systems will not only provide timely diversification of hardware for the continuous development and mass implementation of artificial intelligence technologies but will also enable setting the tasks of a completely new level in creating hybrid intelligence based on the symbiosis of artificial and biological neural networks. Among these tasks are the primary ones of developing brain-like self-learning spiking neural networks and adaptive neurointerfaces based on memristors, which are also discussed in the paper.

Keywords: memristor, CMOS integration, neuromorphic hardware, artificial intelligence.

Introduction

The fourth industrial revolution, on the brink of which the humanity stands, presents entirely new requirements for the hardware of artificial intelligence (AI) technologies, which should approach the capabilities of the human brain (natural intelligence). In addition to demands for compactness and energy efficiency, new AI hardware must be compatible with existing silicon microelectronics technology and with biological systems. Meeting these requirements will enable mass production of AI hardware systems and the implementation of new hybrid forms of AI. The second requirement implies that new electronic AI systems must not only replicate formally

*The paper is published as part of special issue "PST2023: Perspective Supercomputer Technologies".
[1]Lobachevsky State University of Nizhny Novgorod, Nizhny Novgorod, Russian Federation
[2]Murom Institute of Vladimir State University, Murom, Russian Federation
[3]Moscow Institute of Physics and Technology, Moscow, Russian Federation
[4]Joint-Stock Company Molecular Electronics Research Institute, Moscow, Russian Federation

(as they do now), but also reproduce functionally the elements of the nervous system and the brain.

Current paradigmatic changes in electronics are aimed at meeting these requirements associated with the transition from the traditional von Neumann architecture (in which memory and processing are separated in space) to analog computing in memory and massive parallelism in information processing similar to that in the brain. At the core of the new post-digital paradigm is a brain-like electronic component base represented by memristors (analog resistive memory devices) and memristive systems that mimic the functions of elements of the living nervous system (neurons and synapses). The diversity of possible computing architectures is ensured by the universal character of the memristive phenomenon, as it can be implemented in classical and quantum systems, in various artificial materials and structures (inorganic, organic, molecular, etc.), and in living systems.

The results of comprehensive research and diverse applications of memristive devices have become the subject of numerous publications in recent years (e.g., see [4, 12, 19, 20, 53, 55, 60], including roadmaps, reviews, and perspectives in top journals) showing the importance and relevance of this field at the global level, as well as the need to implement a master plan (coordinated and interdisciplinary efforts) in the field of bioinspired systems aimed at technological development of the new component base and creating prototypes of next-generation information-computing systems.

This paper presents the current state and prospects of high-performance computing based on memristors. We consider general approaches to the development of Resistive Random Access Memory (RRAM) integrated with complementary metal-oxide-semiconductor (CMOS) technology required for creating elements and functional blocks of a memristive neuroprocessor, as well as the application of new computing systems in artificial and hybrid intelligence technologies. To show the perspectives of memristor-based neuromorphic computing systems, their achieved parameters are compared to that of traditional computing systems.

The paper is organized as follows. Section 1 is devoted to a discussion of the relevance and prospects of research and development of memristors and memristor-based neuromorphic and neurohybrid systems. In Section 2, we discuss a multilevel and interdisciplinary approach to the development of neuromorphic systems based on CMOS-compatible memristive devices. Section 3 contains a consideration of various options for scaling up the CMOS-integrated memristive crossbars to increase the speed of signal transmission in artificial neural networks. Section 4 contains a comparison of neuromorphic computing systems based on traditional and new component base. Conclusion summarizes the study.

1. Memristor and Memristor-Based Neuromorphic and Neurohybrid Systems

Over the past five decades, global microelectronics has developed according to Moore's law, which predicts an exponential increase in the number of transistors on a chip, resulting in faster computing and reduced energy consumption for each new generation of technology. Currently, this trend has reached a physical limit – further increase in the number of transistors does not lead to an increase in clock speed or a reduction in energy consumption. The main bottleneck is the data exchange between the central processor and external memory, making digital processors based on traditional von Neumann architecture extremely inefficient in terms of

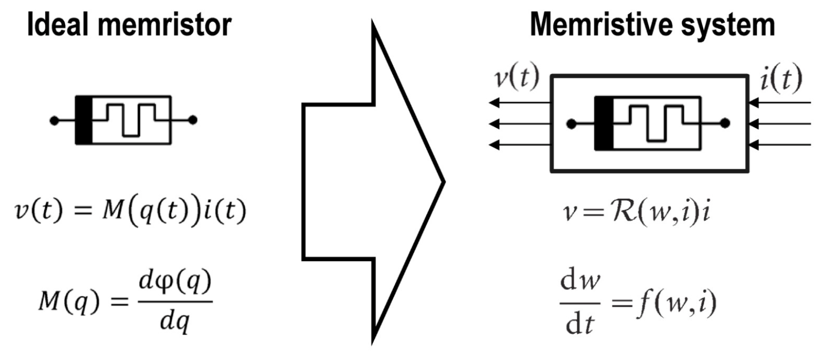


Figure 1. Original and generalized definitions of memristor [13, 14]

energy consumption and time delays. Meanwhile, the volume of digital data requiring processing continues to increase exponentially. Every two years, more data is created than in all of human history before that point. Unstructured data already comprises over 80 % of the total volume of data generated daily. Thus, the demand is growing faster than the performance of modern computers. Breakthrough technological solutions are required to address this von Neumann bottleneck. Currently, two main solutions are being explored in leading scientific centers around the world – combining computation and memory in a single functional unit, and transitioning from traditional von Neumann architectures to neuromorphic architectures that reproduce the principles of information storage and processing in the nervous system and brain.

The new paradigm in electronics, which is associated with a breakthrough in the hardware implementation of neuromorphic information-processing systems, is based on the use of memristors. The memristor (memory resistor) was theoretically described by Leon Chua in 1971 as a missing passive element of electrical circuits that relates the change in magnetic flux $\phi(t)$ to the electrical charge $q(t)$ [13] (Fig. 1). It can be shown that this element is equivalent to a nonlinear resistor that changes its resistance $M(q(t))$ depending on the history of the electrical charge flowing through it. This definition of an ideal memristor still causes doubts and disputes among scientists [15, 22, 50] and stimulates the search for materials and structures that exhibit a physical connection between magnetic and electrical properties [45]. However, in 1976 L. Chua and S. Kang proposed a generalized definition of memristors and memristive dynamical systems [14] that are described by a port equation equivalent to Ohm's law and a set of state equations that describe the dynamics of the internal state variables (w). This definition is universal and describes the change in resistance (memory effect) based on various phenomena in inorganic and organic nanomaterials (ion migration, redox reactions, phase transitions, spin and ferroelectric effects) [53], as well as in photonic [46] and superconducting [37, 41] circuits. Among them, it is necessary to highlight nanostructures of the metal-oxide-metal (MOM) type, which are ideal for creating compact (with nanometer-scale size) and energy-efficient (with femtojoules per switch) RRAM devices that can be integrated into the standard CMOS technological process. Such devices can not only store the logical value determined by conductivity, but also allow it to be changed in the same physical location implementing a non-von Neumann paradigm of in-memory computing. In addition, the simple structure of memristor enables the creation of ultra-dense and, in the future, three-dimensional arrays of crossbars that naturally (based on Ohm's and Kirchhoff's laws and in analog form) implement vector-matrix multiplication (VMM) operations, which underlies inference in traditional artificial neural networks with deep learning and new algorithms for training spiking neural networks [31].

The development of AI technologies relies on the development of neuromorphic computing systems according to the well-known forecast within the international technology roadmap: "The Future of AI is Neuromorphic". Brain-like electronic components with memristors and memristive systems will provide timely diversification of hardware, which mainly imposes fundamental limitations on each cycle of AI development, and will prevent another "winter" of AI. Alternative neuromorphic technologies based on new component base are only just entering maturity, competing with currently dominant digital high-performance computing technologies. A detailed analysis and comparison of the achieved characteristics of neuromorphic computing systems based on memristors and traditional component base have been previously presented in the literature [4, 60], but every year new prototypes and records (see, e.g., [5, 51, 52, 61]) are reported, which are discussed in Section 4. According to the roadmap for brain-inspired computing chips [60], creating memristive general-purpose neuroprocessors is expected within the next 5–10 years. The prototypes of memristive computing systems demonstrated now already compete with the well-known neuromorphic processors based on traditional digital components and specialized architectures (ASIC) [4].

Despite all the successes in the development of AI technologies and the impressive progress in the development of specialized computing systems that implement neural network algorithms, more attention is being paid to the prospects for significantly deeper adaptation of neuromorphic principles than has been achieved so far [38]. In addition to being similar in form and essence to the functioning of the brain, neuromorphic systems (in their narrow understanding) implemented on the basis of memristive systems have significant potential for achieving a new level of cognitive abilities, primarily by means of efficient real-time processing of the electrical activity of biological neural systems as part of so-called bio- or neurohybrid systems [11, 17, 40]. At the same time, the first known examples from the literature in which memristive devices and arrays have been used to process bio-electrical activity only record the fact of communication between electronic and biological systems through individual memristive devices [43] or do so in isolation from the living systems (for example, in recently published papers [29, 30, 62], memristive chips are used to process emulated sequence of rectangular spikes or signals of neuronal activity taken from publicly available databases).

Remarkable progress in the development of memristive neurohybrid systems has been reported in the paper [44], which demonstrates the first bidirectional adaptive neurointerface based on advanced solutions in the field of memristive electronics and neuroengineering (Fig. 2).

A culture of hippocampal neuron cells with functional connections between neuron groups spatially ordered with the help of a microfluidic chip has been used on a multi-electrode array from the side of a living system. A memristive network is used not only to solve the problem of nonlinear classification of the spatial-temporal response of a cell culture to electrical stimuli, but also to control its functional state. Specifically, the output signals of the memristive network correspond to different stimuli and are used for adaptive stimulation control, which allows for the restoration of disrupted functional connections in the neural culture.

There is a great interest in the prospects of using such neurohybrid technologies for neurorehabilitation tasks, restoring or reorganizing biological neuronal functions after the development of a pathological condition [18].The perspective of creating cell cultures that highly reproduce brain architectural features is extremely attractive both from the standpoint of a convenient experimental model and from the standpoint of their use in real neurohybrid technology [10].

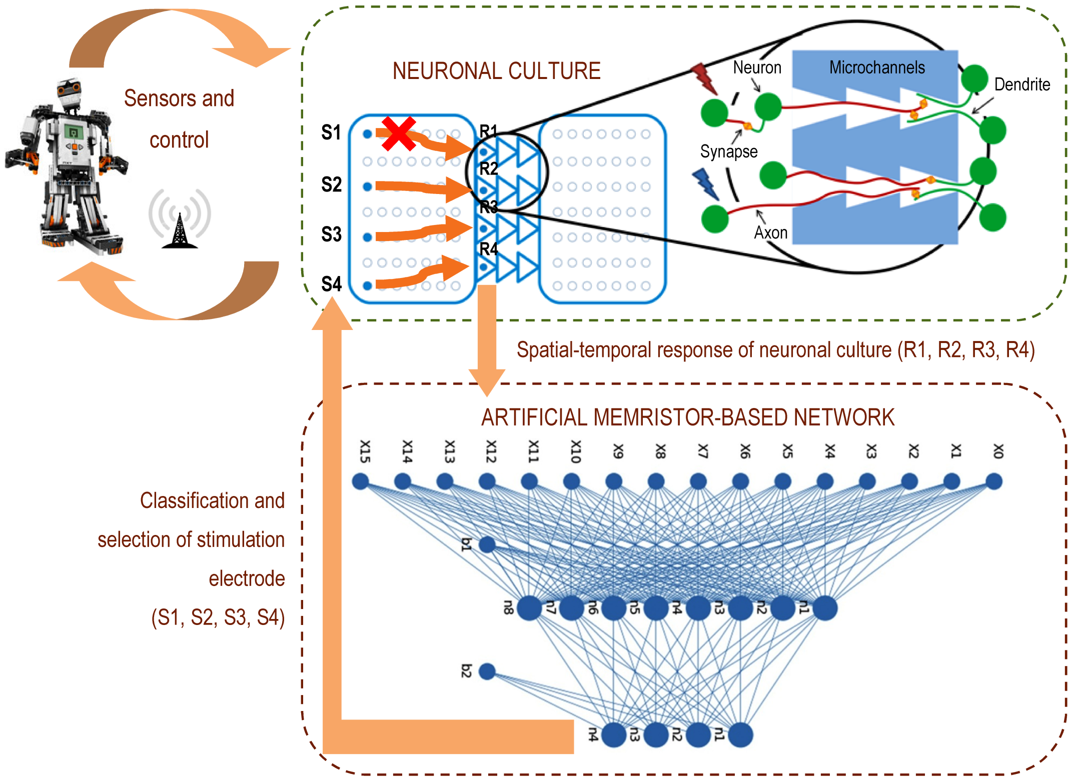

Sensors and control
NEURONAL CULTURE
S1
S2
S3
S4
R1
R2
R3
R4
Neuron
Microchannels
Dendrite
Synapse
Axon
Spatial-temporal response of neuronal culture (R1, R2, R3, R4)
ARTIFICIAL MEMRISTOR-BASED NETWORK
X15
X14
X13
X12
X11
X10
X9
X8
X7
X6
X5
X4
X3
X2
X1
X0
b1
n8
n7
n6
n5
n4
n3
n2
n1
b2
n4
n3
n2
n1
Classification and selection of stimulation electrode (S1, S2, S3, S4)

Figure 2. Bidirectional adaptive neurointerface between ordered neuronal culture and memristor-based artificial neural network [44]

Thus, the combination of high energy efficiency and unique scalability of memristive systems allows for a decisive step from neuromorphic computing systems to neurohybrid systems based on direct (physiological) and safe interaction between artificial electronic systems and living neuronal systems [33]. As a result, memristive neuromorphic systems will have a worthy place in AI medical technologies, providing not only efficient solutions to traditional AI tasks related to processing and analyzing biomedical data, but also creating compact and energy-efficient adaptive systems for replacing / restoring lost or improving existing brain and nervous system functions (neuroprosthetics and instrumental correction / support / enhancement of human cognitive abilities).

2. General Approach to Creating Memristor-Based Neuromorphic Computing Systems

According to recent perspectives [20, 32], research and development in the field of neuromorphic and brain-inspired computing systems are characterized by a complex (multi-level) and interdisciplinary nature. The first characteristic implies that new functional products are born from the co-optimization of solutions at the levels of materials, devices, and systems. The interdisciplinary nature not only requires the integration of different scientific communities (although this is already a big challenge in itself), but also the implementation of a coordinated plan, financing, and support (essentially, a master plan, as we have seen in the field of digital or quantum technologies, for example). In this section, let us consider how this combined approach is implemented in the case of developing neuromorphic and neurohybrid systems [33] based on CMOS-compatible MOM devices with resistive switching (Fig. 3).

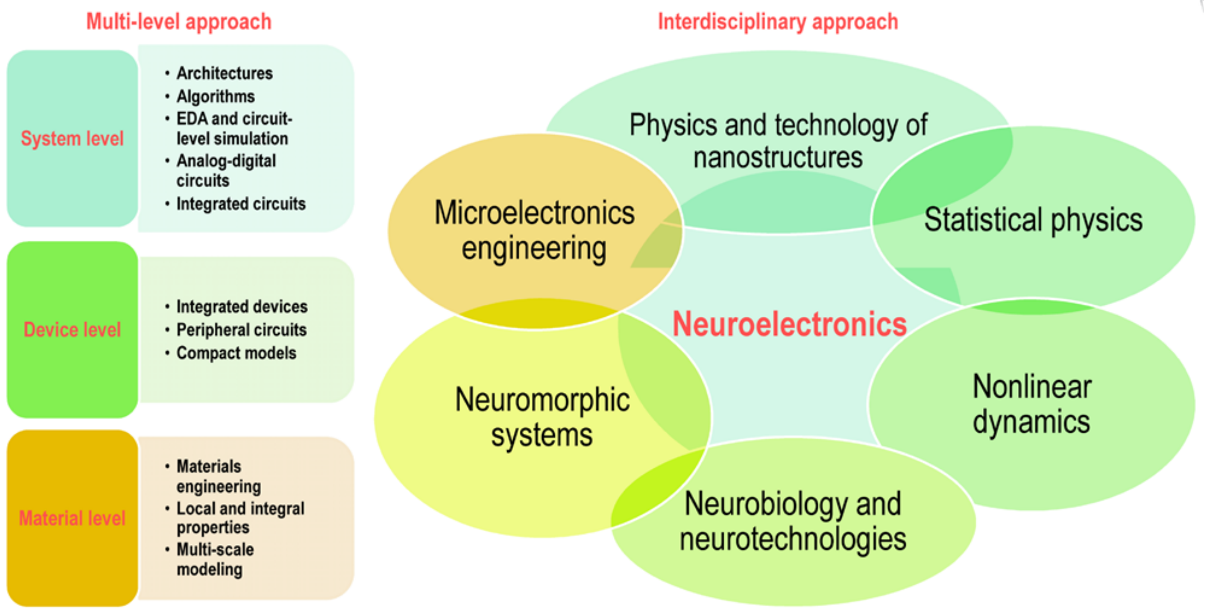


Figure 3. Illustration of complex (multi-level) and interdisciplinary approaches to designing neuromorphic and neurohybrid systems based on memristors

At the material level, MOM nanostructures are fabricated and studied, which exhibit resistive switching (one of the classic mechanisms of the memristive phenomenon). However, for understanding the regularities of memristive phenomenon and controlling its parameters, detailed study of physicochemical phenomena at the nano- or microlevel is insufficient. For example, the combination of different transport phenomena (phonons, electrons, ions) at different time scales makes even one memristor a complex nonlinear system with a rich dynamical response. In order to move further towards neuromorphic and neurohybrid systems, the same developed memristive structures are implemented as integrated devices and chips that are part of various functional circuits at the system level. Experimental work is always carried out in parallel with multiscale modeling: from models of physical phenomena at the micro-, meso-, and macrolevels to compact models of devices and circuit models required for automated design of electronic circuits. At the heart of such an approach lies the cross-cutting technology of memristive devices, compatible with traditional silicon technology and providing the creation of a component base for new brain-like information processing systems with a wide range of applications, including traditional and spiking neural network architectures, and neurointerfaces.

The interdisciplinary nature of the project is also illustrated in Fig. 3. Physics and technology of memristive nanostructures is one of the key areas that, based on traditional and new approaches in microelectronics, creates a technological platform for hardware implementation of memristor-based neuromorphic systems. To interpret, describe, and predict the memristive phenomenon, it is necessary to use the significant scientific knowledge in the fields of statistical physics and nonlinear dynamics. Based on the latest achievements in neurobiology and neurotechnology, the next step towards the symbiosis of artificial electronic and living biological systems can be taken.

To achieve the goal, interrelated tasks should be reached, including: 1) the investigation of new materials and devices, 2) the development of cross-cutting technology of a new component base, and 3) the development and hardware implementation of neural network architectures.

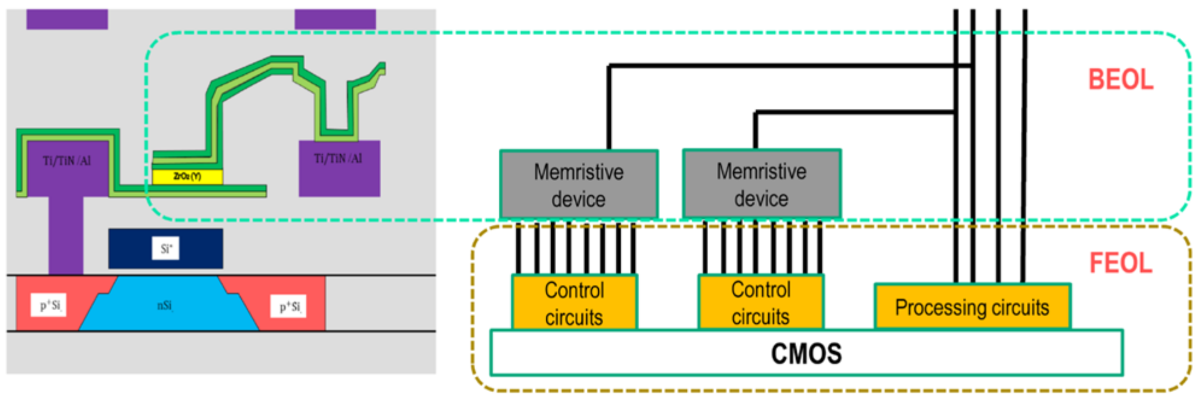


Figure 4. Illustration of memristive nanostructures integrated with CMOS circuitry using the BEOL process

Regarding memristors, reaching the first task is complicated by the fact that the complex nature of memristive phenomenon requires interconnected research at the micro- and macroscopic levels involving physics and chemistry of solid-state nanostructures, nonlinear dynamics, and statistical physics. The development of these interdisciplinary studies results in the discovery of new phenomena and the implementation of new methods to improve the characteristics of electronic devices based on memristive materials. Essentially, reaching this task means resolving fundamental issues associated with the correct description of the memristive phenomenon in various structures and materials and accompanying the design and creation of AI information and computing systems based on new component base.

The development of a cross-cutting technology based on resistive switching devices involves the development of scientific and technological solutions for creating elements and cells of non-volatile RRAM based on memristive nanostuctures with good yield, high endurance and retention parameters. The most important characteristic of memristive devices from the viewpoint of neuromorphic applications is their ability to store information at multiple levels, and significant progress is being made in this area now [39]. The main solution in the development of RRAM technology is the fabrication of functional RRAM blocks based on the integration of memristive structures, which are made at laboratory facilities in top metal layers (back-end-of-line – BEOL process), and the active layer of CMOS (front-end-of-line – FEOL process), which is made in industrial conditions (Fig. 4). Examples of images for the FEOL wafer, its fragment after the completion of the BEOL process, and the ready-made crossbar array of 1T1R (one memristor – one transistor) memristive cells are shown in Fig. 5. In the case of successful implementation, the cross-cutting technology for creating memristive microchips will provide a technological platform for a wide range of products, from RRAM microchips to neurochips, neurointerfaces, and neuroprosthesis for medical applications.

Research and development within this task results in the design and fabrication of test crystals with functional blocks of non-volatile resistive memory (memory cells and RRAM arrays) required to demonstrate the capabilities of new memory devices and basic principles of neuromorphic computing (VMM operations).

The main task within this scientific and technological field is the development of a neuromorphic processor with an array of synaptic weights based on memristors in a crossbar architecture (the most popular active RRAM crossbar is 1T1M). This processor should have digital-analog neurons of the leaky integrate and fire (LIF) type and other configurable parameters, with the

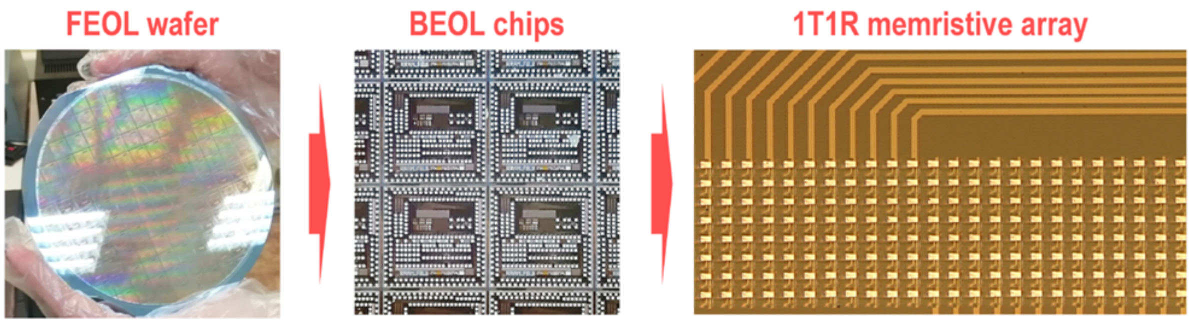


Figure 5. Images of the FEOL wafer, its fragment after completion of the BEOL process, and the final array of 1T1R memristors

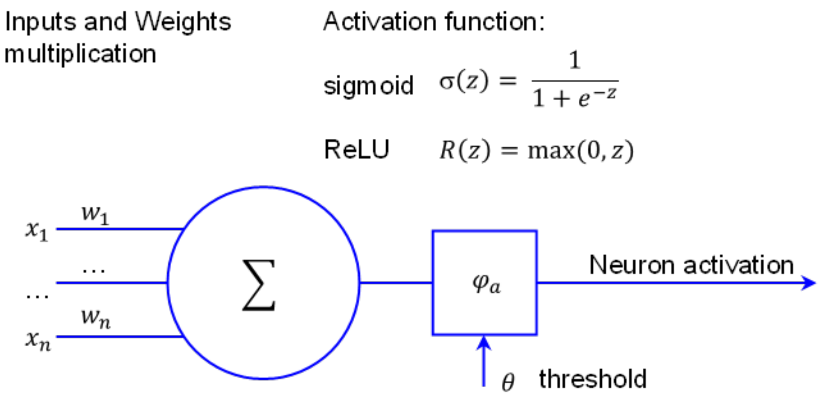


Figure 6. Formal neuron model – the weighted sum of inputs is fed into a basic nonlinear activation function, which can be either a sigmoid or a simpler Rectified Linear Unit (ReLU) transformation

ability to control and rewrite arbitrary memristive cells, supervised and unsupervised learning, including that based on local rules, and working in logical inference modes, as well as algorithms based on formal neural networks and spiking neural networks with spatio-temporal coding of multi-dimensional patterns of the solved problem. In the future, such a neuroprocessor should be able to solve various tasks in the field of AI: recognition of visual images, text and speech processing, analysis of various types of big data, prediction of temporal data series, sensorimotor control of mobile objects, optimization control of data flows in real-time, etc.

Let us take a closer look at the general approach to building an artificial neural network, which is based on a neuron model. There are two types of neuron models: formal (Fig. 6) and spiking one (Fig. 7) [34]. The main difference lies in the way the processed signals are represented: in a formal neuron, these signals have a continuous form, while in a spiking neuron, they are pulse-based. On the one hand, the hardware implementation of spiking neurons has an advantage of several orders of magnitude in terms of energy efficiency, but, on the other hand, the sharp fronts of the pulse signal make differentiation difficult, and as a result, the widely used backpropagation method fails when training a neural network. This situation leads to the necessity of developing new training algorithms for spiking neural networks based on bioplausible local plasticity rules [16].

The formal model of a neuron is widely used in various types of modern coprocessors, such as digital signal processors (DSP, digital signal processing), graphic processing unit (GPU), numerous neural accelerators and tensor accelerators (Google TPU (Google company), IVA

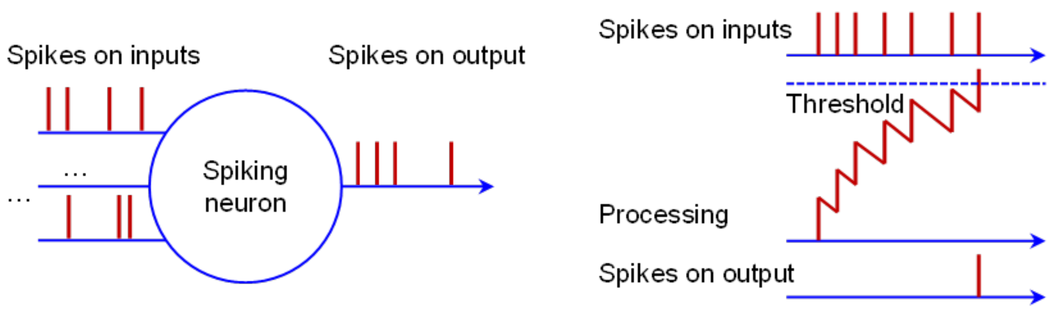


Figure 7. Spiking neuron receives sequences of spikes on its inputs and, under certain conditions, generates a spike at its output; for example, in the LIF model, each spike contributes to the neuron's status – its amplitude, which decays over time; if a sufficient number of spikes contributes to the status in a certain time window, the neuron amplitude exceeds a threshold, and the neuron generates an output spike. The electrical model of such a neuron can be implemented using an operational amplifier (OA) with an integrating RC circuit in the inverting input arm and a comparator

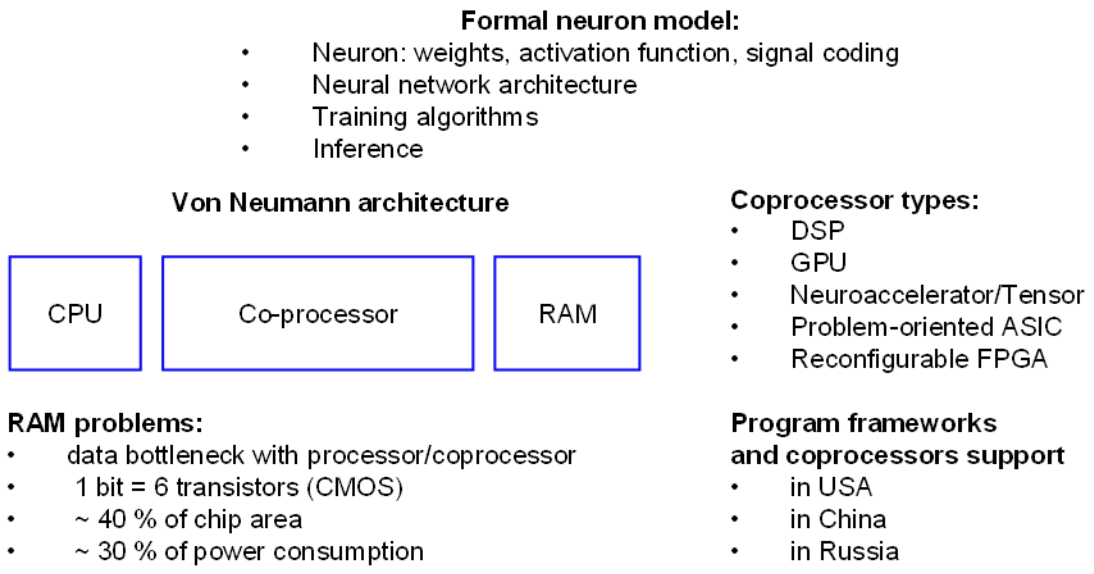


Figure 8. Software-hardware ecosystem for implementing neural networks on the formal neuron model: a significant foundation has been created, and best practices can be used for rapid development and testing of innovative neuromorphic systems

TPU (IVA Technologies company), NM6408 (Scientific and Technical Center "Module"), RoboDeus (Research and Development Center "ELVEES"), and many others), application-specific integrated circuit (ASIC), and field-programmable gate arrays (FPGA). Frameworks have been developed and widely used as software environments for developing neural networks, training them, and performing inference using the aforementioned processors. Thus, full hardware and software ecosystem has been developed for processing neural networks using the formal model of a neuron (Fig. 8). The further development of the formal model continues in the direction of improving processing algorithms and reducing the technology nodes of CMOS processors [23].

This background can be partially used for the hardware and software of new processors based on neuromorphic architectures and on a component base of new physical principles. Currently, the best neuromorphic model is based on the spiking model, but over time, neurophysiologists will discover and justify a more realistic model of neuron operation. At the moment, digital

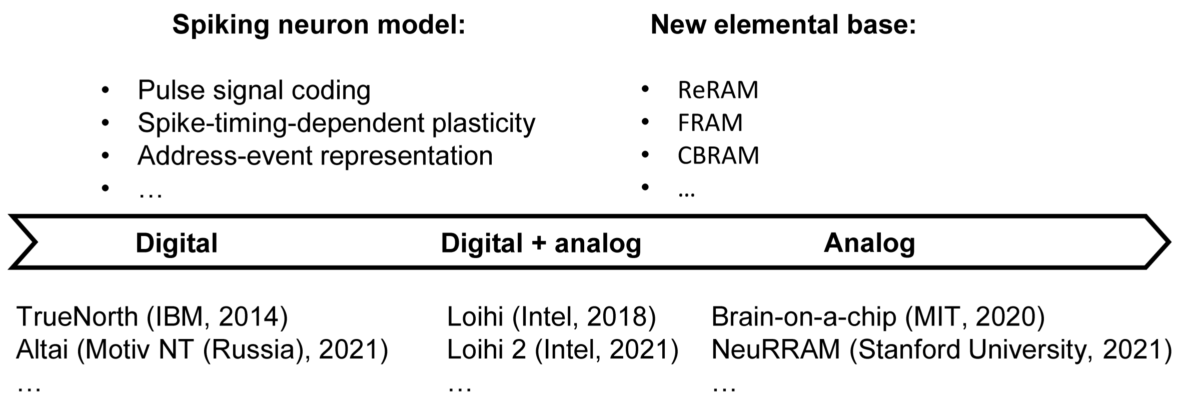


Figure 9. Modern neuromorphic systems based on the spiking neuron model

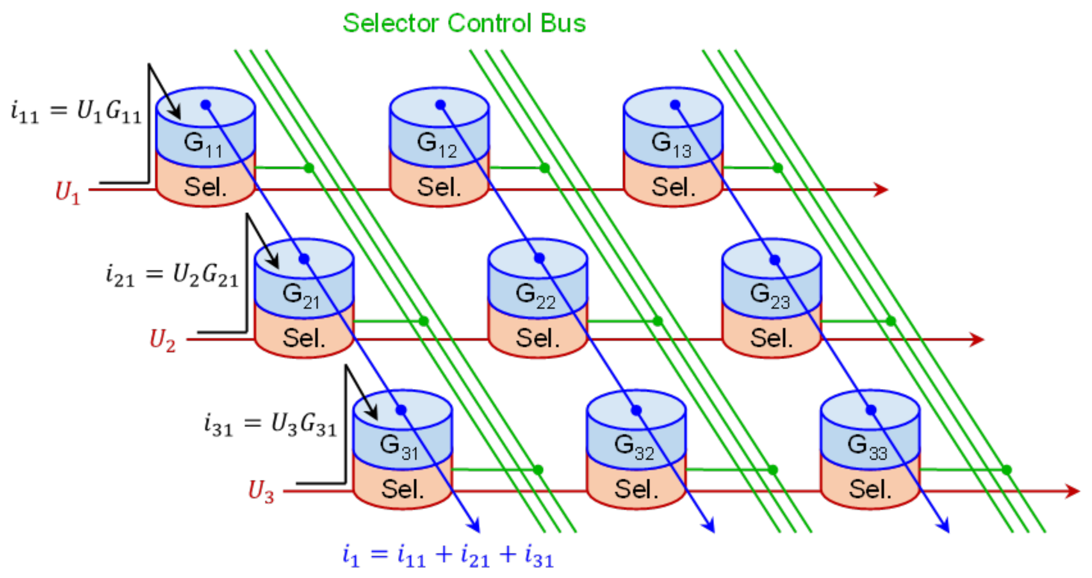


Figure 10. Memristive crossbar used in the VMM (inference) mode: input voltages are multiplied by the conductance G of the corresponding memristors in a certain column, and the resulting currents are summed up in the column. Selectors provide the connection of memristors to the crossbar lines. In some cases, selectors provide reverse connections, where inputs are swapped with outputs (blue lines receive voltages, and red lines extract currents) for the training process

(IBM, Motif NT), digital-analog (Intel), and analog (MIT and others) neuroprocessors have been developed on the spiking model. Naturally, there are many other developments not indicated in Fig. 9. The analog implementation of neurons is based on the use of operational amplifiers (OA) allowing a number of mathematical operations to be performed using currents and voltages in an electrical circuit.

The main operation carried out in neural network computation is VMM. As noted above, VMM is naturally and in analog form implemented in a memristive crossbar, which consists of a set of parallel metal lines in one plane and another set of parallel lines oriented perpendicular in another parallel plane. Memristors with programmable (self-adapting based on local rules) conductance values are placed at the crossbar nodes along with selectors – elements that provide correct addressing when accessing memristors (Fig. 10) [4].

On the one hand, analog representation and processing of information without the clocking characteristic of modern von Neumann architecture processors and coprocessors provides maximum speed and eliminates pipeline delays when obtaining results. On the other hand, digital

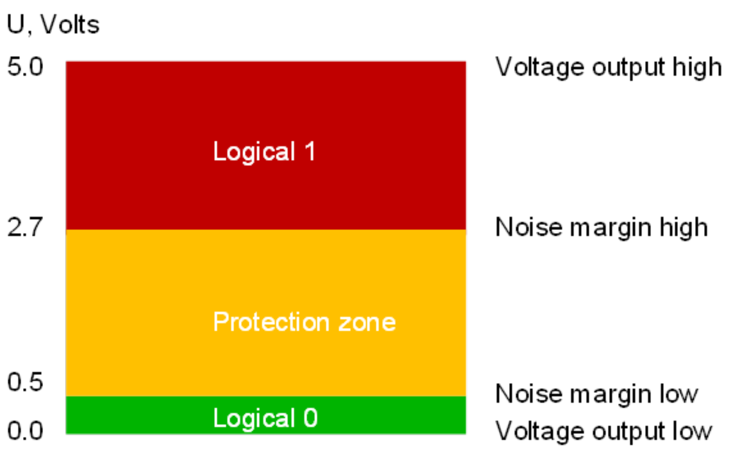


Figure 11. Levels of logical zero and one for a supply voltage range of 5V, ensuring high noise immunity for processed signals. The signal ranges V_O (output of the signal source) for the logical zero and one are wider than their corresponding ranges V_I (input of the signal receiver), compensating for possible voltage fluctuations during signal transmission through the interconnection lines between logical elements, signal source, and receiver

representation of information in the form of logical zeros and ones provides a high level of noise immunity due to the fact that the entire range of power supply voltage is divided into 3 zones (Fig. 11), the middle of which is not used and minimizes the number of possible errors. The use of analog, continuous amplitude scales for processing signals automatically imposes limitations on the dimensionality of the crossbar.

Memristive crossbars are the basis for hardware analog execution of mathematical operations inherent in various architectures of neuromorphic devices. Specifically, they allow for the execution of the VMM, which occupies the majority of data processing time in neuromorphic systems (inference), in parallel for several neurons in a single clock processor cycle with very low (picojoule) energy consumption. However, the potential for high performance and low energy consumption is not automatically realized – the computations in memristive crossbar-based systems must be organized in the most optimal way. Analogous to von Neumann architecture, the task of signal switching and control of a computing device can become a “bottleneck” in neuromorphic systems if not handled correctly.

A characteristic feature of neuromorphic systems is that, similar to biological networks of neurons, they contain a large number of interconnected nodes performing the same operations on the information being processed. For practical applications, the number of nodes (neurons) can be measured in thousands and the number of connections (synapses) – in millions. Individual memristive crossbars, having a specific number of memristive devices determined by the topology of the crystal and existing technological constraints, physically implement only a portion of the connections between neurons in different layers, with several crossbars possibly related to the same neurons. In these conditions, the developed architectures must be scalable.

The scalability of neuromorphic systems based on memristors logically should be implemented both at the neural model architecture level – “horizontally” (to provide the necessary number of neuron layers) and “vertically” (to provide the necessary number of neuron inputs), as well as at the level of parallel processing of data flows by multiple neuromorphic models with the same architecture. Moreover, physically, such scaling also has several levels – increasing the number of crossbars in a single neuroprocessor, increasing the number of neuroprocessors,

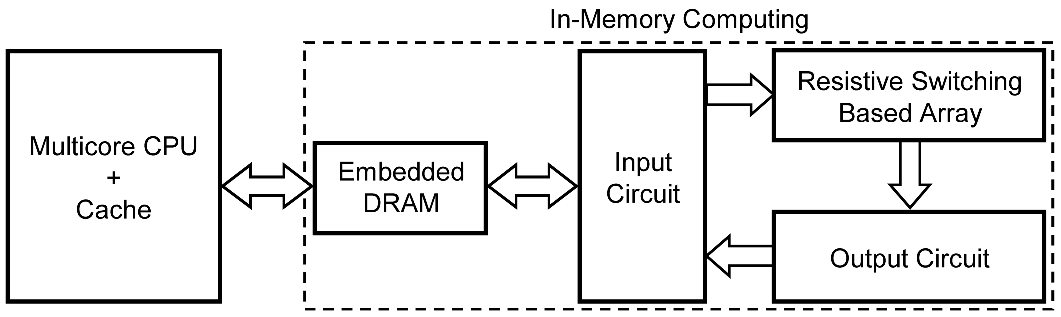


Figure 12. A separate in-memory computing chip based on the memristor-based array equipped with built-in dynamic RAM. Input and output circuits provide binary signal conversion into voltage and the reverse conversion of resulting currents into voltage. Computations are controlled by a multi-core processor device with cache memory implemented on another chip

and combining them into a cluster, and so on. The basic requirement for each level of scaling is to maintain a high level of parallelism in signal commutation for their simultaneous delivery to an equivalent crossbar (single crossbar or several crossbars combined "horizontally" and "vertically") and control of keys and selectors.

3. Approaches to Scaling Up Memristor-Based Neuromorphic Computing Systems

Let us consider various options for scaling active memristive crossbars in a CMOS-integrated form to increase the speed of signal transmission in memristive neural networks with static and spike coding [48].

In the classical von Neumann architecture, separate devices are used for data storage (random access memory, RAM) and computation (arithmetic logic unit, ALU). The operation principle of the slow dynamic memory DRAM limits the speed of reading/writing information of both initial and resulting data of the computational process. Therefore, when computing in memory, a separate chip is equipped with its own memory and computational cirtuit, which is controlled by the central processing unit (CPU) chip – Fig. 12 [4].

The computational process is organized as follows. The processor updates the weight coefficients in the memristive crossbar as needed, loads the input matrix into the embedded eDRAM memory in the chip for in-memory calculations, and issues the command to start the calculations. Data from eDRAM are transferred to the input circuit and converted into voltages required to operate the memristive crossbar. Each column of the memristive crossbar sums the products of input voltage and memristor conductivity in the form of current, performing an analog implementation of multiplication with accumulation in memory. In the output circuit, the results are converted into an output resulting matrix and stored in eDRAM for further use by the processor in the computing process.

The input and output circuits servicing the operation of the memristive crossbar are implemented using digital circuits with the use of analog-to-digital and digital-to-analog converters (ADCs and DACs) designed using CMOS technology – Fig. 13 [4].

The simplest binary neural networks require a relatively small percentage of CMOS processing circuits in the overall hardware implementation taking into account the memristive crossbar. The current level of development in the design and technology of memristive devices reflects the availability of devices with two levels of information storage. Binary networks are very energy-

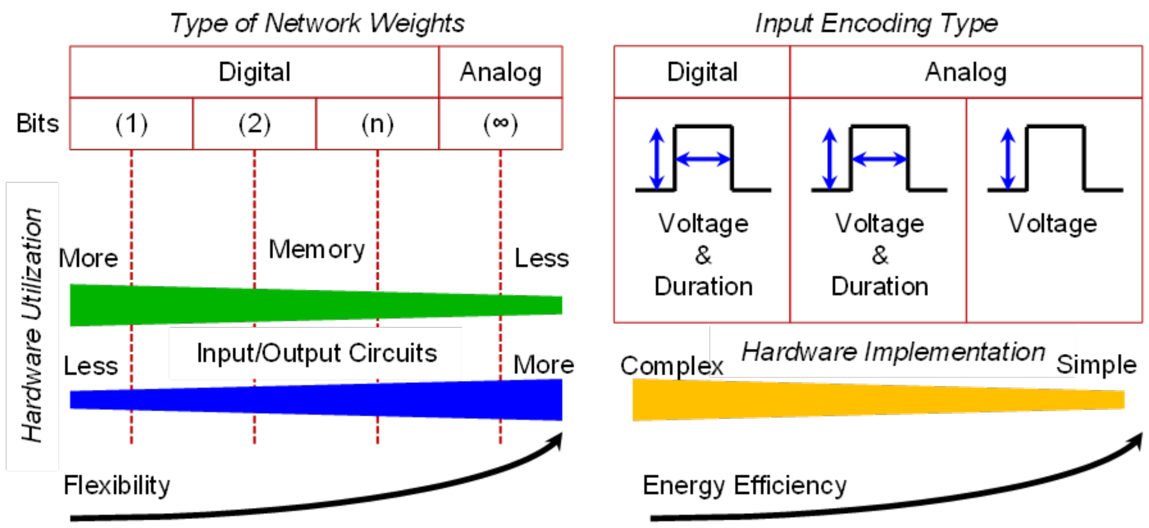


Figure 13. Flexibility and energy efficiency are maximized with analog signal processing. The hardware implementation of post-processing of the input signal (voltage) is also simplified for the transition from digital to analog representation

efficient but capable of solving relatively simple tasks, such as pre-processing and processing of sound and speech.

Moving to three or more binary digits on one side opens the possibility of using more complex neural networks, but also means an increase in the share of CMOS circuits in the overall hardware implementation. New technologies of such multi-level memristors are actively being developed [39].

The use of unlimited (analog) precision of weights requires the use of corresponding memristors, which are not widely available at present, as well as a significant volume of high-precision CMOS component base to provide digital-to-analog and analog-to-digital support for memristive crossbars. The undeniable advantage of neural networks in such implementation is the high degree of accuracy achieved during their operation due to the absence of the need to reduce the bit depth of weight coefficients during the conversion of the model into hardware implementation.

Each column is implemented in the simplest analog encoding circuit with amplitude encoding of the input signal and a 0T1R memristor cell in the crossbar node operated by an OA with a feedback resistor (Fig. 13, analog voltage encoding). The addition of duration to the input signal requires an integrating function in the OA (Fig. 13, analog voltage and duration encoding). To process signals in a full memristive crossbar with 1T1R cells and digitized amplitude and sampled duration of the input signal, the most complex CMOS circuit will be required, using a comparator and counter (Fig. 13, digital voltage and duration encoding).

The activation function is also implemented in circuits using OA (Fig. 14). A circuit containing 2 OA and a set of resistors serves one column of a memristive crossbar [28].

In well-designed CMOS circuits for memristive crossbars, the limiting factor for increasing their dimensionality is the presence of parasitic sneak paths in these crossbars. The problem is that current, in addition to the desired propagation path of row-column, also flows through adjacent undesirable paths. In [64], an analysis of this problem was carried out: the ratio of the voltage range in the crossbar to the voltage range in one memristor was calculated depending on the stored values in the crossbar and the grounding of rows and columns. The presence of parasitic paths depends significantly on the stored values in the memristive crossbar. The dependence of the parameter Δ', which is equal to the ratio of the power supply voltage and the

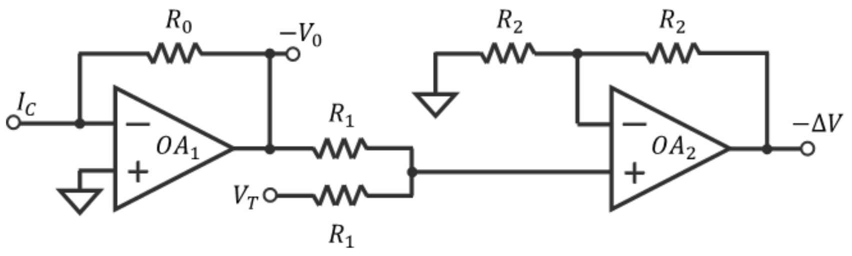


Figure 14. An example of implementing an activation function on an OA for a fully connected layer of a neural network when the total column current I_c is received: V_0 is the activated output, V_T is the target value for V_0, ΔV is the mismatch error between V_0 and V_T

zero voltage (ground) difference for the entire crossbar to the same difference for one memristor, was investigated. In the ideal case, the result is equal to one, in others – less than one. The simulation results show that a significant decrease in the analyzed parameter is observed even for relatively small dimensions of a 16×16 and 64×64 arrays.

To address the issue of parasitic sneak paths, several methods are being considered. The first method is called multi-stage reading and includes five steps: measuring the target cell current, setting the target cell to high-resistance state (HRS) and measuring the current, performing a similar operation for low-resistance state (LRS), comparing the measured currents, and returning the cell to its original state. The second method involves column separation architecture for each memristor. The third and fourth methods involve using a diode and a transistor as a selector (1D1R and 1T1R cells, respectively). The fifth method involves using complementary memristors that provide constant resistance $R_{\text{LRS}}+R_{\text{HRS}}$, significantly reducing parasitic currents. Although the 1T1R cell takes up more space and an additional line is required to control the transistor gate, this method is the most common.

In various crossbar circuits, duplication of elements is used to achieve the required functionality and increase performance, as well as multiplexing the component base for its subsequent reuse to perform various functions with time division. Thus, the HRS and LRS in binary ReRAMs are positive, so the XNOR operation is used for encoding signed weights, and the number of rows of memristive crossbars is doubled (Fig. 15) [44]. In Fig. 15, SL is the source line, BL is the bit line, WL is the word line: lines of source, bits and words; the input signal value "–1" is encoded by a pair of 1 and 0, the value "+1" – by a pair of 0 and 1; the weight "–1" is encoded by a pair of LRS and HRS, the weight "+1" – by a pair of HRS and LRS; eight bit lines are collected into a processing block; the next bit line is selected on the multiplexer and its value (128 levels) is digitized with the help of instrumentation amplifiers. High precision instrument amplifiers operating in voltage mode are used to process signals of the memristive array columns, which are separated by a multiplexer for processing the signal of one of the eight columns (bit lines). In the first case, there is a duplication of the hardware, in the second case, there are savings of the CMOS base by increasing the signal processing time. Various circuits of adaptive compensation of large or small current values in polled memristors are used, for example, the voltage clamp control circuit [59].

The active development of circuits on memristive crossbars is accompanied by the increase in the dimensionality of crossbars on one hand, and by proposals to map neural networks to the hardware implementation of such processors on the other. For example, in [51], the NeuRRAM processor with a multilevel organization of processor units is proposed. At the top level, the implemented hardware neural network is mapped onto such a processor consisting of 48 cores

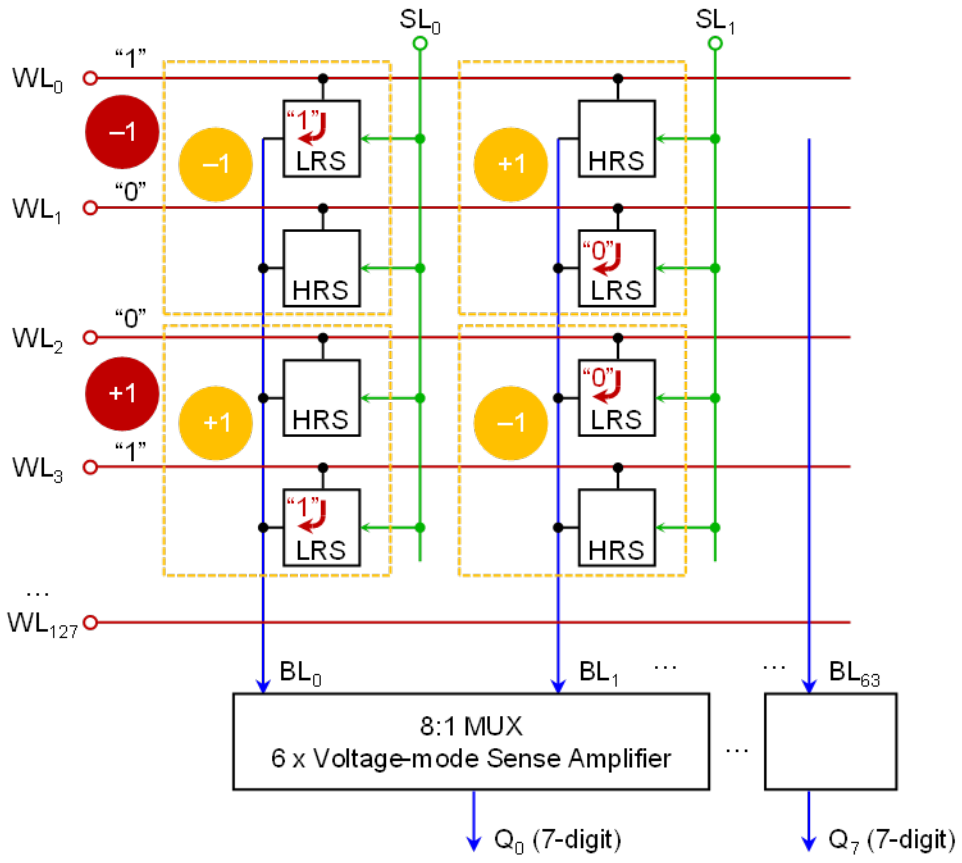


Figure 15. Example of implementing a memristive array with an effective size of 64 × 64

organized in an array of 8 rows by 6 columns (Fig. 16). To operate the neural network, it is mapped onto 48 cores of a chip in one of 6 ways: (1) 1 layer in 1 core, (2) duplication in multiple cores to increase throughput, (3) multiple layers in one core, (4) reordering in one core to increase utilization, (5) and (6) parallelization on multiple cores. Each core consists of an array of 16 × 16 corelets, each of which contains a 16 × 16 RRAM weights and a CMOS neuron. The single-bit BL and SL switches of the corelet can change the direction of the signal being processed by the CMOS neuron from BL to SL or vice versa. This configuration is called Bidirectional transposable neurosynaptic array (TNSA), meaning that the input signals can be fed to both rows and columns with the help of supporting CMOS circuits. At the stage of VMM input, the drivers convert the register inputs (REG) and PRN inputs into analog voltages and transmit them to TNSA. At the stage of VMM output, the drivers transmit digital outputs from neurons back to registers through REG. In addition, various activation functions, including stochastic ones, are implemented in the CMOS circuits.

At the lower level, the corelet consists of a 16 × 16 array of memristors and one CMOS neuron. The neuron is connected to one of 16 bit lines and one of 16 source select lines that pass through the corelet. It is responsible for integrating inputs from all 256 RRAMs connected to a single BL or SL: 16 RRAMs in the current corelet and 240 RRAMs in other corelets along the same row / column. Thanks to an advanced routing system, each core is capable of performing forward, backward, and recurrent VMM on all 256 rows.

The above-mentioned memristive crossbars with CMOS control circuits are implemented as monolithic microchips with 90 and 130 nm technology nodes. As noted above, the CMOS control circuits are located in the FEOL layer, while the memristive crossbar is located between

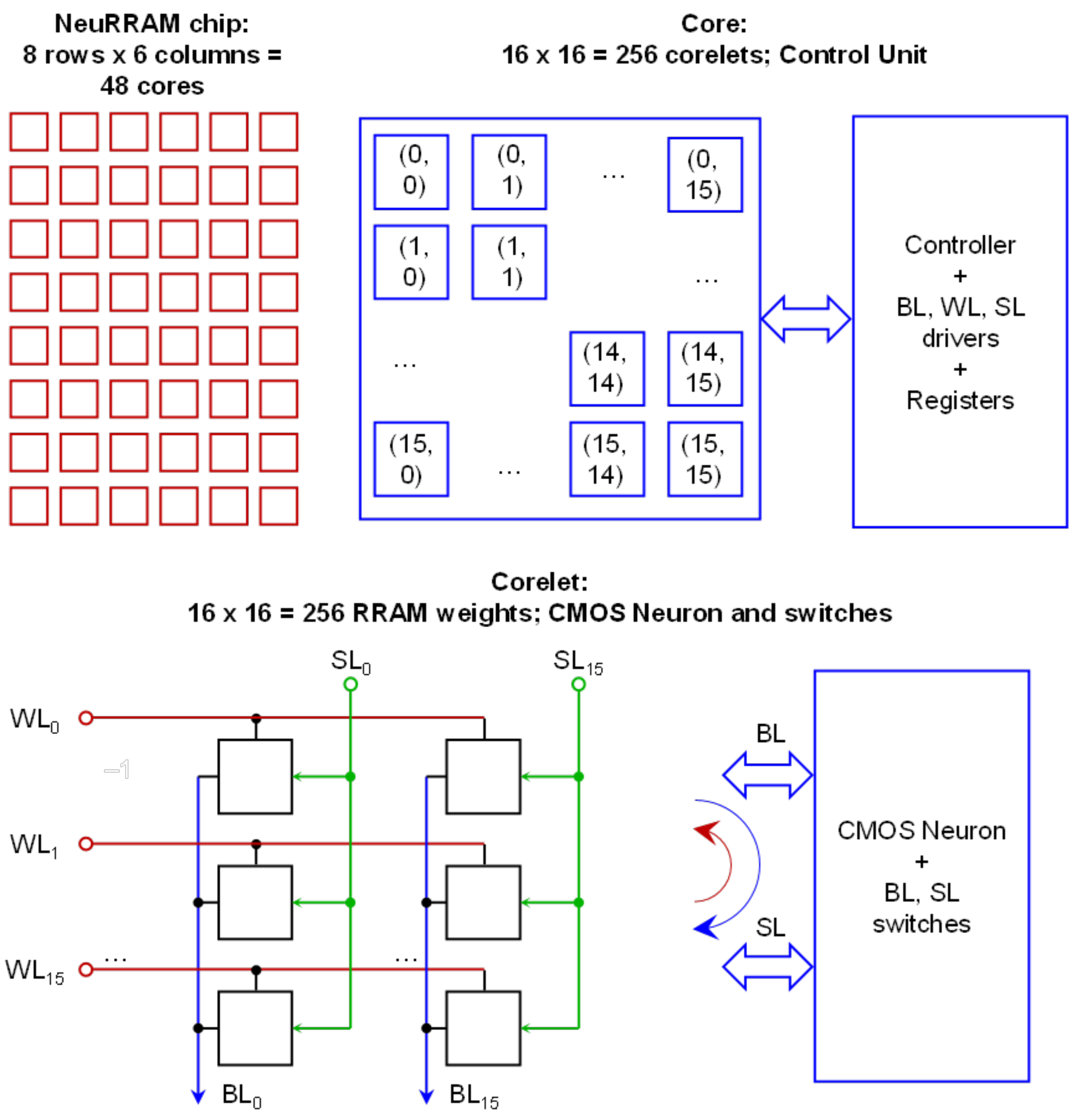


Figure 16. Architecture of the NeuRRAM project [51]

the metallization layers in the BEOL layer or on top of it (Fig. 4). However, there is another relatively new approach to implementing complex devices, including cases where their parts are made using different and possibly incompatible technologies. In the above example, the oxide layer in the memristor may be destroyed by the high temperature during the formation of the upper layers using CMOS technology – exceeding the temperature budget [63].

The idea of dividing a large chip by area into a set of separate chiplets (mini-chips) with their subsequent placement and side-by-side connection on the substrate-interposer plane (2.5D integration) or in the form of a stack (stepped structure, 3D integration) with connection by vertical conductors (TSV, through silicon via) originated in 2015 [25]. Each chiplet is usually a system module, implemented using incompatible technologies or implementing a complex functional block (IP, intellectual property). Pascal Vivet (LETI – Laboratory of Electronics and Information Technologies, European center for research in microelectronics) believes that “Chiplet-based ecosystems will deploy rapidly in high-performance computing and various other market segments, such as *embedded* HPC for the automotive and other sectors” [25]. LETI

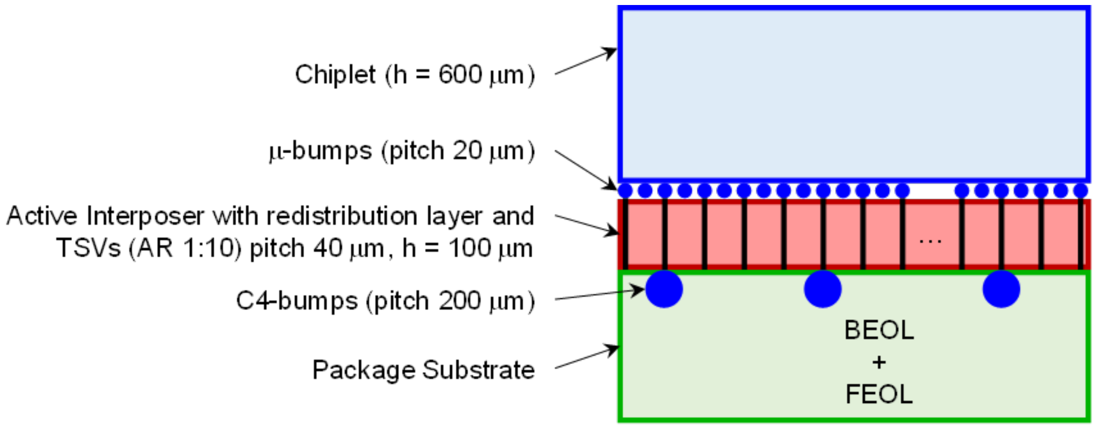


Figure 17. Active interposer presented by the LETI center [25] to combine 96 cores on 6 chiplets. The active interposer with RDL (redistribution layer) allows combining the interposer with a bump pitch of 200 μm (at the bottom) and micro-bump pitch of 20 μm (at the top of the chiplet)

presented an active interposer technology for chiplets, which was used to assemble a structure consisting of 6 chiplets with a total of 96 cores (Fig. 17).

The issues of chiplet assembly, testing, and yield, as well as CAD support, are not yet adequately addressed in the technology of chiplets. However, extensive work is being done to standardize interchip communication technologies, such as Intel's Advanced Interface Bus (AIB), the Optical Internetworking Forum's CEI-112G-XSR, and Open Domain-Specific Architecture's BoW (Bunch of Wires) and OpenHBI (High Bandwidth Interface).

The seriousness of chiplet technology is confirmed by the participation of well-known companies like Boeing, Cadence, Synopsys, Intel, Micron, and others in the project Common Heterogeneous Integration and IP Reuse Strategies (CHIPS, a program for integrating heterogeneous chips and reusing complex functional blocks since 2017), as well as GE, Intel, Keysight, Xilinx, and others in the project The State of The Art (SOTA) Heterogeneous Integrated Packaging (SHIP, an advanced program for packaging heterogeneous chips – to establish interface standards between chiplets and ensure the assembly of complex functional blocks since 2019). Both projects are being implemented by the American agency DARPA.

An actual example of using such technology for in-memory computing on memristive crossbars is the SIAM project – Chiplet-based Scalable In-Memory Acceleration with Mesh for Deep Neural Networks [24], a chiplet-based scalable in-memory computing accelerator for deep neural networks (Fig. 18). At the initial stage, a transition from a neural network to an architecture is made, taking into account: the IMC (In-Memory Computing) chip mode, the network frequency in the package (NoP), the size and number of chiplets, IMC mapping, the number of tiles per chiplet, the size of the crossbar, memory cell type, technology node, and accumulator size; the "engine" for partitioning and mapping: internal chiplet planning, chiplet placement, "engines" for NoP and DRAM; mapping to IMC tiles, external chiplet planning, routing and placement, "engine" for electrical circuit and chip network (NoC, network on chiplet); obtaining a chip partitioning as shown in Fig. 18 on the left.

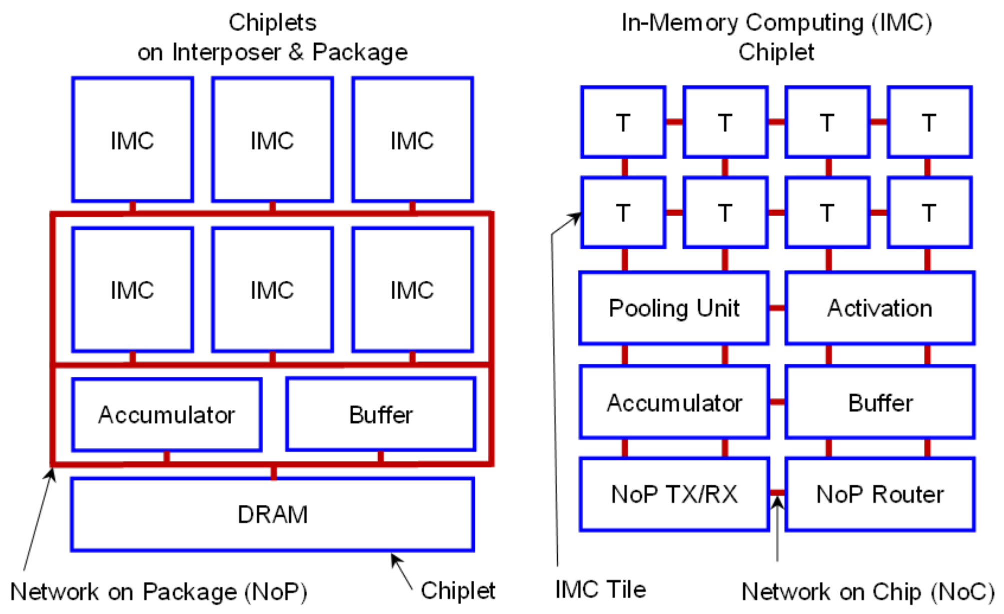


Figure 18. SIAM project [24]: implementing computational functions through chiplets, global accumulator and buffer, DRAM memory placed and connected on the interposer in a package and connected to the NoP (on the left). Each IMC chiplet consists of IMC tiles, calculation modules, communication and routing (on the right). Each tile consists of multiple processing elements (PE), a multiplexer, ADC, instrumentation OA, shift and adder device, buffer; each PE contains a memristive crossbar (not shown)

4. Comparison of Computational Systems Based on Traditional and New Component Base

Let us take a closer look at the results of comparing known GPU and neuromorphic processors based on traditional digital components with prototypes of memristor-based neuromorphic processors. For comparison, we will use two absolute criteria – the number of cells and peak performance (gigaoperations per second, GOPs), and two relative criteria – performance per chip area (GOPs/mm^2) and performance per watt of energy consumption (energy efficiency, GOPs/W). These criteria are calculated for the inference of neural networks, where the basic operation is a VMM. The comparison results are shown in Fig. 19.

For this comparison, specialized neuromorphic processors Altai [1] and Tianjic [36], optimized for spiking neural networks, and the most powerful GPU from NVIDIA – Tesla V100 [2], which is more universal than the previous ones, as it allows solving a wide range of tasks in the field of data processing, were chosen. The performance metrics were taken from open sources (references at the horizontal axis) as indicated by the authors. All prototypes of memristor-based processors selected for comparison are made using CMOS-compatible technology and have a device layer with transistor selectors (except [7]) and other electronics required for operation.

As seen in Fig. 19a, computing systems based on memristive devices have significantly fewer cells than existing processors. However, this is not a disadvantage and is explained by the fact that the presented developments are still prototypes created as a result of research and development. Nevertheless, even such relatively small processors, with up to 4 million cells, demonstrate sufficiently high performance, surpassing Altai and Tianjic processors with 67 and 10 million synapses, respectively (see Fig. 19b).

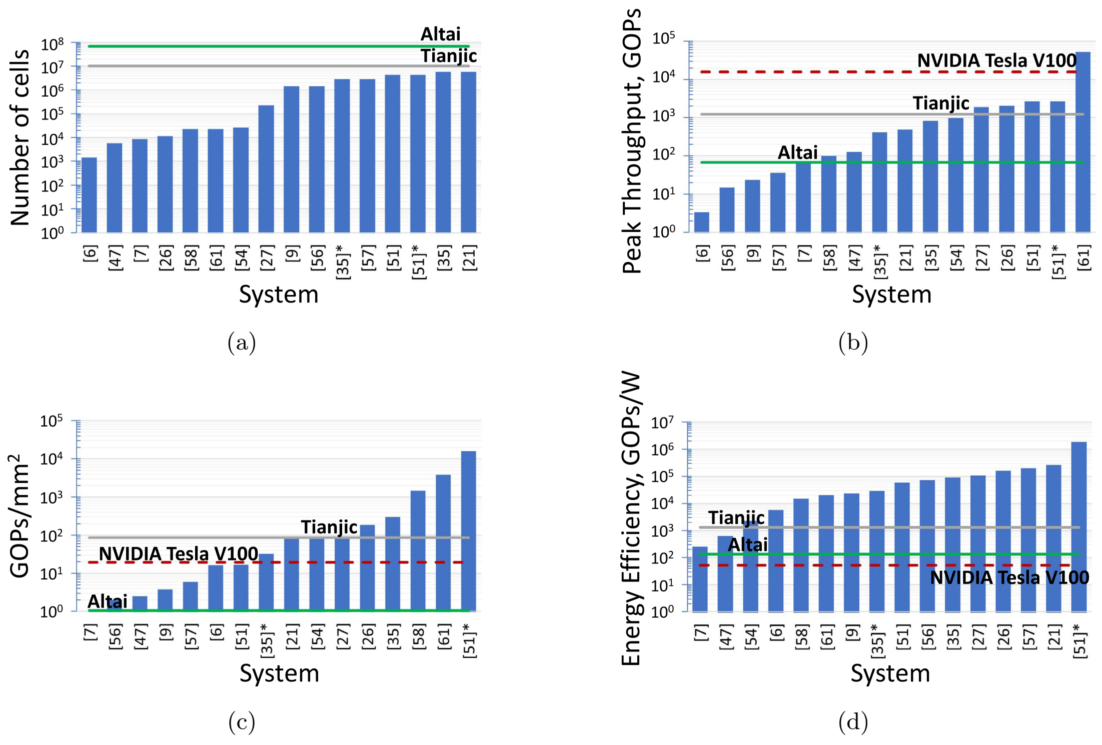


Figure 19. Comparison results of memristor-based computing systems (bar chart columns) with neuromorphic processors (horizontal solid lines) and GPU (horizontal dashed line) based on traditional digital components according to the following criteria: the number of cells (a), peak performance (b), performance per chip area (c) and performance per watt of energy consumption (d)

The advantages of computing systems based on memristive devices are most clearly demonstrated when compared according to relative criteria. The high potential for miniaturization of memristive devices (down to a few nanometers) and RRAM cells (requiring only 1–2 transistors) allows for more efficient use of chip space, as shown in Fig. 19c. For example, the RAND chip (Resistive Analog Neuro Device [35]) made using 40 nm technology has an area of 2.71 mm^2 at a density of 1.48M synapses per mm^2 with drivers, controllers, and multiplexers while providing three times higher relative performance than the Tianjic processor and 12.6 times higher performance than NVIDIA Tesla V100. In turn, the energy efficiency of memristor-based computing systems is 2–3 orders of magnitude better than existing processors (see Fig. 19d). For example, the nvCIM macro chip [21] made using 22 nm technology node demonstrates 12–150 times lower power consumption than Tianjic and 300–3700 times lower consumption than NVIDIA Tesla V100.

With the advancement of technology in creating memristor-based neural processors, the number of cells will increase, meaning that with higher computing density, peak performance will exceed the performance parameters of neuromorphic processors based on traditional digital electronics and specialized architectures presented in Fig. 19b. Of course, this growth cannot be indefinitely large, and potential high performance and energy efficiency will be more influenced by design solutions at the processor and computing system architecture levels, especially the growing overhead costs of routing and input/output data in digital form (see also Section 3). For example, when it comes to processing signals of different nature, performance will be limited by the characteristics of sensors and information transmission interfaces, so devices for computing in sensors with direct transmission of information in analog form to a memristor-based computing device are currently being developed for such tasks [31, 49].

A number of other notable examples of memristor-based computing systems were not included in this comparison, as authors in publications often do not provide the values of the criteria used in Fig. 19. In addition to these, there are more specialized criteria for assessing performance and energy efficiency in relation to the peculiarities of the processor architecture or the specific problem being solved. These criteria include the number of synaptic giga- or teraoperations per second (GSOPs, TSOPs) [3, 36], the number of giga- or teraoperations per second computed per 1 Mb of ReRAM (GOPs/Mb, TOPs/Mb) [51], AiMC TOPS/W [8], the number of processed frames per watt (frames/W) [61], and energy-delay-product (EDP, j·s) [51]. Furthermore, some authors use software simulators (such as XPEsim [58]) to evaluate the performance and energy efficiency characteristics of ReRAM-based chips due to the high cost of prototyping. In the future, a valuable criterion for comparing in-memory computing systems will be the cost of 1 k/M/G byte of memory.

Neuromorphic computing accelerators (standard digital ASICs based on CMOS, system solutions, and memristor-based microchips) presented in Fig. 19 were compared for performance and energy efficiency taking into account their high (comparable to software emulation) accuracy in inference of neural network models for specific tasks such as pattern recognition, classification, segmentation, etc. Table 1 shows the numerical values of characteristics of memristor-based computing systems, including the task, neural network model architecture and achieved accuracy metrics.

From Tab. 1, it can be seen that the considered processors perform at a high level on commonly accepted test tasks for image classification from the MNIST dataset with an accuracy range of 90.8 [35] to 99 % [51], CIFAR-10 – from 85.7 [51] to 95.19 % [21], CIFAR-100 – 65.71 % [57], recognize Google voice commands with an 84.7 % probability [51], and successfully solve other tasks whilst implementing well-known neural network architectures such as MLP (multilayer perceptron), DNN (deep neural network), CNN (convolutional neural network), LSTM (long short-term memory) and ResNet-20, ResNet-50, VGG16 models.

It should be noted that, among the considered prototypes, the most versatile in terms of the ability to launch different architectures of neural networks is the NeuRRAM chip [51]. As can be seen from Fig. 19 and Tab. 1, NeuRRAM already has 33–800 times better energy efficiency at technology node of 130 nm than Tianjic, Altai, and NVIDIA Tesla V100 processors, and provides high relative performance compared to them. At the same time, a many orders of magnitude gain in the mentioned and other parameters is expected when scaling the technology node to 7 nm from the current level of 90–130 nm, which are currently used in creating prototypes of multi-core processors based on memristive devices in the structure of MOM.

Thus, in-memory computing is currently the only way to increase the performance and reduce the energy consumption of AI computing systems, as it is the most bioplausible information processing principle from a functional point of view, and it allows for a significant reduction in data transfer distance and required memory volume (model parameters are constantly stored in the processor), as well as energy consumption required for VMM. For in-memory computing, different types of memory can be used [42]: SRAM, DRAM, Flash, however, the most suitable one is RRAM, as other types of memories have disadvantages (such as low scalability, high cost and volatility for SRAM, poor process compatibility with CMOS for processors and the need for regeneration tens of times per second for DRAM, difficulties in implementing write at arbitrary address for Flash, etc.) and impose significant limitations on the creation of neuromorphic chips.

Table 1. Numerical characteristics of memristor-based computing systems

Ref.	CMOS techn.	Unit cell	Cell numb.	Crossbar size	Peak Throughput (GOPS)	Energy Efficiency (GOPS/W)	Area Efficiency (GOPS/mm^2)	Accuracy on Applications Demonstrated
[51]*	7 nm	1T1R	3M	16×16	2,135	1,360,000	12,800	99 % MNIST, 85.7 % CIFAR-10, 84.7 % Google speech
[21]	22 nm	1T1R	4M	1024×512	394	194,000	65.7	92.01–95.19 % CIFAR-10
[35]	40 nm	1T1R	4M	n/a	660	66,500	240	90.8 % MNIST (MLP)
[51]	130 nm	1T1R	3M	16×16	2,135	43,000	13.4	see row [51]*
[27]	130 nm	2T2R	159k	n/a	1,500	78,400	71	94.4 % MNIST (MLP)
[61]	130 nm	1T1R	16k	128×16	41,900	14,900	3,100	40.21 dB PSNR and 22.38 dB SNR for MRI and CT images
[26]	2 μm	1T1R	8k	128×64	1,640	119,700	150	n/a
[57]	22 nm	1T1R	2M	512×512	29	146,000	4.8	90.88 % CIFAR-10 (ResNet-20), 65.71 % CIFAR-100 (ResNet-20)
[35]*	180 nm	1T1R	2M	n/a	330	21,000	26	see row [35]
[54]	130 nm	1T1R	18k	256×16	780	1,650	69	n/a
[58]	130 nm	1T1R	16k	128×16	81	11,000	1,160	96.92 % MNIST (CNN)
[56]	55 nm	1T1R	1M	512×256	12	53,170	1.6	88.52 % CIFAR-10 (CNN)
[9]	65 nm	1T1R	1M	512×256	19	16,950	3	98.8 % MNIST (LeNet DNN)
[47]	150 nm	1T1R	4k	32×32	101	462	2	n/a
[6]	130 nm	2T2R	1k	32×32	2.7	4,200	13	98.4 % MNIST (MLP), 87 % CIFAR-10 (CNN)
[7]	180 nm	0T1R	6k	54×108	57	187.6	0.9	94.6 % breast cancer screening dataset

Conclusion

Memristors are very simple devices and at the same time very smart and complex nonlinear systems promising a wide range of applications from memory chips and in-memory neuromorphic computing systems to adaptive neural interfaces. The implementation of neuromorphic computing systems based on this new component base requires coordinated and interdisciplinary research and development at various levels. The basis of the corresponding scientific and technological direction is the cross-cutting technology of memristive devices and circuits, providing for the creation of a new brain-like information and computing system base with a wide range of applications. The currently demonstrated perspectives are associated with the monolithic integration of memristive devices and arrays with CMOS circuits, as well as co-optimization of materials, devices, and architectures necessary for creating demonstration prototypes of information and computing systems based on memristors.

Various scaling options of active memristive crossbars in integrated implementation provide an increase in signal transmission speed in memristive neural networks with both static and spike coding. The analysis of circuit solutions based on CMOS component base, which ensure efficient operation of the memristive crossbar during training and inference, demonstrates an increase in effective crossbar dimensions in recent years. An alternative solution to monolithic integrated implementation is also presented in the paper through various examples of chiplet technology-based implementations.

Comparison of neuromorphic computing systems based on traditional and new component bases shows that existing prototypes already significantly (by orders of magnitude) outperform known computing systems based on traditional component base in terms of performance and energy efficiency without reducing precision in vector-matrix multiplication and artificial neural network inference.

Acknowledgements

This study was conducted within the framework of the scientific program of the National Center for Physics and Mathematics, section No. 9 “Artificial intelligence and big data in technical, industrial, natural and social systems”.

References

1. Neurochip “Altai”. `https://motivnt.ru/neurochip-altai/`, accessed: 2023-05-15

2. NVIDIA Tesla V100 GPU architecture the world’s most advanced data center GPU. `https://www.nvidia.cn/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/volta-architecture-whitepaper.pdf`, accessed: 2023-05-15

3. Akopyan, F., Sawada, J., Cassidy, A., *et al.*: TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. IEEE Transactions on Computer-

Aided Design of Integrated Circuits and Systems 34(10), 1537–1557 (oct 2015). https://doi.org/10.1109/TCAD.2015.2474396

4. Amirsoleimani, A., Alibart, F., Yon, V., *et al.*: In-Memory Vector-Matrix Multiplication in Monolithic Complementary Metal-Oxide-Semiconductor-Memristor Integrated Circuits: Design Choices, Challenges, and Perspectives. Advanced Intelligent Systems 2(11), 2000115 (nov 2020). https://doi.org/10.1002/AISY.202000115

5. Bianchi, S., Muñoz-Martin, I., Covi, E., *et al.*: A self-adaptive hardware with resistive switching synapses for experience-based neurocomputing. Nature Communications 14(1), 1–14 (mar 2023). https://doi.org/10.1038/s41467-023-37097-5

6. Bocquet, M., Hirztlin, T., Klein, J.O., *et al.*: In-Memory and Error-Immune Differential RRAM Implementation of Binarized Deep Neural Networks. Technical Digest - International Electron Devices Meeting, IEDM 2018-December, 20.6.1–20.6.4 (jan 2019). https://doi.org/10.1109/IEDM.2018.8614639

7. Cai, F., Correll, J.M., Lee, S.H., *et al.*: A fully integrated reprogrammable memristor-CMOS system for efficient multiply-accumulate operations. Nature Electronics 2(7), 290–299 (jul 2019). https://doi.org/10.1038/s41928-019-0270-x

8. Cai, F., Yen, S.H., Uppala, A., *et al.*: A Fully Integrated System-on-Chip Design with Scalable Resistive Random-Access Memory Tile Design for Analog in-Memory Computing. Advanced Intelligent Systems 4(8), 2200014 (aug 2022). https://doi.org/10.1002/AISY.202200014

9. Chen, W.H., Dou, C., Li, K.X., *et al.*: CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors. Nature Electronics 2(9), 420–428 (aug 2019). https://doi.org/10.1038/s41928-019-0288-0

10. Chiaradia, I., Lancaster, M.A.: Brain organoids for the study of human neurobiology at the interface of in vitro and in vivo. Nature Neuroscience 23(12), 1496–1508 (nov 2020). https://doi.org/10.1038/s41593-020-00730-3

11. Chiolerio, A., Chiappalone, M., Ariano, P., Bocchini, S.: Coupling resistive switching devices with neurons: State of the art and perspectives. Frontiers in Neuroscience 11(FEB), 70 (feb 2017). https://doi.org/10.3389/FNINS.2017.00070/BIBTEX

12. Christensen, D.V., Dittmann, R., Linares-Barranco, B., *et al.*: 2022 roadmap on neuromorphic computing and engineering. Neuromorphic Computing and Engineering 2(2), 022501 (may 2022). https://doi.org/10.1088/2634-4386/AC4A83

13. Chua, L.O.: MemristorThe Missing Circuit Element. IEEE Transactions on Circuit Theory 18(5), 507–519 (1971). https://doi.org/10.1109/TCT.1971.1083337

14. Chua, L.O., Kang, S.M.: Memristive Devices and Systems. Proceedings of the IEEE 64(2), 209–223 (1976). https://doi.org/10.1109/PROC.1976.10092

15. Demin, V.A., Erokhin, V.V.: Hidden symmetry shows what a memristor is. International Journal of Unconventional Computing 12, 433–438 (2016)

16. Demin, V.A., Nekhaev, D.V., Surazhevsky, I.A., *et al.*: Necessary conditions for STDP-based pattern recognition learning in a memristive spiking neural network. Neural Networks 134, 64–75 (feb 2021). https://doi.org/10.1016/J.NEUNET.2020.11.005

17. George, R., Chiappalone, M., Giugliano, M., *et al.*: Plasticity and Adaptation in Neuromorphic Biohybrid Systems. iScience 23(10), 101589 (oct 2020). https://doi.org/10.1016/J.ISCI.2020.101589

18. Guggisberg, A.G., Koch, P.J., Hummel, F.C., Buetefisch, C.M.: Brain networks and their relevance for stroke rehabilitation. Clinical Neurophysiology 130(7), 1098–1124 (jul 2019). https://doi.org/10.1016/J.CLINPH.2019.04.004

19. Ham, D., Park, H., Hwang, S., Kim, K.: Neuromorphic electronics based on copying and pasting the brain. Nature Electronics 4(9), 635–644 (sep 2021). https://doi.org/10.1038/s41928-021-00646-1

20. Huang, Y., Kiani, F., Ye, F., Xia, Q.: From memristive devices to neuromorphic systems. Applied Physics Letters 122(11), 110501 (mar 2023). https://doi.org/10.1063/5.0133044/2880793

21. Hung, J.M., Xue, C.X., Kao, H.Y., *et al.*: A four-megabit compute-in-memory macro with eight-bit precision based on CMOS and resistive random-access memory for AI edge devices. Nature Electronics 4(12), 921–930 (dec 2021). https://doi.org/10.1038/s41928-021-00676-9

22. Kim, J., Pershin, Y.V., Yin, M., *et al.*: An Experimental Proof that Resistance-Switching Memory Cells are not Memristors. Advanced Electronic Materials 6(7), 2000010 (jul 2020). https://doi.org/10.1002/AELM.202000010

23. Krasnikov, G.Y.: The capabilities of microelectronic processes with 5 nm critical dimension and less. Nanoindustry Russia 13(S5-1(102)), 13–19 (2020)

24. Krishnan, G., Mandal, S.K., Pannala, M., *et al.*: SIAM: Chiplet-based Scalable In-Memory Acceleration with Mesh for Deep Neural Networks. ACM Transactions on Embedded Computing Systems (TECS) 20(5s) (sep 2021). https://doi.org/10.1145/3476999

25. LaPedus, M.: Chiplet Momentum rising. Semiconductor Engineering. https://semiengineering.com/chiplet-momentum-rising/ (2020), accessed: 2022-10-28

26. Li, C., Hu, M., Li, Y., *et al.*: Analogue signal and image processing with large memristor crossbars. Nature Electronics 1(1), 52–59 (dec 2017). https://doi.org/10.1038/s41928-017-0002-z

27. Liu, Q., Gao, B., Yao, P., *et al.*: A Fully Integrated Analog ReRAM Based 78.4TOPS/W Compute-In-Memory Chip with Fully Parallel MAC Computing. Digest of Technical Papers - IEEE International Solid-State Circuits Conference 2020-February, 500–502 (feb 2020). https://doi.org/10.1109/ISSCC19947.2020.9062953

28. Liu, X., Zeng, Z.: Memristor crossbar architectures for implementing deep neural networks. Complex and Intelligent Systems 8(2), 787–802 (apr 2022). https://doi.org/10.1007/S40747-021-00282-4/TABLES/7

29. Liu, Z., Tang, J., Gao, B., *et al.*: Multichannel parallel processing of neural signals in memristor arrays. Science Advances 6(41) (oct 2020). `https://doi.org/10.1126/SCIADV.ABC4797/SUPPL_FILE/ABC4797_SM.PDF`

30. Liu, Z., Tang, J., Gao, B., *et al.*: Neural signal analysis with memristor arrays towards high-efficiency brain-machine interfaces. Nature Communications 11(1), 1–9 (aug 2020). `https://doi.org/10.1038/s41467-020-18105-4`

31. Makarov, V.A., Lobov, S.A., Shchanikov, S., *et al.*: Toward Reflective Spiking Neural Networks Exploiting Memristive Devices. Frontiers in Computational Neuroscience 16, 62 (jun 2022). `https://doi.org/10.3389/FNCOM.2022.859874/BIBTEX`

32. Mehonic, A., Kenyon, A.J.: Brain-inspired computing needs a master plan. Nature 604(7905), 255–260 (apr 2022). `https://doi.org/10.1038/s41586-021-04362-w`

33. Mikhaylov, A., Pimashkin, A., Pigareva, Y., *et al.*: Neurohybrid memristive cmos-integrated systems for biosensors and neuroprosthetics. Frontiers in Neuroscience 14, 358 (apr 2020). `https://doi.org/10.3389/FNINS.2020.00358/BIBTEX`

34. Miranda, E., Suñé, J.: Memristors for Neuromorphic Circuits and Artificial Intelligence Applications. Materials 13(4), 938 (feb 2020). `https://doi.org/10.3390/MA13040938`

35. Mochida, R., Kouno, K., Hayata, Y., *et al.*: A 4M synapses integrated analog ReRAM based 66.5 TOPS/W neural-network processor with cell current controlled writing and flexible network architecture. Digest of Technical Papers - Symposium on VLSI Technology 2018-June, 175–176 (oct 2018). `https://doi.org/10.1109/VLSIT.2018.8510676`

36. Pei, J., Deng, L., Song, S., *et al.*: Towards artificial general intelligence with hybrid Tianjic chip architecture. Nature 572(7767), 106–111 (jul 2019). `https://doi.org/10.1038/s41586-019-1424-8`

37. Pfeiffer, P., Egusquiza, I.L., DI Ventra, M., *et al.*: Quantum memristors. Scientific Reports 6(1), 1–6 (jul 2016). `https://doi.org/10.1038/srep29507`

38. Pulvermüller, F., Tomasello, R., Henningsen-Schomers, M.R., Wennekers, T.: Biological constraints on neural network models of cognitive function. Nature Reviews Neuroscience 22(8), 488–502 (jun 2021). `https://doi.org/10.1038/s41583-021-00473-5`

39. Rao, M., Tang, H., Wu, J., *et al.*: Thousands of conductance levels in memristors integrated on CMOS. Nature 615(7954), 823–829 (mar 2023). `https://doi.org/10.1038/s41586-023-05759-5`

40. Roy, K., Jaiswal, A., Panda, P.: Towards spike-based machine intelligence with neuromorphic computing. Nature 575(7784), 607–617 (nov 2019). `https://doi.org/10.1038/s41586-019-1677-2`

41. Schegolev, A.E., Klenov, N.V., Soloviev, I.I., *et al.*: Superconducting Neural Networks: from an Idea to Fundamentals and, Further, to Application. Nanobiotechnology Reports 16(6), 811–820 (nov 2021). `https://doi.org/10.1134/S2635167621060227/METRICS`

42. Sebastian, A., Le Gallo, M., Khaddam-Aljameh, R., Eleftheriou, E.: Memory devices and applications for in-memory computing. Nature Nanotechnology 15(7), 529–544 (mar 2020). https://doi.org/10.1038/s41565-020-0655-z

43. Serb, A., Corna, A., George, R., *et al.*: Memristive synapses connect brain and silicon spiking neurons. Scientific Reports 10(1), 1–7 (feb 2020). https://doi.org/10.1038/s41598-020-58831-9

44. Shchanikov, S., Zuev, A., Bordanov, I., *et al.*: Designing a bidirectional, adaptive neural interface incorporating machine learning capabilities and memristor-enhanced hardware. Chaos, Solitons and Fractals 142, 110504 (jan 2021). https://doi.org/10.1016/J.CHAOS.2020.110504

45. Shen, J., Shang, D., Chai, Y., *et al.*: Nonvolatile Multilevel Memory and Boolean Logic Gates Based on a Single Ni/ [Pb (Mg1/3Nb2/3) O3] 0.7 [PbTiO3] 0.3 /Ni Heterostructure. Physical Review Applied 6(6), 064028 (dec 2016). https://doi.org/10.1103/PHYSREVAPPLIED.6.064028/FIGURES/5/MEDIUM

46. Spagnolo, M., Morris, J., Piacentini, S., *et al.*: Experimental photonic quantum memristor. Nature Photonics 16(4), 318–323 (mar 2022). https://doi.org/10.1038/s41566-022-00973-5

47. Su, F., Chen, W.H., Xia, L., *et al.*: A 462GOPs/J RRAM-based nonvolatile intelligent processor for energy harvesting IoE system featuring nonvolatile logics and processing-in-memory. Digest of Technical Papers - Symposium on VLSI Technology pp. C260–C261 (jul 2017). https://doi.org/10.23919/VLSIT.2017.7998149

48. Telminov, O., Gornev, E.: Possibilities and Limitations of Memristor Crossbars for Neuromorphic Computing. Proceedings - 6th Scientific School "Dynamics of Complex Networks and their Applications", DCNA 2022 pp. 278–281 (2022). https://doi.org/10.1109/DCNA56428.2022.9923302

49. Vasileiadis, N., Ntinas, V., Sirakoulis, G.C., Dimitrakis, P.: In-Memory-Computing Realization with a Photodiode/Memristor Based Vision Sensor. Materials 14(18), 5223 (sep 2021). https://doi.org/10.3390/MA14185223

50. Vongehr, S., Meng, X.: The Missing Memristor has Not been Found. Scientific Reports 5(1), 1–7 (jun 2015). https://doi.org/10.1038/srep11657

51. Wan, W., Kubendran, R., Schaefer, C., *et al.*: A compute-in-memory chip based on resistive random-access memory. Nature 608(7923), 504–512 (aug 2022). https://doi.org/10.1038/s41586-022-04992-8

52. Wang, S., Li, Y., Wang, D., *et al.*: Echo state graph neural networks with analogue random resistive memory arrays. Nature Machine Intelligence 5(2), 104–113 (feb 2023). https://doi.org/10.1038/s42256-023-00609-5

53. Wang, Z., Wu, H., Burr, G.W., *et al.*: Resistive switching materials for information processing. Nature Reviews Materials 5(3), 173–195 (jan 2020). https://doi.org/10.1038/s41578-019-0159-3

54. Wu, T.F., Le, B.Q., Radway, R., *et al.*: 14.3 A 43pJ/Cycle Non-Volatile Microcontroller with 4.7μs Shutdown/Wake-up Integrating 2.3-bit/Cell Resistive RAM and Resilience Techniques. Digest of Technical Papers - IEEE International Solid-State Circuits Conference 2019-February, 226–228 (mar 2019). `https://doi.org/10.1109/ISSCC.2019.8662402`

55. Xia, Q., Yang, J.J.: Memristive crossbar arrays for brain-inspired computing. Nature Material 18(4), 309–323 (mar 2019). `https://doi.org/10.1038/s41563-019-0291-x`

56. Xue, C.X., Chen, W.H., Liu, J.S., *et al.*: 24.1 A 1Mb Multibit ReRAM Computing-In-Memory Macro with 14.6ns Parallel MAC Computing Time for CNN Based AI Edge Processors. Digest of Technical Papers - IEEE International Solid-State Circuits Conference 2019-February, 388–390 (mar 2019). `https://doi.org/10.1109/ISSCC.2019.8662395`

57. Xue, C.X., Chiu, Y.C., Liu, T.W., *et al.*: A CMOS-integrated compute-in-memory macro based on resistive random-access memory for AI edge devices. Nature Electronics 4(1), 81–90 (dec 2020). `https://doi.org/10.1038/s41928-020-00505-5`

58. Yao, P., Wu, H., Gao, B., *et al.*: Fully hardware-implemented memristor convolutional neural network. Nature 577(7792), 641–646 (jan 2020). `https://doi.org/10.1038/s41586-020-1942-4`

59. Yin, S., Sun, X., Yu, S., Seo, J.S.: High-Throughput In-Memory Computing for Binary Deep Neural Networks with Monolithically Integrated RRAM and 90-nm CMOS. IEEE Transactions on Electron Devices 67(10), 4185–4192 (oct 2020). `https://doi.org/10.1109/TED.2020.3015178`

60. Zhang, W., Gao, B., Tang, J., *et al.*: Neuro-inspired computing chips. Nature Electronics 3(7), 371–382 (jul 2020). `https://doi.org/10.1038/s41928-020-0435-7`

61. Zhao, H., Liu, Z., Tang, J., *et al.*: Energy-efficient high-fidelity image reconstruction with memristor arrays for medical diagnosis. Nature Communications 14(1), 1–10 (apr 2023). `https://doi.org/10.1038/s41467-023-38021-7`

62. Zhu, X., Wang, Q., Lu, W.D.: Memristor networks for real-time neural activity analysis. Nature Communications 11(1), 1–9 (may 2020). `https://doi.org/10.1038/s41467-020-16261-1`

63. Zhuk, M., Zarubin, S., Karateev, I., *et al.*: On-Chip TaOx-Based Non-volatile Resistive Memory for in vitro Neurointerfaces. Frontiers in Neuroscience 14, 94 (feb 2020). `https://doi.org/10.3389/FNINS.2020.00094/BIBTEX`

64. Zidan, M.A., Fahmy, H.A.H., Hussain, M.M., Salama, K.N.: Memristor-based memory: The sneak paths problem and solutions. Microelectronics Journal 44(2), 176–183 (feb 2013). `https://doi.org/10.1016/J.MEJO.2012.10.001`

DOI: 10.14529/jsfi230207

Improving Efficiency of Hybrid HPC Systems Using a Multi-agent Scheduler and Machine Learning Methods

Vladimir S. Zaborovsky[1], *Lev V. Utkin*[1], *Vladimir A. Muliukha*[1], *Alexey A. Lukashin*[1]

One of the promising directions for improving hybrid reconfigurable high-performance computer platforms operating in the mode of collaborative applied computing centers is their inclusion as an active component in the machine learning ecosystem, which opens up new opportunities to enhance the actual outperformance of solving various application tasks by intellectualizing the management of available computing resources. The task scheduler operation is crucial in improving the efficiency of hybrid supercomputer platforms, which combine dozens of processor blocks with different architectures, including specialized graphics and reconfigurable accelerators. To form an optimal order of jobs in the HPC queue, the article proposes to apply deep survival machine learning models, which increase the accuracy of the estimated time of the tasks successful execution and the required amount of computing resources. The main peculiarity of the machine learning models is that they are trained on censored heterogeneous data collected from previous periods of task execution observations using a multi-agent scheduler. In order to ensure high accuracy, the random survival forest is used as a part of the machine learning model which provides survival and hazard functions in the framework of the survival analysis. A specific weighted clustering procedure is proposed to divide tasks in accordance with their execution times as well as the feature vectors. Various numerical experiments with actual data illustrate the outperformance of the presented approach.

Keywords: high performance computing, hybrid computing systems, machine learning, multi-agent scheduler, random survival forest, survival analysis, survival function, XAI.

Introduction

A modern supercomputer is a very complex technical system that simultaneously performs quadrillions (a number with 15 zeros) operations used to solve complex mathematical computations and process huge amounts of data that arise in the process of solving various scientific and engineering problems. Supercomputer users, whose number is steadily growing every year, are interested in the real result of their calculations, and not just the peak performance of the supercomputer, which is nominally expressed in the number of floating-point arithmetic operations per second. As the performance of supercomputers increases, the complexity of their use also increases. Therefore, users need to have a deep knowledge not only in the field of their professional activity but also real skills in the most efficient use of available computing resources to consume the real performance of a supercomputer. In this process, the task scheduler plays an important role, combining different groups of processors, including graphics (vector) and reconfigurable (FPGA) accelerators, into a consistent hybrid computing field available for a specific user task. To improve the efficiency of the scheduler for a wide range of applied tasks, machine learning models with attention mechanisms are proposed. These models allow calculating with high accuracy the probability of successful completion of a user task in a given time interval, as well as evaluating specific task parameters that can greatly affect the actual performance of the hybrid platform. The system parameters, machine learning models, and promising transformer-based architecture presented in the article were obtained based on the analysis of the functioning of a

[1]Peter the Great St.Petersburg Polytechnic University, St.Petersburg, Russian Federation

hybrid supercomputer cluster of St. Petersburg Polytechnic University with a peak performance of more than 1 PFlops.

One of the tools for dealing with the task completion problem is survival analysis [7] which is used in many applied areas [13, 20, 27]. A comprehensive review of survival analysis methods and their implementation by the machine learning models can be found in [27]. An important peculiarity of the survival models is that their outcomes are functions (the survival function, the hazard function, the cumulative hazard function, etc.) as predictions instead of point-valued data which are predicted by most machine learning models.

All survival models can be divided into three groups. The first group consists of parametric models for which a probability distribution of time to event is known, but its parameters are unknown. The second group contains semi-parametric models for which it is assumed that a functional dependence between features and the model outcomes is known. The well-known Cox proportional hazards model [4] belongs to the second group. Models of the third group are called non-parametric. They assume that the probability time-to-event distribution is unknown, and the relationship between features and the model outcomes is also unknown. The Kaplan–Meier model [27] is one of the models from the third group.

Following the Cox model, many its extensions have been proposed, which use the non-linear relationship between covariates and the time of event [20]. A lot of models are based on neural networks. However, the difficulty to train a neural network, when the number of training examples is restricted, led to applying another approach based on using the random survival forests [2], which can be regarded as an extension of the original random forest [3] to the case of survival analysis. It turns out that the random survival forests are a powerful and efficient tool for survival analysis due to their several useful peculiarities. First of all, random forests require a few tuning parameters [9]. Random forests are highly data adaptive and can deal with both low and high dimensional data [26]. Therefore, we use the random survival forests as a basic model for analyzing the task completion time.

Our contributions can be summarized as follows:

1. Collection of job execution data and comprehensive statistical analysis of performed jobs taking into account domain and user characteristics.
2. Survival analysis for computing probabilistic characteristics of the task completion time (survival functions, expected times of the task completion, etc.) is proposed to be applied. One of the reasons for using survival analysis is the availability of many tasks which have been terminated due to several reasons and are considered as censored data.
3. The predictions of the task completion time by means of a set of the random survival forests are proposed to be performed. The random survival forest is a powerful and efficient tool for the case when the number of training data is limited.
4. A specific procedure for clustering training data and for training several random survival forests is proposed, which allows us to take into account the fact that tasks may be quite different and they should be separated into clusters with a homogeneous structure. This approach is supplemented by a specific procedure of computing predictions for new tasks.
5. A simple approach for taking into account the user history is proposed, which is reduced to computing probability distributions of different task completion events. The corresponding probability distributions are concatenated with the initial feature vector obtained for the analyzed user.

6. We consider questions how to explain prediction of the task completion time in order to have an opportunity for the user to change the task parameters or just to understand why the corresponding task can be completed unsuccessfully.
7. An approach of integrating the developed prediction model into the existing environment of a hybrid supercomputer environment by integration of the predicted execution time of a job to a scheduler.

The article is organized as follows. A description of the supercomputer analyzed and of its component (the Slurm Task Scheduler) is given in Section 1. General approaches to improving the efficiency of the SCC are discussed in Section 2. Elements of survival analysis are considered in Section 3. The main algorithm for the task completion prediction is provided in Section 4. Ways for improving the proposed algorithm based on survival analysis are described in Section 5. Section 6 considers the important question of interpretation of the machine learning analysis results. Numerical experiments with the analysis of results are considered in Section 7. Concluding remarks and perspectives are discussed in the Conclusion section.

1. Current Situation in the Supercomputer Center "Polytechnic"

1.1. Description of the Supercomputer Center "Polytechnic"

Supercomputing Centre "Polytechnic"(hereinafter SCC) is a hybrid high performance computing system consisting of several computing clusters with different architectures (homogeneous and heterogeneous) located in a single information and computing field and connected using 56 Gb/s Infiniband FDR, as well as a common storage Luster, with a volume of about 1 PB, which allows file exchange between different computing clusters. All SCC systems provide more than 1.5 PFlops performance on double-precision floating point computation and more than 2.5 PFlops on single and half-precision computations that are typically used for training machine learning algorithms.

All registered users get access to SCC resources from the dedicated server using the terminal client protocol. SCC resources are managed using the Slurm Workload Manager. The principle of its operation is as follows (Fig. 1): the user requests some resource (processor cores, memory, etc.), placing his task in the queue; the system, based on the user's priorities and the current filling of the queue, selects the moment of task launch. A queue is a sequence of tasks that must be solved on a specific computing resource (a group of nodes). At the same time, each node at the current time can be occupied by only one task of one user. Thus, the node is assigned to the exclusive use of the task hosted on it, and other tasks on the busy node will not be executed.

The life cycle of a task created by a user in the SCC consists of four stages:

- user connection to the console of the control node of the SCC through a client (for example, PuTTY for Windows OS, or built-in SSH client for Linux OS);
- creating and running a task in the console in interactive or batch mode;
- execution of the task on the selected resources of the SCC;
- completion of the task.

SCC provides different systems by dividing resources into multiple queues (partitions in SLURM terminology). Currently, the following clusters are available:

- Tornado: homogeneous cluster with 28-core and 64 GB of RAM nodes;

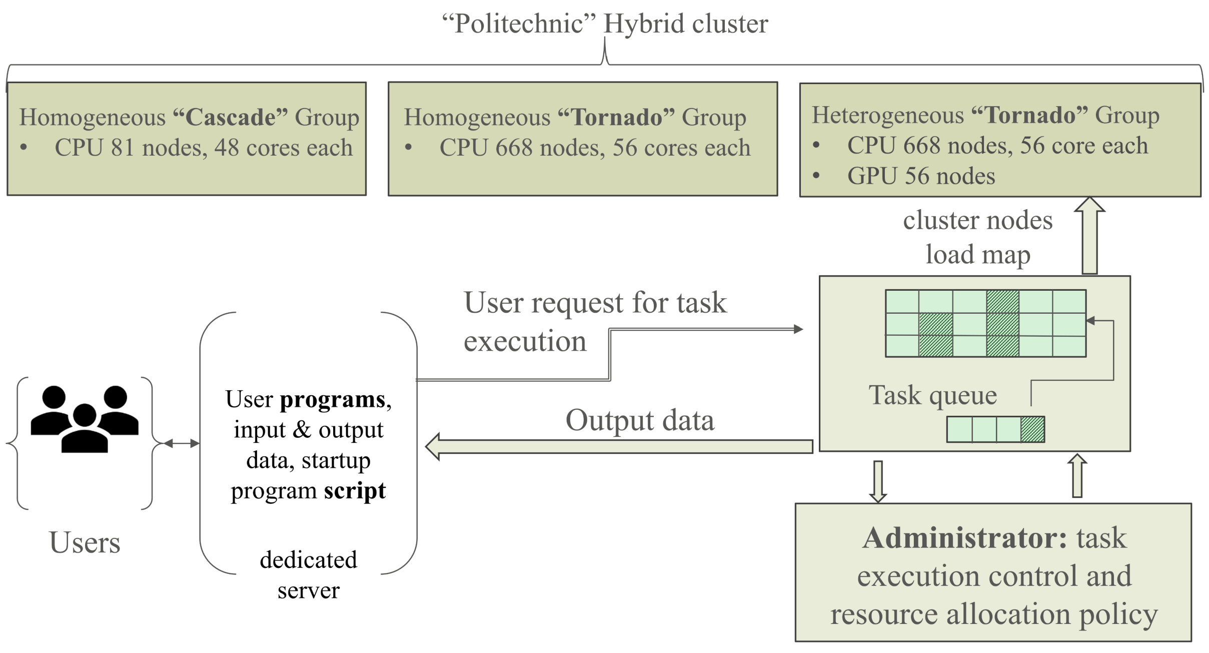

"Politechnic" Hybrid cluster
Homogeneous "Cascade" Group
• CPU 81 nodes, 48 cores each
Homogeneous "Tornado" Group
• CPU 668 nodes, 56 cores each
Heterogeneous "Tornado" Group
• CPU 668 nodes, 56 core each
• GPU 56 nodes
cluster nodes load map
User request for task execution
Task queue
Output data
User programs, input & output data, startup program script
dedicated server
Users
Administrator: task execution control and resource allocation policy

Figure 1. Structure and information flows of the supercomputer center "Polytechnic"

- Tornado-k40: heterogeneous cluster with 28-core, 64 GB of RAM, and 2 Tesla GPUs nodes;
- Cascade: homogeneous cluster with 48-core and 192 GB of RAM nodes;
- NV: heterogeneous cluster with 48-core, 768 GB of RAM, and 8 Tesla GPUs nodes.

Every job is submitted into a particular partition with specific resources available. All clusters are connected to the same storage. That provides users with the ability to use different compute resources for the different stages of numerical simulations and data processing.

1.2. Description of the Slurm Task Scheduler

Slurm is a cluster management and task planning system. It performs three main functions:
- distributes access to resources (computing nodes) for users;
- provides an environment for launching, executing and monitoring tasks on dedicated nodes;
- manages the task queue.

The user can submit tasks in two modes via the command line – interactive and batch modes.

In interactive mode, the user can act according to two algorithms. In the first case, resources are requested, the application necessary for operation is launched on the head node of the SCC, and then the user is working on the application. In the second case, a connection is made to the selected node, and the rest of the actions are performed on this node.

In batch mode, the user prepares the task script in the command window: determines the parameters for launching a task, configures the working environment necessary for the application to work, and determines the task launch sequence. After that, the task is placed in the corresponding queue.

Job configuration supports a large number of different parameters like the number of cores or the amount of GBs RAM per process. But the most important parameters are the amount of compute resources (nodes) and the time of their allocation.

Additionally, the number of processors per subtask, the amount of memory in the node, the amount of memory in the processor, the number of subtasks in the task, the start time of the task, and others can be specified. The user has the ability to select a variety of options for a detailed description of how the task should be performed.

The Slurm forms a separate job queue for each type of SCC computing resource. While creating a task, users independently select the required resources. Due to the qualification of most users in the field of parallel programming, they prefer to use traditional CPU-based nodes. And often they use only 1–2 nodes for their task. Thus, different clusters within the SCC are loaded differently. An example of the loading of computing clusters of the SCC “Polytechnic”is shown in Fig. 2.

All tasks were divided into classes depending on the field of science: Astrophysics, Bioinformatics, Biophysics, Energy, Geophysics, Informatics, Mechanical Engineering, Mechanics, Physics and Radiophysics. For clarity, in Fig. 2, each of the classes is indicated by its own color:

- Astrophysics is turquoise;
- Bioinformatics is red;
- Biophysics is pink;
- Energy is green;
- Geophysics is yellow;
- Informatics is black;
- Mechanical Engineering is blue;
- Mechanics is gray;
- Physics is lime;
- Radiophysics is orange;
- Uncertain Tasks are purple.

2. Approaches to Improving the Efficiency of the SCC

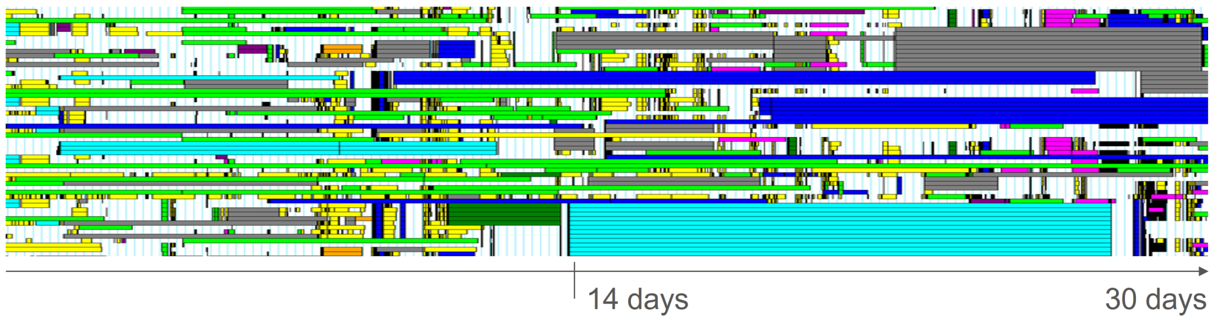
One of the promising directions for increasing the efficiency of using supercomputer resources is the use of various task parameters to adapt the process of its solution.

Currently, SPbPU considers two main areas of adaptation of the work of the SCC:

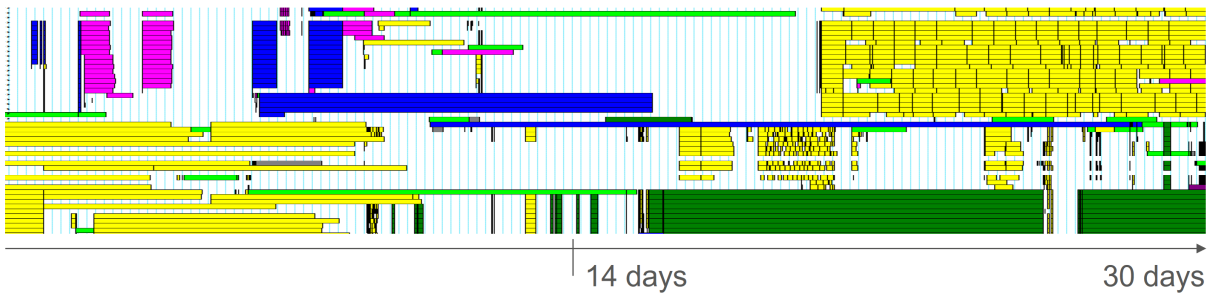
1. The use of intelligent technologies and machine learning methods for a preliminary assessment of the effectiveness of solving a new problem based on statistical data on solving similar problems in the past.
2. Development of methods for adapting the SCC hardware to the requirements of a specific task by using reconfigurable computers.

2.1. Application of Machine Learning Methods to Optimize the Parameters of Tasks Received by the SCC

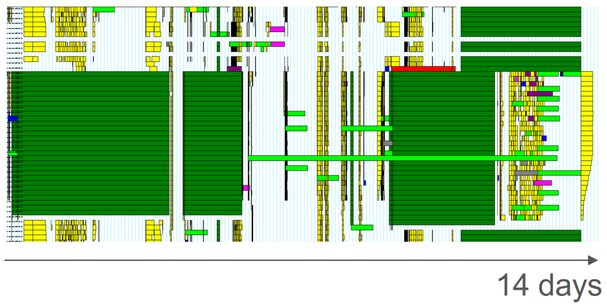
In this work using a machine learning approach to improve the real performance of SCC is proposed. The hybrid supercomputer cluster should act as an active component of the machine learning ecosystem (Fig. 3). In this mode, the supercomputer generates a multiset of calculation data, as well as information on how the application tasks affect the cluster resources. Although all data are in principle available to users, their interpretation requires high skill and, therefore, is much more difficult in practice. Obviously, most of this data can be used for machine learning



(a) Homogeneous "Tornado" Group nodes



(b) Heterogeneous "Tornado" Group nodes



(c) Homogeneous "Cascade" Group nodes

Figure 2. Uneven loading of cluster nodes of the supercomputer center "Polytechnic"

of intelligent blocks cluster scheduler, including for constructing a user qualification model that characterizes the description confidence level and script parameters for the executable task.

As a result, the intelligent block of the Slurm scheduler can generate estimates of the efficiency of using available resources, as well as generate a statistical metric of confidence in the results obtained, including recommendations for improving the efficiency of calculations. All data generated by the intelligent block of the Slurm scheduler, along with the direct results of calculations, are provided to users, and are also taken into account as parameters in the trained model of his qualification profile.

The machine learning ecosystem infrastructure built in this way, which includes both the supercomputer itself and its various users, allows you to analyze the entire course of calculations performed and fixing the trajectory of each application process, organizing an effective machine learning process of the resource scheduler, including for the tasks of new users.

In this case, the training sample includes both "successful" and "unsuccessful" sets of completing tasks, that carry out important information characterizing the so-called "survivor's error". Generating an explanation of why the application task "survived" or why it was not completed within the time specified by the scheduler will speed up the user's understanding of the

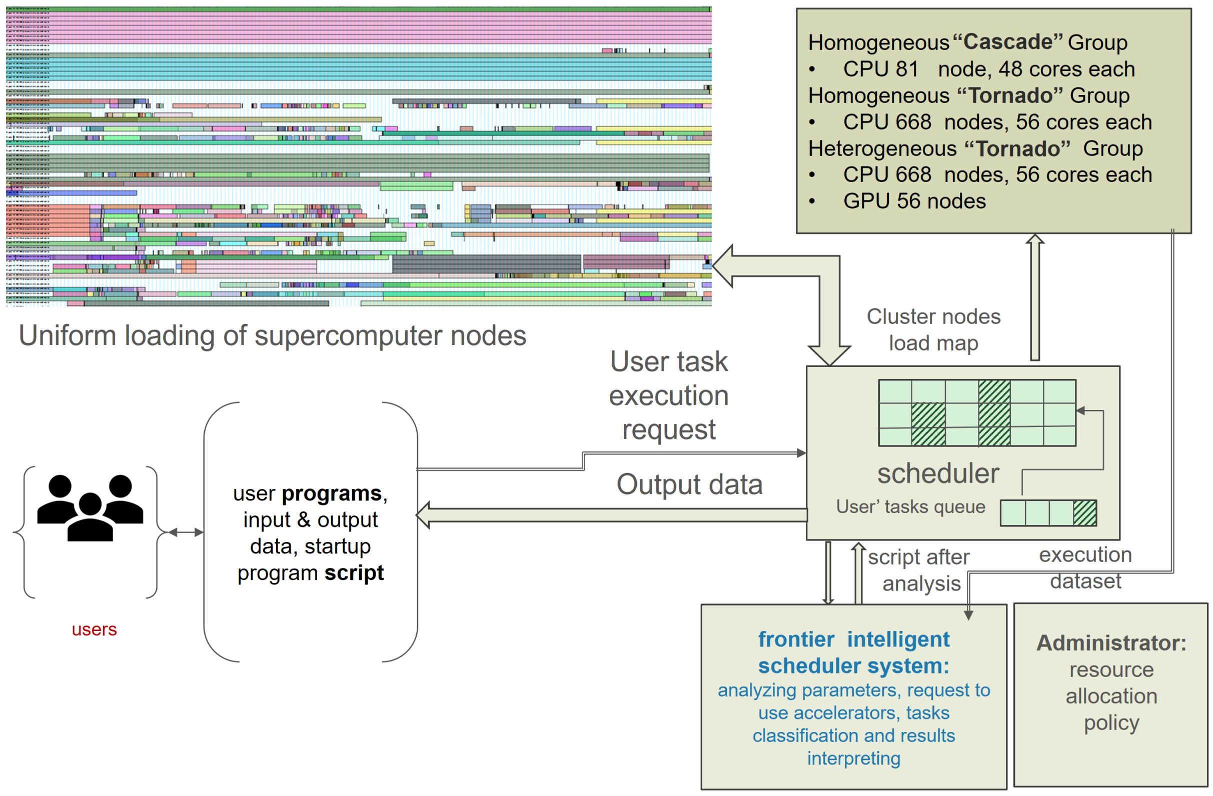

Homogeneous "Cascade" Group
• CPU 81 node, 48 cores each
Homogeneous "Tornado" Group
• CPU 668 nodes, 56 cores each
Heterogeneous "Tornado" Group
• CPU 668 nodes, 56 cores each
• GPU 56 nodes
Uniform loading of supercomputer nodes
Cluster nodes load map
User task execution request
Output data
scheduler
User' tasks queue
user programs, input & output data, startup program script
users
script after analysis
execution dataset
frontier intelligent scheduler system: analyzing parameters, request to use accelerators, tasks classification and results interpreting
Administrator: resource allocation policy

Figure 3. Structure and information flows of the supercomputer center "Polytechnic"

specifics of using supercomputer resources. At the same time, the dispatcher will be able to predictively calculate the parameters of the description of the applied task, which increase the probability of its successful completion, as well as "learn"to respond to new tasks, including reconfiguration of accelerators used to solve special applied tasks.

2.2. Multi-agent Scheduler of Reconfigurable Computational Resources

To expand the capabilities of the SCC in terms of solving specialized tasks, a cluster of Tertius-2 nodes containing FPGA appeared as part of our hybrid SCC in 2022. The work on the integration of the reconfigurable nodes (reconfigurable computation units – RCU) into the computational field of the SCC is in progress. The following scheme for integrating RCU into the structure of the SCC "Polytechnic"is proposed in Fig. 4. The advantage of the proposed scheme is the presence of regular mechanisms for integrating the RCU into the computational field of the SCC "Polytechnic", as well as the implementation of the possibility of direct control of RCU by the Multi-agent scheduler after the initialization procedure.

According to research works [11, 12] the multi-agent scheduler (MAS) is an effective decentralized method of managing a distributed computing cluster, including a heterogeneous one.

In general terms, the process of solving problems with the use of RCU consists of the following steps:

1. Upon request from the MAS RCU to Slurm, the latter allocates RCU resources to full control of the MAS by launching a software agent on each allocated RCU.
2. Among the tasks received from the user, the AI module for reconfigurable system (AIM) selects those that can be solved in the RCU using the RCU firmware that exist in the

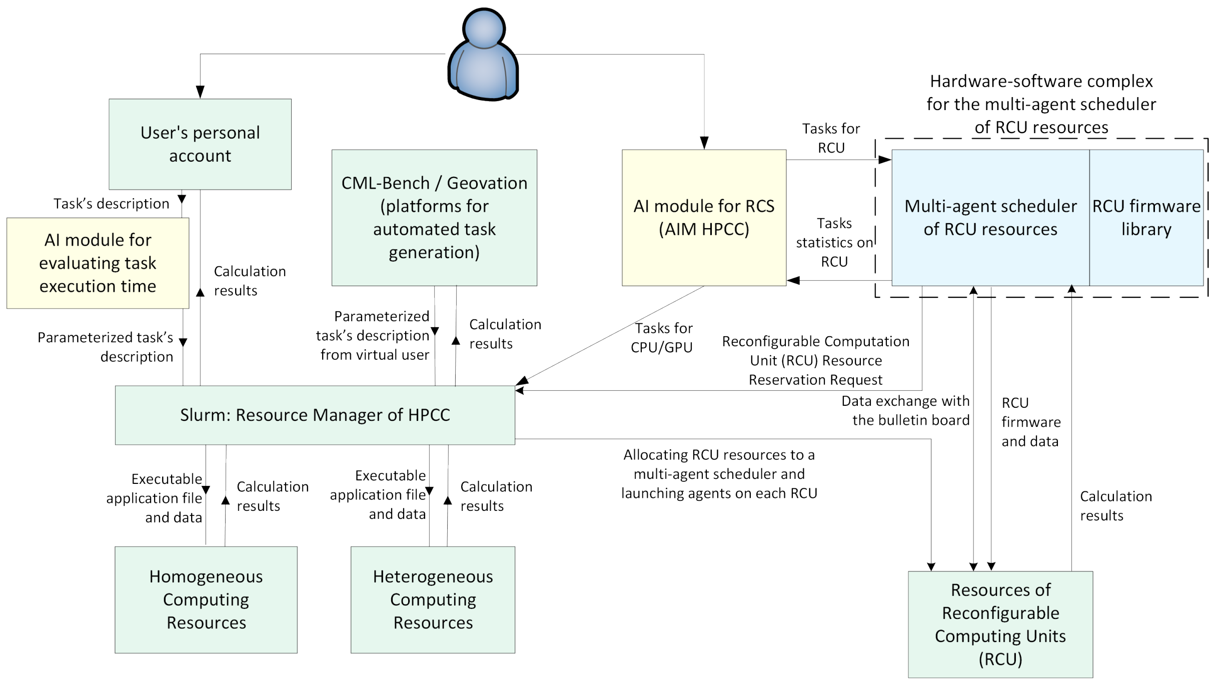

User's personal account
Task's description
AI module for evaluating task execution time
Calculation results
Parameterized task's description
CML-Bench / Geovation (platforms for automated task generation)
Parameterized task's description from virtual user
Calculation results
AI module for RCS (AIM HPCC)
Tasks for RCU
Tasks statistics on RCU
Hardware-software complex for the multi-agent scheduler of RCU resources
Multi-agent scheduler of RCU resources
RCU firmware library
Tasks for CPU/GPU
Reconfigurable Computation Unit (RCU) Resource Reservation Request
Data exchange with the bulletin board
RCU firmware and data
Slurm: Resource Manager of HPCC
Executable application file and data
Calculation results
Executable application file and data
Calculation results
Allocating RCU resources to a multi-agent scheduler and launching agents on each RCU
Calculation results
Homogeneous Computing Resources
Heterogeneous Computing Resources
Resources of Reconfigurable Computing Units (RCU)

Figure 4. Proposal for the integration of reconfigurable nodes under the control of a multi-agent scheduler into the SCC "Polytechnic"

firmware library. A task descriptor is formed as a set of parameters characterizing the selected task.

3. AIM, if necessary, decomposes the input data of the task into blocks that can be processed by the RCU, and for each of the blocks, the preferred solution method is selected from the existing RCU configuration modules in the library.
4. AIM transmits the generated task descriptions with a descriptor to any of the bulletin board.
5. Each software agent on the RCU independently polls all bulletin boards to find the most suitable user tasks. For each of the tasks on the bulletin board, using the information generated by the AIM, the agent determines whether it can be performed.
6. If the agent can compute the task, it, using information from the AIM, determines which RCU configuration module from those available in the library it should use for this.
7. If the agent decides to solve a problem, if necessary, it downloads the recommended RCU configuration module from the library and loads it into the RCU, executes the task in the RCU computational field, and sends the results to the required address.
8. The statistics of the task execution time on the RCU are stored for the subsequent use of additional training of the AIM.

In the course of future work, it is planned to consider the possibility of IM integration directly on each MAS software agent.

As part of the work in 2023, the library will contain at least two RCU firmware designed for solving matrices up to 4000 by 4000 using the LU decomposition method and solving diagonal matrices with 3, 5, and 7 diagonals using the Jacobi method. In the future, the list of RCU firmware in the library can be expanded.

Upon request from MAS to Slurm, the latter allocates RCU resources for full management by launching a certain software agent on each allocated RCU (it is a task for Slurm).

With the use of AIM, among the tasks arriving at the SCC, there are computationally time-consuming tasks that can be solved in the RCU using the configuration modules existing in the library. The data of tasks, if necessary, are divided in the AIM of the SCCs into parts corresponding to the capabilities of the RCU. As a result, a task is formed with a descriptor (set of parameters) for solving on the Tertius-2 RCU, and all this is transferred to any of the MAS bulletin boards to ensure the possibility of a parallel solution in the RCS.

A software agent (hereinafter PA) is software for managing and adapting resources of a heterogeneous computing cluster for solving problems of a dynamic search for a load for an agent-controlled computing resource and adaptive reconfiguration of hardware components of the RCU, constantly running on the universal processor of the RCU and, together with other MAS agents, forming a single computing – communication space.

The software agent can receive data about its parameters from the RCU. In addition, the SCC already has software that allows you to monitor the parameters of all RCUs.

Each of the PA, in case of readiness of its RTM for work and the absence of tasks currently performed on its RTM, independently interrogates all DOs to find the most suitable user tasks.

For each of the tasks on the DO, using the information generated by the RCC IM and attached to the task, the agent determines whether it can complete it. If it can fulfill the task, it, using information from the IM SKTS, determines which RCU configuration modules from those available in the library must be used. If the agent decides to solve a problem, if necessary, it downloads the configuration module from the library and loads it into the RCU, executes the task in the RCU computational field, and sends the results to the required address.

Depending on the mode of operation of the RCS, in the event of a long idle time or at the command of the operator, the agent can complete its work and return control to the RBC Slurm.

The first step in increasing the efficiency of a supercomputer is to minimize the average time jobs are spending in the queue. For making a denser queue, Slurm needs a reliable information about the task execution time. As studies have shown, the results of which are given below, users significantly overestimate the task execution time and the Slurm is forced to look for a larger slot in queue to put the task in it. This situation negatively affects the length and density of the queue.

Machine learning models are proposed to be used for more accurate estimation of the task execution time. The input of such models is the vector of task's parameters $\mathbf{x}_i = (x_{i1}, ..., x_{im})$, and the output is the expected execution time of this task. The values of the individual parameters of the task i in vector $\mathbf{x}_i$ are taken from the data of the Slurm processes: scontrol, sacct, and sbatch. For example, x_{i1} is the ID of the user running the task, and x_{i2} is the ID of the user's group, followed by the requested number of computing nodes, and so on, including meta-parameters of the executable file, such as included libraries. A description of the machine learning methods used in the work is given below.

3. Some Elements of Survival Analysis and Preliminaries

3.1. Basic Concepts of Survival Analysis

We represent a dataset D of tasks in the framework of survival analysis as a set of triplets of the form $(\mathbf{x}_i, \delta_i, T_i)$, where $\mathbf{x}_i = (x_{i1}, ..., x_{im})$ is the feature vector which contains all available information about the i-th task represented by m features; T_i is the i-th task completion time.

In contrast to the standard regression analysis, there is the additional component δ_i of the dataset which is the indicator function so that $\delta_i = 1$ if we observe a successful task completion, and $\delta_i = 0$ if the i-th task has not been successfully completed. In the first case ($\delta_i = 1$), time T_i corresponds to the time between the baseline time and the time of the successful task completion. This case corresponds to the uncensored observation. In the second case ($\delta_i = 0$), T_i is the observation time, i.e., the time moment when the task is terminated due to several reasons, and we have the censored observation. The aim of survival analysis is to predict the completion time of a new task characterized by the feature vector $\mathbf{x}$ by using the training dataset consisting of n examples $(\mathbf{x}_i, \delta_i, T_i)$, $i = 1, ..., n$.

Important concepts in survival analysis are the survival and hazard functions. The survival function, denoted as $S(t|\mathbf{x})$, is the probability that the task $\mathbf{x}$ is not completed up to the time t, that is $S(t|\mathbf{x}) = \Pr\{T > t|\mathbf{x}\}$. The hazard function or the hazard rate $h(t|\mathbf{x})$ is the rate of the task completion at time t given that no tasks completed before time t. It can be written as

$$h(t|\mathbf{x}) = \lim_{\Delta t \to 0} \frac{\Pr\{t \leq T \leq t + \Delta t | T \geq t|\mathbf{x}\}}{\Delta t} = \frac{f(t|\mathbf{x})}{S(t|\mathbf{x})}, \tag{1}$$

where $f(t|\mathbf{x})$ is the density function of the task completion.

The hazard function can also be expressed though the survival function as follows:

$$h(t|\mathbf{x}) = -\frac{d}{dt} \ln S(t|\mathbf{x}). \tag{2}$$

Hence, we can express the survival function through the cumulative hazard function $H(t)$ as

$$S(t|\mathbf{x}) = \exp\left(-\int_0^t h(z|\mathbf{x}) \mathrm{d}z \right) = \exp\left(-H(t|\mathbf{x}) \right). \tag{3}$$

It can be seen from the above that the functions depend on the vector $\mathbf{x}$.

There are several well-known models used in survival analysis. First, we should mention the non-parametric Kaplan–Meier model for estimating the survival function. However, the Kaplan–Meier estimator gives the average view of objects (tasks) and does not take into account the task features $\mathbf{x}$. Therefore, we cannot evaluate how the task features influence the survival probabilities. The first model which allows us to estimate the survival or hazard functions depending on the vector $\mathbf{x}$ is the Cox proportional hazards model [4]. According to the model, the hazard function at time t given the feature vector $\mathbf{x}$ is defined as

$$h(t|\mathbf{x}) = h_0(t) \exp\left(\mathbf{x}\mathbf{b}^{\mathrm{T}} \right), \tag{4}$$

where $h_0(t)$ is an arbitrary baseline hazard function; $\mathbf{b} = (b_1, ..., b_m)$ is an unknown vector of regression coefficients or parameters.

The survival function $S(t|\mathbf{x})$ is computed as

$$S(t|\mathbf{x}) = \left(S_0(t) \right)^{\exp\left(\mathbf{x}\mathbf{b}^{\mathrm{T}} \right)}. \tag{5}$$

Here $S_0(t)$ is the baseline survival function. It is important to point out that $S_0(t)$ as well as $h_0(t)$ do not depend on $\mathbf{x}$ and are estimated by using the Kaplan–Meier estimator.

The main peculiarity of the Cox model is the linear combination of features. On the one hand, it simplifies the model and allows us to use it in the interpretation of the model predictions. On the other hand, it restricts the Cox model use because real datasets usually have a more complex structure. Various modifications have been proposed to generalize the Cox model by replacing the linear relationship with some non-linear functions, for example, with neural networks [5, 20], the support vector machine [14, 22].

3.2. Random Survival Forests

Despite the availability of many machine learning models for survival analysis, they require a large amount of training data to obtain reasonable estimations. One of the efficient models dealing with small and heterogeneous data is the random survival forests [10]. A general algorithm of constructing RSFs can be represented as follows:

1. Q bootstrap samples of size N are selected from the original data, where Q is a hyperparameter defining the number of survival trees in the random survival forests.
2. A survival tree is constructed by using a single bootstrap sample so that each node of the tree is split using the candidate feature that maximizes survival difference between daughter nodes. The depth of the trees is also a hyperparameter.
3. The survival function or the cumulative hazard function at each leaf of a tree is calculated by using the Kaplan–Meier estimator or the Nelson–Aalen estimator.
4. The ensemble cumulative hazard function or the ensemble survival function is obtained by averaging the cumulative hazard functions over all trees.

One of the important peculiarities of the tree construction is a splitting rule. Several splitting rules are reviewed in [27]. The most popular rule is the log-rank rule which separates nodes of the decision tree by selecting the split that yields the largest log-rank test [25]. The best split is given by the greatest difference between two daughter nodes which is given by the largest value of the log-rank test.

If we denote the cumulative hazard estimate of the k-th tree as $H_k(t|\mathbf{x})$, then the ensemble cumulative hazard estimate for the random survival forest consisting of Q trees is determined as follows [10]:

$$H(t|\mathbf{x}) = \frac{1}{Q}\sum_{k=1}^{Q} H_k(t|\mathbf{x}). \tag{6}$$

In fact, the above expression can be regarded as the mean function over all cumulative hazard functions predicted by each tree in the random survival forest. The obtained function will be used later for computing the survival function and the expected time to the task completion. In order to verify whether the obtained survival function is the best one from the optimal hyperparameters point of view, the C-index is used.

3.3. C-index

An important question in survival analysis is how to compare the machine learning survival models. One of the measures for comparison is the C-index (the concordance index) proposed by Harrell et al. [6]. The C-index allows us to estimates how good the model is at ranking survival times. This is the probability that the task completion times of a pair of tasks are correctly ranking [19]. In order to formally define the C-index, we first define inadmissible pairs. A pair is not admissible if the task completion events are both right-censored or if the earliest time in the pair is censored. The C-index is calculated as the ratio of the number of pairs correctly ordered by the model to the total number of admissible pairs. If the C-index is equal to 1, then the corresponding survival model is supposed to be perfect. If the C-index is 0.5, then the model is no better than random guessing. By using the predicted survival function $S(t|\mathbf{x})$, we can write the C-index as [27]:

$$C = \frac{1}{M} \sum_{i:\delta_i=1} \sum_{j:T_i<T_j} \mathbf{1}\left[S(T_i|\mathbf{x}_i) > S(T_j|\mathbf{x}_j)\right]. \tag{7}$$

Here M is the number of all comparable or admissible pairs; $\mathbf{1}[\cdot]$ is the indicator function taking the value of 1 if its argument is true, and of 0 – otherwise.

Another definition of the C-index is

$$C = \frac{\sum_i \sum_j \mathbf{1}\left[T_i < T_j\right] \cdot \mathbf{1}\left[\widehat{T}_i < \widehat{T}_j\right] \cdot \delta_i}{\sum_i \sum_j \mathbf{1}\left[T_i < T_j\right] \cdot \delta_i}, \tag{8}$$

where $\widehat{T}_i$ is the predicted survival duration; i and j are indices of all examples from the dataset, $i, j = 1, ..., n$.

4. Algorithm for the Task Completion Prediction

A goal of the proposed algorithm of training the random survival forest is to predict the probabilistic characteristics of the task completion, including the survival function and the expected task completion time. At the first stage of study when the dataset of examples is restricted, we propose to apply the random survival forest as one of the efficient models dealing with the limited number of training data. The random survival forest consists of Q survival trees.

Every vector $\mathbf{x}$ consists of m features which include the most important ones: UserID (the ID of a user), GroupID (the ID of the group on behalf of which the task was queued), NumNodes (the number of nodes requested or allocated for the task), NCPUs (the total number of processors allocated to the task), NumTasks (the total number of subtasks in the task), CPUs/Task (the ratio of the total number of processors to the number of subtasks), ReqB:S:C:T (the number of different hardware components requested for the task), Socks/Node (the desired ratio of the number of sockets to the number of compute nodes), NtasksPerN:B:S:C (the number of subtasks required to run on a specific number of hardware components), CoreSpec (the number of cores reserved), MinCPUsNode (the minimum ratio of the number of processors to the number of nodes), MinMemoryNode (the minimum ratio of memory in MB to the number of nodes), JobID (the task identification number), Priority (the task priority determined by the SLURM scheduler), etc. It should be noted that the completion times T_i of different tasks are changed in a large interval of time. The distribution of tasks in accordance with their completion times is shown in Fig. 5. It can be seen from this distribution that the number of "long" tasks is rather small in comparison with tasks completed in a short time. This causes a difficulty for training the random survival forest because survival trees are mainly trained on the "short" task and do not take into account the "long" tasks. In order to overcome this problem, we propose to cluster all training examples into K groups which are separated by using the completion time T as well as the feature vector $\mathbf{x}$. However, the completion time is more important in comparison with the feature vector because it determines the presented distribution of tasks. At the same time, the vector $\mathbf{x}$ should be also used. Therefore, we propose to introduce weights w_0 and w_1 of the completion time as well as the feature vector, respectively, in the clustering procedure so that $w_0 + w_1 = 1$, $w_0 > w_1$. The weighted K-means clustering procedure is used, where the distance

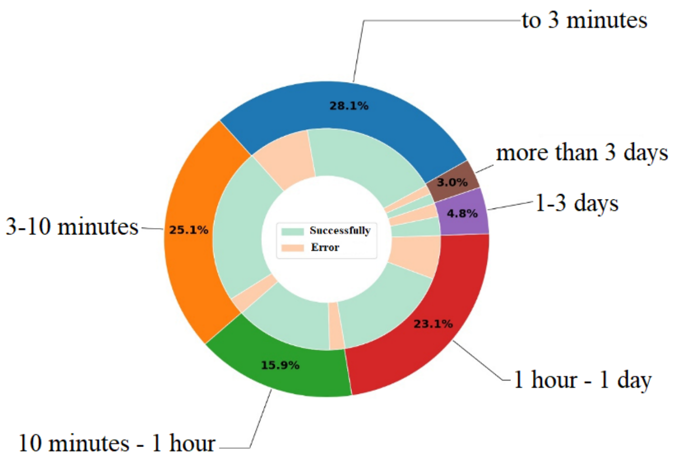


Figure 5. Distribution of tasks in accordance with their completion times

between the centroid $\mathbf{c}$ of points from a cluster C_k and the vector $\mathbf{x}$ are computed as follows:

$$dist(\mathbf{c}, \mathbf{x} \cup T) = \sqrt{w_0 \left(T - c^{(T)}\right)^2 + w_1 \left\| \mathbf{x} - \mathbf{c}^{(f)} \right\|^2}, \tag{9}$$

where $c^{(T)}$ and $\mathbf{c}^{(f)}$ are two parts of the centroid vector corresponding to the completion time and the feature vector, respectively, so that $c^{(T)} \cup \mathbf{c}^{(f)} = \mathbf{c}$.

The cluster procedure divides the dataset D into K clusters $D_1, ..., D_K$ such that $D_1 \cup ... \cup D_K = D$. Now we can train K random survival forests on K obtained datasets (clusters). In sum, we have K models for predicting the survival function. The next question is how to use these models to obtain prediction for a new task $\mathbf{x}$. We do not know the cluster for $\mathbf{x}$ and cannot determine it because we do not know the completion time T for the task. Therefore, we propose the following procedure for solving this problem. The new task is fed to K random survival forest, and we obtain K survival functions $S_1(t|\mathbf{x}), ..., S_K(t|\mathbf{x})$. It is obvious that only one of the survival functions is correct. In order to select the unique result, we propose the following approach. By having survival functions, we compute K expected times to the task completion $E_1(\mathbf{x}), ..., E_K(\mathbf{x})$, where $E_k(\mathbf{x}) = \int_0^\infty S_k(t|\mathbf{x})\mathrm{d}t$. Since the number of observations is finite, say n, and it is assumed that all times to the task completion are different, then this integral is reduced to the sum:

$$E_k(\mathbf{x}) = \sum_{i=1}^{n} S_k(t_i|\mathbf{x})(t_i - t_{i-1}),\ t_0 = 0. \tag{10}$$

Assuming that the time to the task completion for the new task $\mathbf{x}$ is $E_k(\mathbf{x})$, we obtain vectors $\mathbf{x} \cup E_k(\mathbf{x})$, $i = 1, ..., K$. A simple rule for selecting the cluster for $\mathbf{x}$ is to find the smallest distance between the obtained vector and the centroid, i.e., we select the k-th cluster and the k-th survival function $S_k(t|\mathbf{x})$ if there holds

$$k = \arg \min_{j=1,...,K} dist(\mathbf{c}_j, \mathbf{x} \cup E_j(\mathbf{x})). \tag{11}$$

In order to find the optimal or suboptimal hyperparameters, we use the C-index given in (7) or (8), which can be regarded as a measure of the model quality.

Finally, we can write a scheme of the following algorithm for training and testing the survival model and for computing the survival functions of a new task.

1. Initial data: dataset $D = \{(\mathbf{x}_i, \delta_i, T_i), i = 1, ..., n\}$. Hyperparameters: the number of random survival trees Q in the forest, the number of examples N from the dataset for constructing random trees; the number of clusters K, weights w_0 and w_1.
   Training part:
2. Divide the dataset D into two subsets: training D_{train} and testing D_{test}.
3. Divide all examples from the dataset D_{train} into K clusters $D_1, ..., D_K$ by using the K-means algorithm based on the distance metric given in (9) with weights w_0 and w_1.
4. Train K random survival forests consisting of Q trees on datasets $D_1, ..., D_K$ by randomly selecting N examples for every tree, respectively.
   Testing part:
5. Take examples $\mathbf{x}$ from the testing subset D_{test}.
6. Compute the cumulative hazard function $H_i^{(k)}(t|\mathbf{x})$ for the i-th tree from the k-th random forest, $i = 1, ..., Q$, $k = 1, ..., K$.
7. Compute the ensemble cumulative hazard function $H^{(k)}(t|\mathbf{x})$ for the k-th random forest by using (6) for all $k = 1, ..., K$.
8. Compute the survival function $S_k(t|\mathbf{x})$ from $H^{(k)}(t|\mathbf{x})$ by using (3) for all $k = 1, ..., K$.
9. Compute expected times $E_k(\mathbf{x})$ to the task completion by using (10) for all $k = 1, ..., K$.
10. Compute distances $dist(\mathbf{c}_j, \mathbf{x} \cup E_j(\mathbf{x}))$, $j = 1, ..., K$, and find k corresponding to the smallest distance.
11. Output $S_k(t|\mathbf{x})$.
12. Compute the C-index by using (7) or (8) and the testing dataset D_{test}. If the C-index does not satisfy the model requirements, then the hyperparameters are changed and go to Step 1, otherwise it is supposed that the model is successfully trained and can be used for new tasks.

5. Improving the Algorithm

One of the difficulties when the proposed algorithm is implemented is the restricted information about a user. Indeed, we did not use a history of tasks that had been previously performed by the user. This information may significantly improve the algorithm and enhance the quality of predictions.

Suppose that there are R events of the task unsuccessful termination and S events of the task successful completion when the user ran tasks. The unsuccessful terminations of the task may be due to the following reasons:

- termination of the task as a result of the intended stop when the program execution time exceeded the time requested by the user (r_1);
- termination of the task as a result of division by zero (r_2);
- termination of the task as a result of an infinite loop (r_3);
- termination of the task as a result of an unknown failure (r_4);
- termination of the task due to lack of memory (r_5).

Here $r_1, ..., r_5$ are numbers of unsuccessful terminations.

The successful completion of the task can also include the following events:

- the ratio of the program execution time and the time requested by the user does not exceed 20% (s_1);
- the ratio of the program execution time and the time requested by the user is between 20% and 40% (s_2);

- the ratio of the program execution time and the time requested by the user is between 40% and 60% (s_3);
- ...
- the ratio of the program execution time and the time requested by the user exceeds 100% (s_6).

Here $s_1, ..., s_6$ are numbers of different events of the task successful completion.

Let us construct two probability distributions $\pi = (\pi_1, ..., \pi_5)$ and $\varphi = (\varphi_1, ..., \varphi_6)$ defined as

$$\pi_i = r_i/R,\ i = 1, ..., 5,\ \varphi_j = s_j/S,\ j = 1, ..., 6.$$

The above implies that the user history can be represented by means of the above features in the form of the two distributions. Hence, we extend the feature vector $\mathbf{x}$ of a user by the probability distributions π and φ. If a new user is analyzed, we can consider the uniform probability distribution to characterize the new user. Another way for taking into account the new user is to add two additional features which take values of 0 if the user is not new and has some history of the task performance, and of 1 if the user is new. It is interesting to point out that these additional features can be in interval $[0, 1]$ indicating uncertainty of the historical observation. In particular, when the number of runs by the user is small, then values of the two additional features should be close to 1, otherwise they are close to 0. The choice of the uncertainty measure is a direction for further research.

6. Interpretation of the Machine Learning Analysis Results

One of the important problems to optimize the computer performance is to explain why the user task is characterized by the obtained survival function or the expected time to the task completion. This problem can be referred to the direction called the eXplainable Artificial Intelligence (XAI). Methods of XAI try to answer the question which features of an example significantly influence a prediction of a machine learning model. Most methods are explained locally, that is, they explain a prediction of a single example, and assume that the analyzed machine learning model is a black box which means that we know only its input and output data. A lot of the explanation methods are based on local approximating the unknown prediction function at a point by means of the linear function of features because coefficients of the function can be regarded as quantitative impacts on the prediction.

One of the first local explanation methods is the Local Interpretable Model-agnostic Explanations (LIME) [23], which uses linear model to approximate predictions of black-box models. According to LIME, the explanation is derived from a set of synthetic examples generated randomly in the neighborhood of the explained example. Every synthetic example has a weight depending on its proximity to the explained example. However, LIME cannot be applied to survival machine learning models because the output of the models is not a point, but the function (the cumulative hazard function or the survival function).

To overcome this difficulty, another method called SurvLIME (Survival LIME) has been proposed in [16]. The main idea behind SurvLIME is to approximate the black-box survival model predictions by using the Cox model. It can be seen from (4) that the hazard function in the Cox model contains the linear function of features. This implies that coefficients of the linear function can be viewed as quantitative impacts on the predicted hazard function. In other words, SurvLIME uses the Cox model as an explainable meta-model or an approximation of the cumulative hazard function or the survival function predicted for an example by the black-box model. According to SurvLIME, a set of examples $\{\mathbf{x}_1, ..., \mathbf{x}_L\}$ around the explained point $\mathbf{x}$ are generated and then fed to the black-box survival model, which produces a set of cumulative hazard functions $\{H(t|\mathbf{x}_1), ..., H(t|\mathbf{x}_L)\}$. Simultaneously, the cumulative hazard functions $H_{\text{Cox}}(t|\mathbf{x}_k, \mathbf{b})$, $k = 1, ..., L$, of the Cox model as functions of coefficients $\mathbf{b}$ are computed for all generated examples $\{\mathbf{x}_1, ..., \mathbf{x}_L\}$ by using(4). The parameters $\mathbf{b}$ of the Cox model are calculated by minimizing the average distance between the cumulative hazard functions of the Cox model and the black-box survival model. Many numerical results with using real open datasets provided in [16] demonstrate that SurvLIME is an efficient method for explaining the survival models. Moreover, following the results obtained in [16], an open-source Python package that implements the SurvLIME algorithm has been presented in [21]. Another method for explaining the survival model predictions is called SurvNAM [24]. It is based on applying of the generalized additive model $g_1(x_1) + ... + g_m(x_m)$ instead of the simple linear model. Here g_i is a univariate shape function of one variable (feature). Two ideas underlying SurvNAM are the following. First, the functions $g_i(x_i)$ are usually unknown and any assumptions about their form may lead to incorrect results. Therefore, it is proposed to implement the functions by means of simple neural networks with a single-feature input. This idea allows to implement arbitrary functions. The training weights of the networks are parameters of the explanation model. In sum, the whole model represents m fully connected neural networks implementing the functions $g_i(x_i)$, which are connected by the summation operation. The second idea is that the whole explanation neural network model implementing the generalized additive model is trained again by using the extended Cox model where the linear combination of features $\mathbf{x}\mathbf{b}^{\mathrm{T}}$ is replaced with the generalized additive model $g_1(x_1) + ... + g_m(x_m)$.

Finally, the algorithm of the explanation model SurvLIME in terms of the considered task completion problem is the following.

1. Initial data: the explained example $\mathbf{x}$; the standard deviation σ for generating the perturbed examples; the number of the generated examples.
2. Generate random examples $\mathbf{x}_1, ..., \mathbf{x}_L$ around the explained point $\mathbf{x}$ having the normal distribution with the expectation $\mathbf{x}$ and the predefined standard deviation σ.
3. Find the cumulative hazard functions $H_{k_1}(t|\mathbf{x}_1), ..., H_{k_L}(t|\mathbf{x}_L)$ as predictions of the random survival forest (the random forest). Index k_j corresponds to the optimal cluster which is selected when the generated example $\mathbf{x}_j$ is fed to the black-box model (the random forest).
4. Find the vector $\mathbf{b}_{opt}$ which minimizes the function of distances:

$$\mathbf{b}_{opt} = \arg\min_{\mathbf{b}} \sum_{j=1}^{L} dist(H_{k_j}(t|\mathbf{x}_1), H_{\text{Cox}}(t|\mathbf{x}_k, \mathbf{b})).$$

Results of the interpretation by means of SurvLIME and SurvNAM allow the user to determine which features significantly influence the survival function or the expected time to the task completion, and how the feature can be changed to obtain the reasonable survival function which ensures to complete the task in a required time with a certain probability.

7. Analysis of Tasks Statistics and Survival Functions

A structured form of the computational process metadata was obtained. It includes the parameters for launching the task, which the user has specified when have queued the task for execution to the Slurm. The considered form of data representation can be used in the machine learning methods for task schedulers.

A JSON file structure has been developed. It consists of all 89 possible attributes for tuning the sbatch SLURM command of version 21.08. Attributes, according to the custom entity, have been structured and distributed into 5 different groups: User Information, Job Accounting, Resource Management, Job Control, and Job Interaction.

The structure of the database for storing data about the tasks launched on the SCC was also proposed. It consists of three tables scontrol, sacct and sbatch, corresponding to the sources of the collected data. Each table contains an integer field ID of the task, as well as two fields of the VARCHAR(500) type, which allows us to dynamically allocate memory for the size of the attribute loaded into it, not exceeding 500 characters. These two fields allow to specify a key-value pair for an attribute and its corresponding value.

A special program was also developed for extracting, lexical processing and sending task data to the developed database. The program has been tested on a laboratory stand by running test tasks. As a result, 60 attributes are collected in the scontrol table for each task, 107 attributes are collected in the sacct table, and in the sbatch table as many records are stored as the number of parameters specified by the user during the startup script. The user, on average, specifies no more than 8 parameters in the startup script. In total, for each task ID, the database contains about 180 parameters, while some fields may be empty if Slurm was unable to obtain information on these attributes.

A program for the collection of statistical data of the SCC was developed. The program is called by the event of allocation of resources for the task, as well as by the event of the completion of the task. Thus, at the output of the program, two corresponding databases are formed. The running time for each call is about 4–5 seconds due to waiting for requests from the SCC control computer to the remote database server, so the program runs in the background so as not to cause any delay in the scheduler. As a result of the work of the statistics collection program, more than 400,000 tasks from users were collected in 2022, and more than 200,000 tasks were collected in 2023.

To solve the problem of optimizing the structure of the queue, machine learning models have been developed and implemented to predict one of the parameters of the task: its execution time, taking into account the current load of the SCC nodes. During the analysis of the data, the following problems were identified:

- a large spread of the estimated execution time value (from several seconds up to two weeks);
- a strong imbalance of tasks across the ranges to which they belong (more than 53 percent are tasks that take less than 10 minutes);
- insufficient amount of information in the available factors on which the target value is estimated.

Various machine learning models have been implemented, including regression models, data classification and clustering models, and survival models, which also allow the use of data on tasks that were forced to complete by Slurm.

Figure 6 shows the error matrix based on user ratings, normalized by the number of tasks falling within each range. The matrix shows that more than 89 percent of all tasks with an actual duration of [0, 10] minutes indicate an approximate execution time from 1 hour to 15 days. In most cases, users significantly overestimate the time it takes to complete tasks.

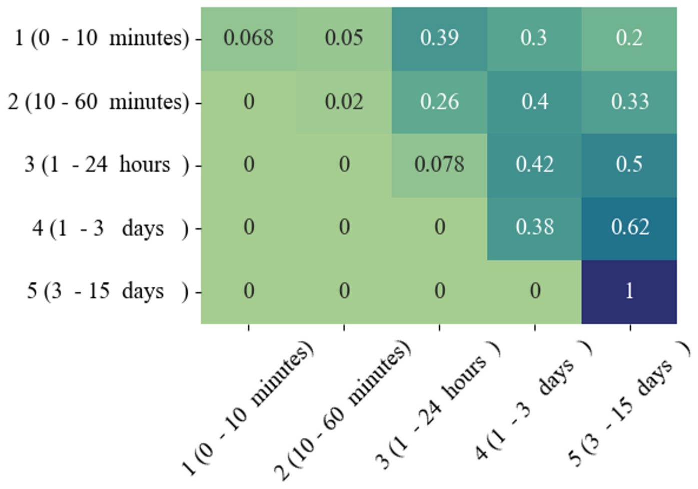


Figure 6. Error matrix based on user ratings, normalized by the number of tasks falling within each range

Users behavior studies were conducted to assess the impact of these data on the tasks. Users and virtual users of geovation were separated due to significant differences in their data. The following estimations were received:

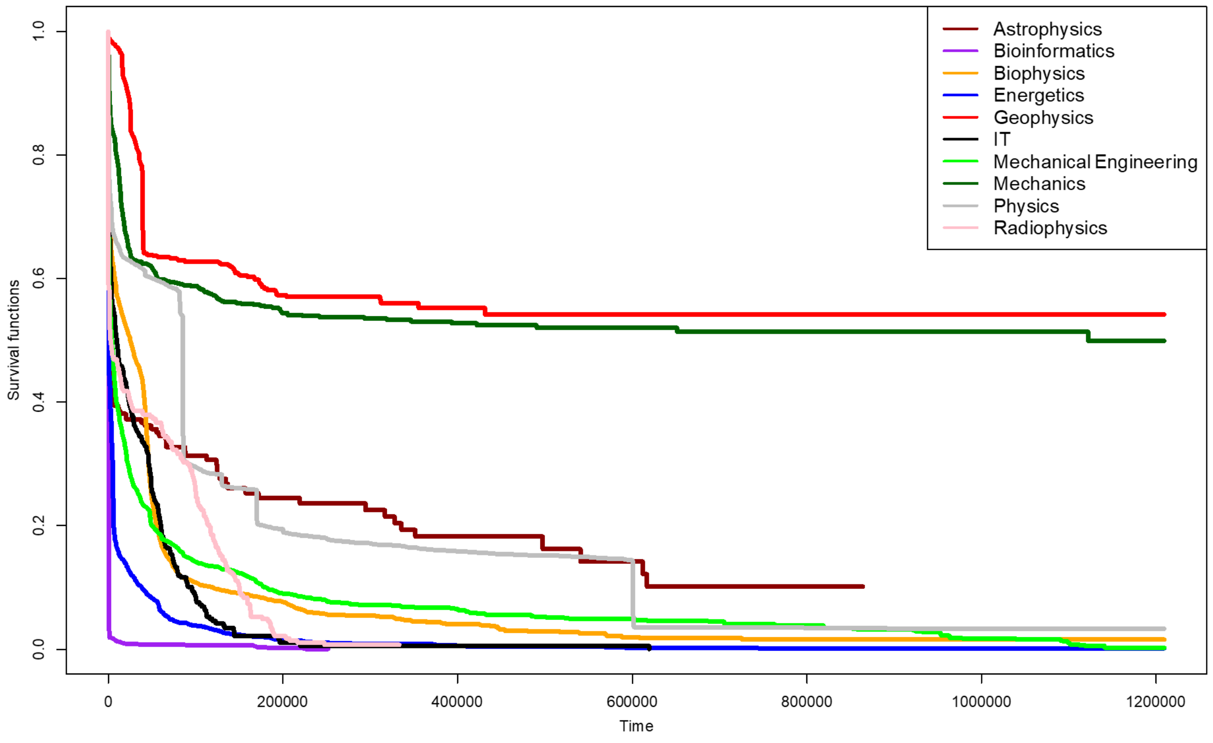
- more than 90 percent of users posted less than 100 tasks;
- more than 60 percent of virtual users of geovation posted from 30 to 60 thousand tasks and two posted more than 100 thousand tasks;
- more than 64 percent of users completed tasks in a very short time (less than 10 minutes);
- more than 92 percent of non-geovation users used just one compute node for their tasks.

The information obtained during the research showed that some of the tasks are executed exactly at the time that the user set and after that Slurm forcibly terminated them. So there is a set of tasks whose real execution time is not known, since they were forcibly terminated before they were completed. Survival models are used to work with such data. Kaplan–Meier model was used to estimate the distribution of the required time and processor time for tasks in various areas of knowledge (Fig. 7). Visual analysis of the obtained estimations of survival functions in various fields of knowledge allows to establish that tasks in the fields of geophysics and mechanics require stochastically more time to be successfully completed than tasks from other fields of knowledge, and the shortest tasks in terms of processor time usage are typical for bioinformatics.

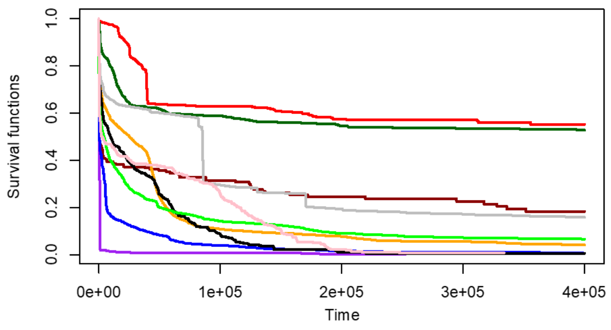
Conclusion

Obviously, an NLP outline of the description of applied tasks can be added to the machine learning ecosystem so that the user can conduct a dialogue with the supercomputer in terms of meaningful queries in the context of applied tasks. Using the capabilities of a pre-trained transformer (Fig. 8), which generates the source code of the description of the applied task in response to the meaningful request, the user can analyze the accuracy of the formulated queries and in a recursive mode, conducting a meaningful interaction with a supercomputer, that inevitably improves his qualifications and task understanding.

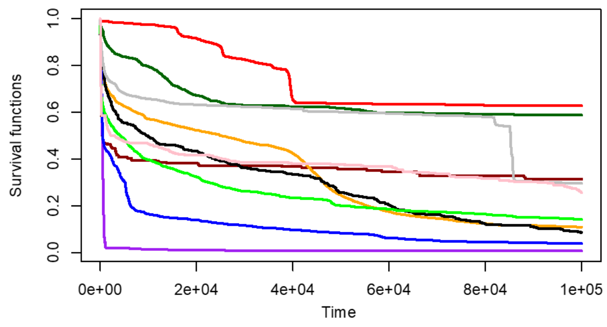
Let us point out several perspective directions for further research and development.



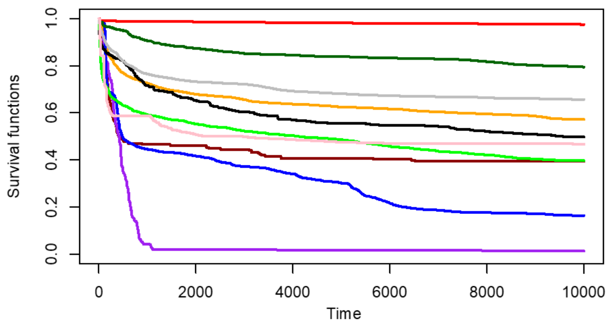
(a) $1.2 * 10^6$ seconds



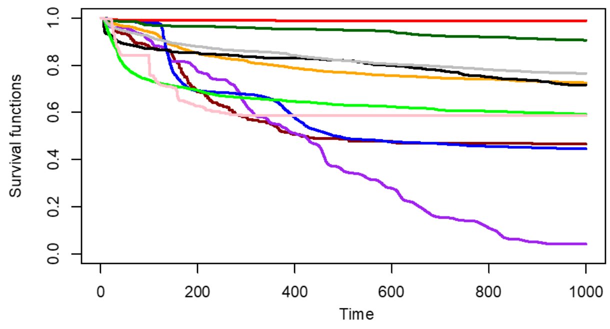
(b) $4 * 10^5$ seconds



(c) $1 * 10^5$ seconds



(d) $1 * 10^4$ seconds



(e) $1 * 10^3$ seconds

Figure 7. Kaplan–Meier estimations of task execution time distributions in various fields of knowledge with detailed area of task execution times not exceeding

We have proposed the specific weighted scheme of clustering, which allows us to take into account the difference between predicted times of the task completion as well as the difference between the feature vectors. However, a more interesting approach is to assign weights to each feature. Moreover, these weights should not be hyperparameters, but they have to be trained

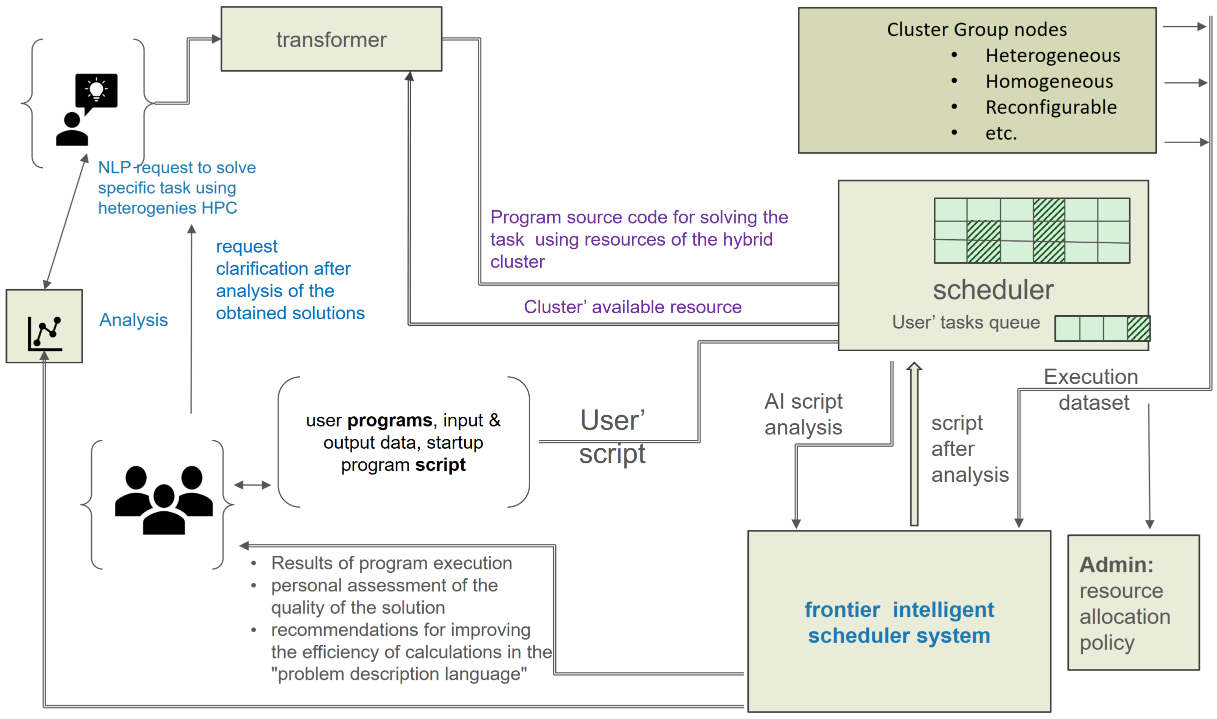

transformer
Cluster Group nodes
Heterogeneous
Homogeneous
Reconfigurable
etc.
NLP request to solve specific task using heterogenies HPC
Program source code for solving the task using resources of the hybrid cluster
request clarification after analysis of the obtained solutions
Analysis
Cluster' available resource
scheduler
User' tasks queue
Execution dataset
AI script analysis
User' script
script after analysis
user programs, input & output data, startup program script
Results of program execution
personal assessment of the quality of the solution
recommendations for improving the efficiency of calculations in the "problem description language"
frontier intelligent scheduler system
Admin: resource allocation policy

Figure 8. Proposal for the integration of reconfigurable nodes under the control of a multi-agent scheduler into the SCC "Polytechnic"

during the training of the whole system. This approach has two advantages. First, it allows us to make the clustering procedure to be more flexible and enhance the prediction quality. Second, it allows us to supplement the explanation procedure. The obtained weights of features show the importance of features and answer the question of which features are important from the clustering point of view. However, in order to implement the training of weights, the random survival forests have to be replaced with neural networks to train in an end-to-end manner. This requirement leads to the second perspective which is to develop a transformer to solve the survival problems and process multi-modal data. By collecting training data, we obtain the opportunity to totally or partially avoid the random forests and to construct the neural network models. However, the idea of the task description in terms of the natural language requires the development of more complex and efficient structures based on the attention mechanism and transformers. It should be noted that transformers have been proposed to solve the survival analysis problems [8, 28]. However, these transformers do not take the peculiarities of the problems which are solved when the task completion time optimization problem is solved. Another interesting idea is to combine the random survival forests and the transformer. It has been partially solved for original random forests in [15]. However, this approach cannot be directly used in survival analysis. New approaches are needed to develop an efficient multi-modal transformer-based system.

It is important to note that one of the perspective directions is to adapt the trained system to changes in the supercomputer structure, for example, to new additional computer blocks which can be supplemented in some time. In this case, we cannot directly use the trained model and need to re-train the whole prediction system. In order to avoid that, we propose to apply ideas of the heterogeneous treatment effect [1, 17] or transfer learning [18, 29]. These approaches allow us to do hard computations for training the system when the structure of the supercomputer has been changed.

"A network is a computer", a slogan that had originally been used by Sun Microsystems in the early 1980s, became a basic truism of computer science: "computers must be networked, otherwise they are... not computers". Today we propose two new extensions of former slogan, namely "A frontier ML system is HPC", and vice versa "HPC is a frontier ML system". These two slogans have not become truism yet but they clearly reflect ideas of our article and obvious computer evolution tendency – frontier high-performance computing systems become not only a driving force of global digital transformation process, but also active part of machine learning ecosystem.

Acknowledgements

The research is partially funded by the Ministry of Science and Higher Education of the Russian Federation as part of state assignments "Development of a Multi-Agent Resource Manager for a Heterogeneous Supercomputer Platform Using Machine Learning and Artificial Intelligence" (topic FSEG-2022-0001).

The results of the work were obtained using computational resources of the supercomputer center in Peter the Great Saint-Petersburg Polytechnic University (`https://scc.spbstu.ru`).

References

1. Alaa, A., van der Schaar, M.: Limits of estimating heterogeneous treatment effects: Guidelines for practical algorithm design. In: Proceedings of the International Conference on Machine Learning, pp. 129–138. PMLR (2018)

2. Bou-Hamad, I., Larocque, D., Ben-Ameur, H.: A review of survival trees. Statistics Surveys 5, 44–71 (2011). `https://doi.org/10.1214/09-SS047`

3. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001). `https://doi.org/10.1023/A:1010933404324`

4. Cox, D.: Regression models and life-tables. Journal of the Royal Statistical Society, Series B (Methodological) 34(2), 187–220 (1972). `https://doi.org/10.1111/j.2517-6161.1972.tb00899/x`

5. Faraggi, D., Simon, R.: A neural network model for survival data. Statistics in Medicine 14(1), 73–82 (1995). `https://doi.org/10.1002/sim.4780140108`

6. Harrell, F., Califf, R., Pryor, D., *et al.*: Evaluating the yield of medical tests. Journal of the American Medical Association 247, 2543–2546 (1982). `https://doi.org/10.1001/jama.1982.03320430047030`

7. Hosmer, D., Lemeshow, S., May, S.: Applied Survival Analysis: Regression Modeling of Time to Event Data. John Wiley & Sons, New Jersey (2008) `https://doi.org/10.1007/s00362-010-0360-3`

8. Hu, S., Fridgeirsson, E., van Wingen, G., Welling, M.: Transformer-based deep survival analysis. In: Survival Prediction-Algorithms, Challenges and Applications, pp. 132–148. PMLR (2021)

9. Ishwaran, H., Kogalur, U.: Random survival forests for R. R News 7(2), 25–31 (2007). `https://doi.org/10.1214/08-AOAS169`

10. Ishwaran, H., Kogalur, U., Blackstone, E., Lauer, M.: Random survival forests. Annals of Applied Statistics 2, 841–860 (2008). `https://doi.org/10.1214/08-AOAS169`

11. Kalyaev, A., Kalyaev, I., Khisamutdinov, M., *et al.*: An effective algorithm for multiagent dispatching of resources in heterogeneous cloud environments. In: 5th International Conference on Informatics, Electronics and Vision (ICIEV), pp. 1140–1142. IEEE (2016). `https://doi.org/10.1109/ICIEV.2016.7760177`

12. Kalyaev, I.A., Kalyaev, A.I. Method and Algorithms for Adaptive Multiagent Resource Scheduling in Heterogeneous Distributed Computing Environments. Autom Remote Control 83, 1228–1245 (2022). `https://doi.org/10.1134/S0005117922080069`

13. Katzman, J., Shaham, U., Cloninger, A., *et al.*: Deepsurv: Personalized treatment recommender system using a Cox proportional hazards deep neural network. BMC Medical Research Methodology 18(24), 1–12 (2018). `https://doi.org/10.1186/s12874-018-0482-1`

14. Khan, F., Zubek, V.: Support vector regression for censored data (SVRc): a novel tool for survival analysis. In: 2008 Eighth IEEE International Conference on Data Mining. pp. 863–868. IEEE (2008). `https://doi.org/10.1109/ICDM.2008.50`

15. Konstantinov, A., Utkin, L., Lukashin, A., Muliukha, V.: Neural attention forests: Transformer-based forest improvement (Apr 2023), arXiv:2304.05980. `https://doi.org/10.48550/arXiv.2304.05980`

16. Kovalev, M., Utkin, L., Kasimov, E.: SurvLIME: A method for explaining machine learning survival models. Knowledge-Based Systems 203, 106164 (2020). `https://doi.org/10.1016/j.knosys.2020.106164`

17. Kunzel, S., Stadie, B., Vemuri, N., *et al.*: Transfer learning for estimating causal effects using neural networks (Aug 2018), arXiv:1808.07804. `https://doi.org/10.48550/arXiv.1808.07804`

18. Lu, J., Behbood, V., Hao, P., *et al.*: Transfer learning using computational intelligence: A survey. Knowledge-Based Systems 80, 14–23 (2015). `https://doi.org/10.1016/j.knosys.2015.01.010`

19. May, M., Royston, P., Egger, M., *et al.*: Development and validation of a prognostic model for survival time data: application to prognosis of HIV positive patients treated with antiretroviral therapy. Statistics in Medicine 23, 2375–2398 (2004). `https://doi.org/10.1002/sim.1825`

20. Nezhad, M., Sadati, N., Yang, K., Zhu, D.: A deep active survival analysis approach for precision treatment recommendations: Application of prostate cancer. Expert Systems with Applications 115, 16–26 (2019). https://doi.org/10.1016/j.eswa.2018.07.070

21. Pachon-Garcia, C., Hernandez-Perez, C., Delicado, P., Vilaplana, V.: SurvLIMEpy: A Python package implementing SurvLIME (Feb 2023), arXiv:2302.10571. https://doi.org/10.48550/arXiv.2302.10571

22. Polsterl, S., Navab, N., Katouzian, A.: An efficient training algorithm for kernel survival support vector machines (Nov 2016), arXiv:1611.07054v. https://doi.org/10.48550/arXiv.1611.07054

23. Ribeiro, M., Singh, S., Guestrin, C.: "Why should I trust You?" Explaining the predictions of any classifier In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144. ACM (2016). https://doi.org/10.1145/2939672.2939778

24. Utkin, L., Satyukov, E., Konstantinov, A.: SurvNAM: The machine learning survival model explanation. Neural Networks 147, 81–102 (2022). https://doi.org/10.1016/j.neunet.2021.12.015

25. Waititu, H., Koske, J., Onyango, N.: Analysis of balanced random survival forest using different splitting rules: Application on child mortality. International Journal of Statistics and Applications 11(2), 37–49 (2021). https://doi.org/10.5923/j.statistics.20211102.03

26. Wang, H., Zhou, L.: Random survival forest with space extensions for censored data. Artificial Intelligence in Medicine 79, 52–61 (2017). https://doi.org/10.1016/j.artmed.2017.06.005

27. Wang, P., Li, Y., Reddy, C.: Machine learning for survival analysis: A survey. ACM Computing Surveys (CSUR) 51(6), 1–36 (2019). https://doi.org/10.1145/3214306

28. Wang, Z., Sun, J.: SurvTRACE: Transformers for survival analysis with competing events. In: Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, pp. 1–9. ACM (2022). https://doi.org/10.1145/3535508.3545521

29. Weiss, K., Khoshgoftaar, T., Wang, D.: A survey of transfer learning. Journal of Big Data 3(1), 1–40 (2016). https://doi.org/10.1186/s40537-016-0043-6

DOI: 10.14529/jsfi230208

The High Performance Interconnect Architecture for Supercomputers

Alexey S. Simonov[1,2], ***Alexander S. Semenov***[1], ***Andrey N. Shcherbak***[1], ***Ivan A. Zhabin***[1]

In this paper, we introduce the design of an advanced high-performance interconnect architecture for supercomputers. In the first part of the paper, we consider the first generation high-performance Angara interconnect (Angara G1). The Angara interconnect is based on the router ASIC, which supports a 4D torus topology, a deterministic and an adaptive routing, and has the hardware support of the RDMA technology. The interface with a processor unit is PCI Express. The Angara G1 interconnect has an extremely low communication latency of 850 ns using the MPI library, as well as a link bandwidth of 75 Gbps. In the paper, we present the scalability performance results of the considered application problems on the supercomputers equipped with the Angara G1 interconnect. In the second part of the paper, using research results and experience we present the architecture of the advanced interconnect for supercomputers (G2). The G2 architecture supports 6D torus topology, the advanced deterministic and zone adaptive routing algorithms, and a low-level interconnect operations including acknowledgments and notifications. G2 includes support for exceptions, performance counters, and SR-IOV virtualization. A G2 hardware is planned in the form factor of a 32-port switch with the QSFP-DD connectors and a two-port low profile PCI Express adapter. The switches can be combined to 4D torus topology. We show the performance evaluation of an experimental FPGA prototype, which confirm the possibility of implementing the proposed advanced high performance interconnect architecture.

Keywords: interconnect, high performance computing, supercomputer, Angara.

Introduction

An interconnection network (interconnect) is a critical supercomputer component to achieve high scalability and performance on different applications. Modern supercomputer applications include mathematical physics, business analytics and machine learning problems.

Analysis of world experience in designing custom interconnection networks for high-end supercomputers, primarily the IBM BlueGene L/P/Q series [6, 8, 11] and Cray SeaStar/Gemini [1, 5, 7], simulation results [20] allowed to design the principles of operation of the first generation of the high-performance Angara interconnect (Angara G1) [21]. Series-produced Angara G1 hardware was presented in 2013. The Angara G1 interconnect is a direct network, it supports topologies from 1D-mesh to 4D-torus and provides the possibility of building a supercomputer with a size of up to 32K nodes.

The operation experience of existing high-performance interconnect solutions allows to develop architecture features and design the principles of operation of an advanced high-performance interconnection network for supercomputers.

The paper is structured as follows. Section 1 describes the architecture and the performance results of the Angara G1 interconnect. Section 2 presents the architecture and features of the advanced high-performance interconnect for supercomputers, and provides the performance evaluation obtained on a FPGA prototype of the proposed interconnect architecture. Conclusion summarizes the paper and points directions for further work.

[1]Federal State Unitary Enterprise “Russian Federal Nuclear Center-Zababakhin All – Russia Research Institute of Technical Physics”, Snezhinsk, Russia
[2]Federal State Budgetary Educational Institution of Higher Education Moscow Aviation Institute (National Research University), Moscow, Russia

1. The First Generation Angara Interconnect

The choice of the interconnect topology determines the range of applicable routing algorithms and methods for solving the main problems of communication networks: deadlocks, starvation and livelocks. As a research result, the following decisions were made on the Angara G1 interconnect architecture:

- torus topology;
- the deterministic directional order routing algorithm with a fixed order of directions [10], a direction bit (dirbit) routing rule does not allow both positive and negative directions of a dimension in a route. There is a possibility of the first and last steps of a route in an arbitrary positive and negative directions for bypassing failed nodes and links [18];
- the bubble routing method [2, 16] is used to avoid deadlocks in a ring (movement without changing a direction). The direction order routing [9] garantees no deadlocks in torus directions. The first and last steps can violate the direction order rule, the routing table generation algorithm provides deadlock freedom [12]. The adaptive routing has the ability by a timeout to switch to a nonblocking deterministic virtual channel in case of potential deadlock;
- a minimal full-adaptive routing algorithm with assignment of possible directions;
- virtual subnets for deterministic, adaptive and broadcast routing based on virtual channels;
- a virtual cut-through flow control method;
- the used routing algorithms are minimal, i.e., each step of a packet route reduces the distance to a destination node. Minimal routing provides livelock freedom;
- fair arbitrage algorithms allow to avoid starvation;
- the PCI Express interface with CPU;
- a network adapter at the hardware level supports remote direct memory access (RDMA) write, read, and atomic operations;
- hardware and software support for the main functions of the SHMEM and MPI libraries;
- each node has a dedicated memory region available for remote access from other nodes to support, OpenSHMEM and partitioned global address space (PGAS);
- router implementation in the form of custom ASIC.

On the basis of custom designed router ASIC the following options of the Angara G1 interconnect hardware were released:

- a full-height full-length PCI Express adapter, which provides the ability to combine directly up to 32 thousand computing nodes of a supercomputer into a topology up to 4D torus topology $8 \times 16 \times 16 \times 16$;
- a 24-port switch for installation in a 19-inch rack and a low profile PCI Express adapter for installation in supercomputer nodes. This option provides the ability to combine up to 2048 computing nodes with 2D torus topology of switches.

1.1. Experimental Results and Performance Evaluation

The first computing system equipped with the Angara G1 interconnect is the Angara-C1 36-node computing cluster [3], designed for hardware and system software development of the interconnect.

During the development, different options for combining computing nodes into a torus topology were tested, system software was debugged, including the Slurm job scheduling system, a plugin for the Zabbix monitoring system. Figure 1 shows the Angara G1 system software stack.

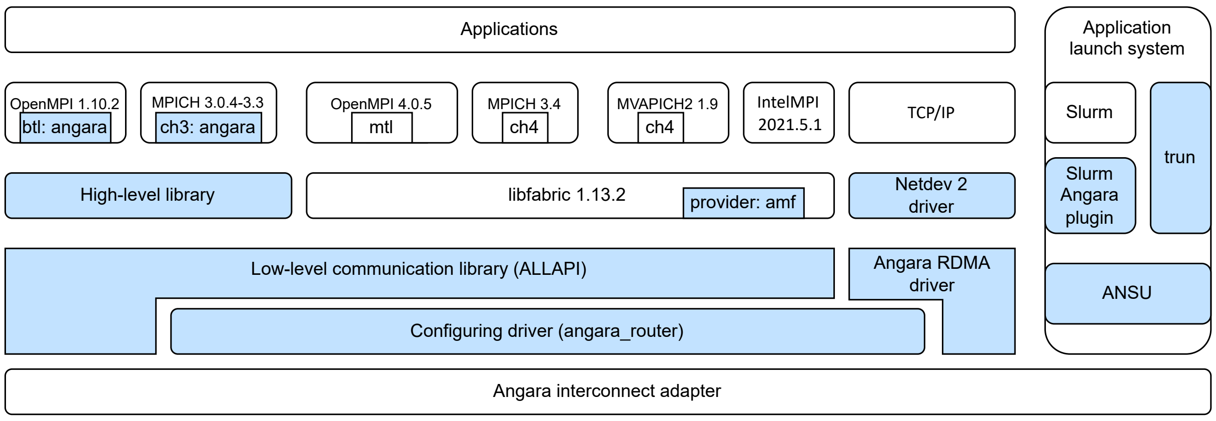


Figure 1. The Angara G1 system software stack

A supercomputer series of difference performance based on the Angara G1 interconnect were produced, including the Desmos [22] and Fisher [19, 23] supercomputers installed at the JIHT RAS.

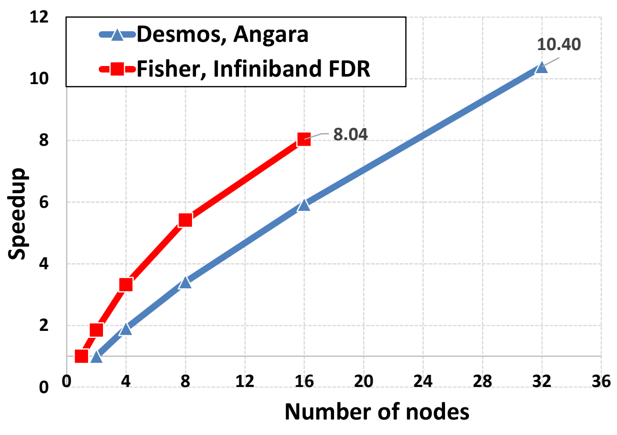
Several years of operation and maintenance of supercomputers based on the the Angara G1 interconnect made it possible to identify both the advantages and disadvantages of Angara G1. The advantages include:

- extremely low communication latency, for example on Desmos the obtained latency is 850 ns using the MPI library on the osu_latency benchmark. In comparison with other commercially available communication networks, on Angara G1 the latency is stable with a network load increase and with an increase of the supercomputer node number;
- 75 Gbit/s link bandwidth, which does not change with increasing distance between supercomputer nodes. This feature is ensured both by the chosen topology and by a good balance of the stages of the entire Angara G1 pipeline.

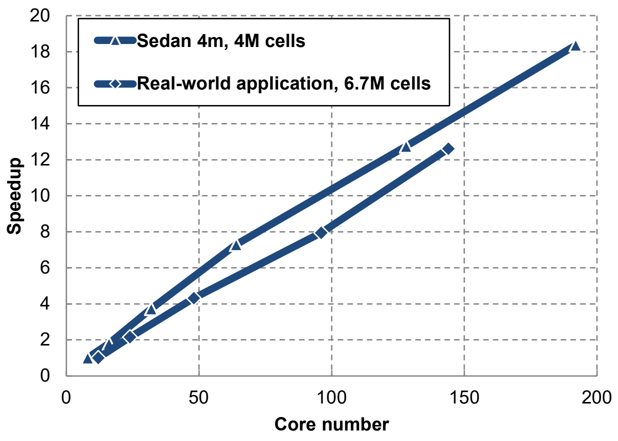
The mentioned advantages made it possible to obtain good performance scalability of benchmarks and applications on the supercomputers based on the Angara G1 interconnect [4, 13–15, 17, 22–24]. Figure 2 presents performance scalability obtained on the Desmos, Fisher and Angara C1 supercomputers on quantum mechanical calculation software called VASP and a fluid simulation software.

The main disadvantages of the Angara G1 interconnect are the following:

- network connectivity degradation in case of computing node and communication cable failure, associated with the limited capabilities of the routing algorithms for bypassing the failed network resources;
- supercomputer performance degradation in the situation of interconnect congestions, associated with the features of the deterministic and adaptive routing algorithms;
- insufficient hardware support of fault tolerance in the router ASIC;
- insufficient hardware support of modern multi-core processors;
- insufficient hardware support for GPU integration;
- lack of hardware support of the TCP/IP protocol stack;
- lack of hardware support of virtualization;



(a) Quantum mechanical calculations VASP, Desmos



(b) A fluid simulation software, Angara-C1

Figure 2. Performance result scaling of the Angara based supercomputers on the simulation application softwares

- insufficient hardware support for supercomputer monitoring and control systems, which is especially important for high-end large supercomputers;
- large physical dimensions of the full-height full-length PCI Express Angara G1 adapter;
- limitation on the number of supercomputer nodes when using the Angara G1 24-port switch and the low profile PCI Express adapter.

These disadvantages were taken into account when developing an advanced high-performance interconnect architecture.

2. Advanced Interconnect Architecture

An advanced G2 high-performance interconnect is designed to combine computing nodes of supercomputers up to a subexascale performance level. During its development, technical risks were minimized, the advantages and disadvantages of the Angara G1 interconnect were taken into account. In addition to the goal of ensuring high performance and high scalability of supercomputers built using the developed interconnect, the goal was to extend the product segments of the interconnect. The intended product segments are not only supercomputers, but also storage systems, business analytic processing systems and data centers.

To ensure the scalability of application performance and support new product areas, it was necessary to:

- use more flexible routing algorithms;
- have low communication latency;
- increase the message rate;
- extend the interconnect functional capabilities and features.

2.1. Interconnect Architecture

The essential differences between the G2 interconnect architecture and the existing solution are the following:

- flexible options for combining available topologies from 1D mesh to 6D torus, topologies with shifted connections are also supported;

- an advanced deterministic delta routing algorithm that provides more route options when bypassing failed or congested network sections;
- an advanced zone-adaptive routing algorithm;
- an extended low-level network operation set, including acknowledgements, notifications and synchronization operations;
- multihost hardware support, that allows to divide the PCI Express interconnect adapter interface into 2 or 4 independent interfaces and connecting different computing nodes via them;
- hardware support for modern multicore processors, 128 independent injection pipelines are implemented;
- hardware methods to solve the starvation network problem;
- advanced fault tolerance support, as well as monitoring and control systems for supercomputers, including support for exceptions, traps and performance counters;
- hardware support for SR-IOV virtualization, up to 15 virtual functions are supported in each PCI Express endpoint;
- hardware support for hot swap of failed interconnect hardware.

The most important difference between the G2 interconnect and the existing solution is 6D torus topology. This architecture feature follows the global trend of high-radix communication networks to increase the connectivity of each node or switch.

The advanced delta routing algorithm implements the dimension order routing rule. In the dirbit routing algorithm implemented in the Angara G1 interconnect a packet header stores not only a target node address, but also a bit that specify a torus direction, in which a packet will move. In contrast to the dirbit routing algorithm, in the delta routing algorithm a packet header for each torus coordinate stores only the difference between the current and a target node, which allowed to reduce bit number in the packet header. Also, the delta routing algorithm is more flexible and allows you to transmit a packet both in a positive and in a negative torus direction, and by a large number of steps, which allows to significantly extend the possibilities of bypassing failed supercomputer nodes or communication cables. In addition, the delta routing algorithm allows to build shifted connections in a torus topology, which in some cases can reduce the network diameter and improve communication latency.

The second important difference is the possibility of a zone-adaptive routing. In the Angara G1 interconnect, when a packet moves using adaptive routing, in case of, for example, congestion or failure, the packet waits too long for a connection, the packet is transferred to the deterministic subnet, and it is impossible to return the packet back to movement with the adaptive routing. The zone-adaptive routing allows to specify in which directions a packet can be transmitted using an adaptive algorithm, and in which directions using the deterministic algorithm. Thus, the packet movement using the deterministic algorithm will be used only in hyperplanes in which there are problems, for example, failed computing nodes or communication channels.

The nomenclature of low-level G2 operations includes simple put, get, and atomic operations (including returning a value) in a memory of a remote node. In addition to the mentioned operations there are write to a segment operations, moreover three types of segments are available: number of incoming packets, aggregating segments and circular segments. Also there are operations with acknowledgements and notifications, including interrupts. The use of these types of operations allows to optimize the implementation of low-level software of the MPI and TCP/IP libraries, reduces the load on CPU and significantly improves the application perfor-

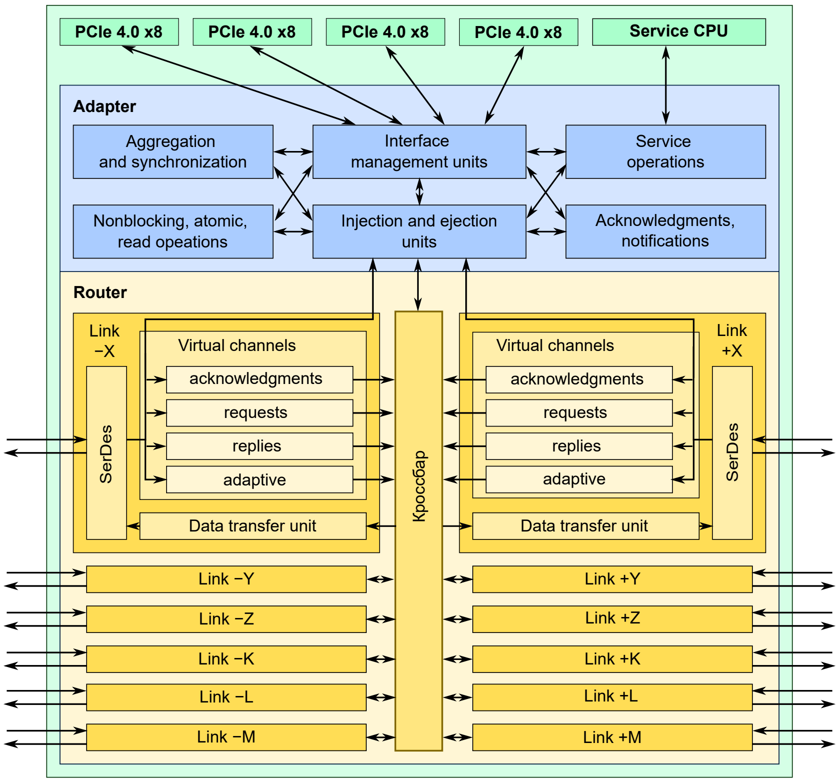


Figure 3. The G2 interconnect architecture

mance and scalability. The G2 interconnect supports the technology of direct memory access to the remote node (RDMA), which significantly affects the obtained performance not only in the high-performance computing systems, but also in storage and Big Data processing systems.

Figure 3 presents the G2 interconnect architecture, which consists of two parts: a router and an adapter. The router is intended for routing and transmission of packets between nodes, includes four virtual channels in each link, including: three deterministic virtual channels for request, response and confirmations subnets and an adaptive subnet. Deterministic virtual channels implement the deterministic delta routing algorithm with dimension-order routing and an order of dimensions can be set at system startup.

The adapter performs the injection of packets into the network and the processing of packets ejected from the network, as well as a set of service functions that ensure the minimization of traffic over the network and through the interface to the node processor. The adapter includes:

- interface control units, which support PCI Express protocols and implement virtual functions;
- injection and ejection units, that form packets for sending to the network and parse the headers of packets that came from the network;
- a service unit, that processes packets going to and from the service processor;
- aggregation and synchronization units, as well as confirmation and notification units, which support the execution of different network operations;
- non-blocking, atomic and read operation units, which support RDMA technology.

The G2 interconnect hardware is planned to be released in the form factor of a 32-port switch for installation in a standard 19-inch rack and a low profile PCI Express adapter for installation in supercomputer nodes. This option provides the possibility of combining up to several thousand computing nodes by connecting switches in a topology from 1D to 4D torus.

Each switch will include 32 ports with QSFP-DD connectors, and the low profile adapter will include 2 ports with QSFP-DD connectors and support multihost operation of several PCI Express interfaces.

2.2. Prototype Performance Evaluation

Development of the underlying architectural solutions of G2 was on a simulation model, as well as on a cluster with prototype network adapters based on the Virtex7 FPGA, each computing node includes 2x Intel Xeon CPU E5-2630 v3 processors @ 2.40 GHz, 32 GB of memory.

A prototype network adapter is made in the form factor of a full-length full-height PCI Express expansion card. Due to the technical limitations of the FPGA, PCI Express 2.0 x8 was used as an interface with CPU, there are four independent injection packet pipelines and 8 QSFP connectors were used for communication with other adapters.

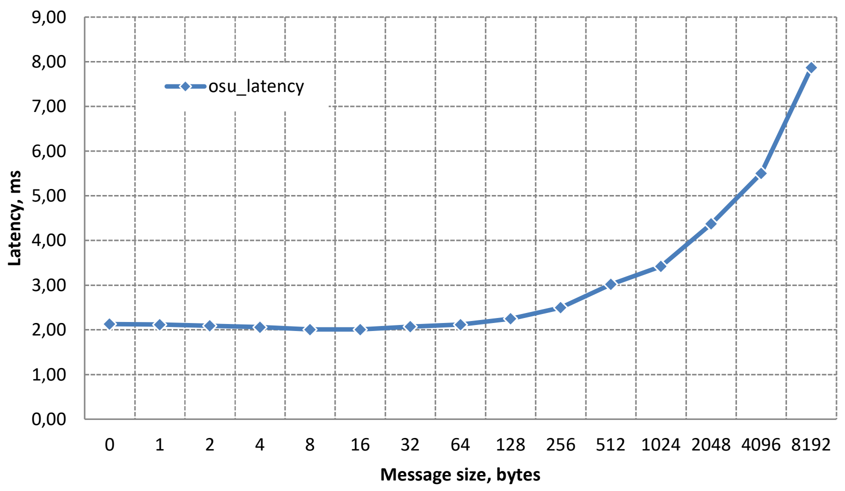


Figure 4. The osu_latency results obtained on two nodes of the M4 prototyping cluster

The frequency of the prototype adapter is 167 MHz, we obtained 2.01 us communication latency between two neighboring nodes on the osu_latency test using the MPI library. Figure 4 presents osu_latency results for different message sizes. The latency for each additional hop is no more than 0.5 us. The 30 Gbit/s bandwidth of the prototype adapter is limited by the performance of packet injection into the network, the overhead of the G2 interconnect data transmission protocol when transmitting large messages is less than 9% of the peak bandwidth of the communication channel.

Conclusion

In this paper, we introduce the main features of the advanced high-performance interconnect architecture for supercomputers, including topology, adaptive and deterministic routing algorithms, remote direct memory access technology.

We have considered the first generation high-performance Angara interconnect (Angara G1). Angara G1 has an extremely low communication latency of 850 ns, as well as a link bandwidth of 75 Gbps. However, it has insufficient hardware support for GPU integration, TCP/IP protocol stack, virtualization, for supercomputer monitoring and control systems, which is especially important for high-end large supercomputers.

Based on the performance evaluation and operation of the high-performance computing systems built with the Angara G1 interconnect solution, we present the main architectural features for the advanced interconnect for supercomputers. The proposed G2 architecture supports 6D torus topology, the advanced deterministic delta routing and zone adaptive routing algorithms, and extended low-level interconnect operations. G2 includes support for exceptions, performance counters, and the SR-IOV virtualization technology. The G2 hardware is planned in the form factor of a 32-port switch with the QSFP-DD connectors and a two-port low profile PCI Express adapter. We showed the performance evaluation of the experimental FPGA prototype. In future works, we plan to present more detailed performance evaluation of the FPGA prototype.

Acknowledgements

This work is supported by the Ministry of Industry and Trade of the Russian Federation under the complex project, which is performed by FSUE "VNIITF named after Academ. E. I. Zababakhin".

References

1. Abts, D.: The Cray XT4 and Seastar 3D torus interconnect (2011)

2. Adiga, N.R., Blumrich, M.A., Chen, D., *et al.*: Blue Gene/L torus interconnection network. IBM Journal of Research and Development 49(2.3), 265–276 (2005). https://doi.org/10.1147/rd.492.0265

3. Agarkov, A., Ismagilov, T., Makagon, D., *et al.*: Performance evaluation of the Angara interconnect. In: Proc. Int. Conf. on Russian Supercomputing Days, Moscow, Russia. pp. 626–639 (2016). https://russianscdays.org/files/pdf16/626.pdf, accessed: 2023-05-15 (in Russian)

4. Akimov, V., Silaev, D., Aksenov, A., *et al.*: FlowVision scalability on supercomputers with Angara interconnect. Lobachevskii Journal of Mathematics 39, 1159–1169 (2018). https://doi.org/10.1134/S1995080218090081

5. Alam, S.R., Kuehn, J.A., Barrett, R.F., *et al.*: Cray XT4: an early evaluation for petascale scientific simulation. In: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing. pp. 1–12 (2007). https://doi.org/10.1145/1362622.1362675

6. Almasi, G., Asaad, S., Bellofatto, R.E., *et al.*: Overview of the IBM Blue Gene/P project. IBM Journal of Research and Development 52(1-2), 199–220 (2008). https://doi.org/10.1147/rd.521.0199

7. Alverson, R., Roweth, D., Kaplan, L.: The Gemini system interconnect. In: 2010 18th IEEE Symposium on High Performance Interconnects. pp. 83–87. IEEE (2010). https://doi.org/10.1109/HOTI.2010.23

8. Chen, D., Eisley, N., Heidelberger, P., *et al.*: The IBM Blue Gene/Q interconnection fabric. IEEE Micro 32(1), 32–43 (2011). https://doi.org/10.1109/MM.2011.96

9. Dally, W.J., Seitz, C.L.: The torus routing chip. Distributed computing 1(4), 187–196 (1986). https://doi.org/10.1007/BF01660031

10. Duato, J., Yalamanchili, S., Ni, L.: Interconnection networks. Morgan Kaufmann (2003)

11. Gara, A., Blumrich, M.A., Chen, D., *et al.*: Overview of the Blue Gene/L system architecture. IBM Journal of research and development 49(2.3), 195–212 (2005). https://doi.org/10.1147/rd.492.0195

12. Mukosey, A., Simonov, A., Semenov, A.: Extended routing table generation algorithm for the Angara interconnect. In: Russian Supercomputing Days. pp. 573–583. Springer (2019). https://doi.org/10.1007/978-3-030-36592-9_47

13. Nikolskiy, V., Pavlov, D., Stegailov, V.: State-of-the-art molecular dynamics packages for GPU computations: Performance, scalability and limitations. In: Russian Supercomputing Days. pp. 342–355. Springer (2022). https://doi.org/10.1007/978-3-031-22941-1_25

14. Ostroumova, G., Orekhov, N., Stegailov, V.: Reactive molecular-dynamics study of onion-like carbon nanoparticle formation. Diamond and Related Materials 94, 14–20 (2019). https://doi.org/10.1016/j.diamond.2019.01.019

15. Polyakov, S., Podryga, V., Puzyrkov, D.: High performance computing in multiscale problems of gas dynamics. Lobachevskii Journal of Mathematics 39(9), 1239–1250 (2018). https://doi.org/10.1134/S1995080218090160

16. Puente, V., Izu, C., Beivide, R., *et al.*: The adaptive bubble router. Journal of Parallel and Distributed Computing 61(9), 1180–1208 (2001). https://doi.org/10.1006/jpdc.2001.1746

17. Pugachev, L., Umarov, I., Popov, V., *et al.*: PIConGPU on Desmos supercomputer: GPU acceleration, scalability and storage bottleneck. In: Russian Supercomputing Days. pp. 290–302. Springer (2022). https://doi.org/10.1007/978-3-031-22941-1_21

18. Scott, S.L., *et al.*: The Cray T3E network: adaptive routing in a high performance 3D torus (1996)

19. Shamsutdinov, A., Khalilov, M., Ismagilov, T., *et al.*: Performance of supercomputers based on Angara interconnect and novel AMD CPUs/GPUs. In: International Conference on Mathematical Modeling and Supercomputer Technologies. pp. 401–416. Springer (2020). https://doi.org/10.1007/978-3-030-78759-2_33

20. Simonov, A.: Simulation model of high-speed Angara communication network with kd-tor topology. Trudy MAI (109), 22–22 (2019). https://doi.org/10.34759/trd-2019-109-22, (in Russian)

21. Simonov, A., Makagon, D., Zhabin, I., *et al.*: The first generation of Angara high-speed interconnect. Science Technologies 15(1), 21–28 (2014) (in Russian)

22. Stegailov, V., Dlinnova, E., Ismagilov, T., *et al.*: Angara interconnect makes GPU-based Desmos supercomputer an efficient tool for molecular dynamics calculations. The International Journal of High Performance Computing Applications 33(3) (2019). https://doi.org/10.1177/1094342019826667

23. Stegailov, V., Smirnov, G., Vecher, V.: VASP hits the memory wall: Processors efficiency comparison. Concurrency and Computation: Practice and Experience, p. e5136 (2019). https://doi.org/10.1002/cpe.5136

24. Tolstykh, M., Goyman, G., Fadeev, R., Shashkin, V.: Structure and algorithms of SLAV atmosphere model parallel program complex. Lobachevskii Journal of Mathematics 39(4), 587–595 (2018). https://doi.org/10.1134/S1995080218040145