DOI: 10.14529/jsfi150307

A Case for Energy-Efficient Acceleration of Graph Problems using Embedded FPGA-based SoCs

Pradeep Moorthy[1,2], *Nachiket Kapre*[1,3]

Sparse graph problems are notoriously hard to accelerate on conventional platforms due to irregular memory access patterns resulting in underutilization of memory bandwidth. These bottlenecks on traditional x86-based systems mean that sparse graph problems scale very poorly, both in terms of performance and power efficiency. A cluster of embedded SoCs (systems-on-chip) with closely-coupled FPGA accelerators can support distributed memory access with better matched low-power processing. We first conduct preliminary experiments across a range of COTS (commercial off-the-shelf) embedded SoCs to establish promise for energy-efficiency acceleration of sparse problems. We select the Xilinx Zynq SoC with FPGA accelerators to construct a prototype 32-node Beowulf cluster. We develop specialized MPI routines and memory DMA offload engines to support irregular communication efficiently. In this setup, we use the ARM processor as a data marshaller for local DMA traffic as well as remote MPI traffic while the FPGA may be used as a programmable accelerator. Across a set of benchmark graphs, we show that 32-node embedded SoC cluster can exceed the energy efficiency of an Intel E5-2407 by as much as 1.7× at a total graph processing capacity of 91–95 MTEPS for graphs as large as 32 million nodes and edges.

Keywords: energy efficiency, sparse graphs, embedded SoCs, FPGAs.

Introduction

During the pioneering years of HPC, computer architects built systems exclusively from specialized vector hardware; such as the Cray-I [1] and other bespoke machines like the NEC SX-3 and Fujitsu Numerical Wind Tunnel. The early 90s saw x86-based systems rise in popularity due to their low cost, simplicity and standardization of the ISA/floating-point system (Intel 8087 was an early example of IEEE-754 compliant processor hardware). Beowulf clusters of these x86 platforms began as low cost hobbyist alternative to state-of-art HPC systems. Based on the idea of connecting relatively inexpensive COTS computers to solve a particular problem collectively, the first such cluster was developed in 1994 by connecting 16 Intel DX4 processors with 10Mbps Ethernet. This eventually paved way for the creation of the first cluster based supercomputer in 1997, the ASCI Red, which employed 7,246 Intel x86 Pentium Pro processors linked using a custom-interconnect architecture. Peaking the TOP500 list for nearly three years, it set out the foundation for the dominance of x86 cluster systems we see today.

The same era saw the introduction of Reduced Instruction Set Architecture (RISC) based systems in place of Complex Instruction Set Architecture (CISC) machines in the form of PowerPC processors used in the IBM BlueGene. This supercomputer series was launched in 2004 to exploit the low power capabilities of RISC instead of CISC chips by combining multiple PowerPC processors onto each chip. Thus, the usage of multiple low power processors, typically RISC based, in place of a single power hungry "fat" processor was recognized as a way to improve energy efficiency. In lieu of PowerPC hardware, ARM chips have been gaining more interest in the research community since they are fabricated extensively in mobile devices to deliver low power at low cost. The largest ARM-based cluster studied was the Tibidabo cluster [2], which consisted

[1]Nanyang Technological University, Singapore
[2]University of Toronto
[3]nachiket@ieee.org

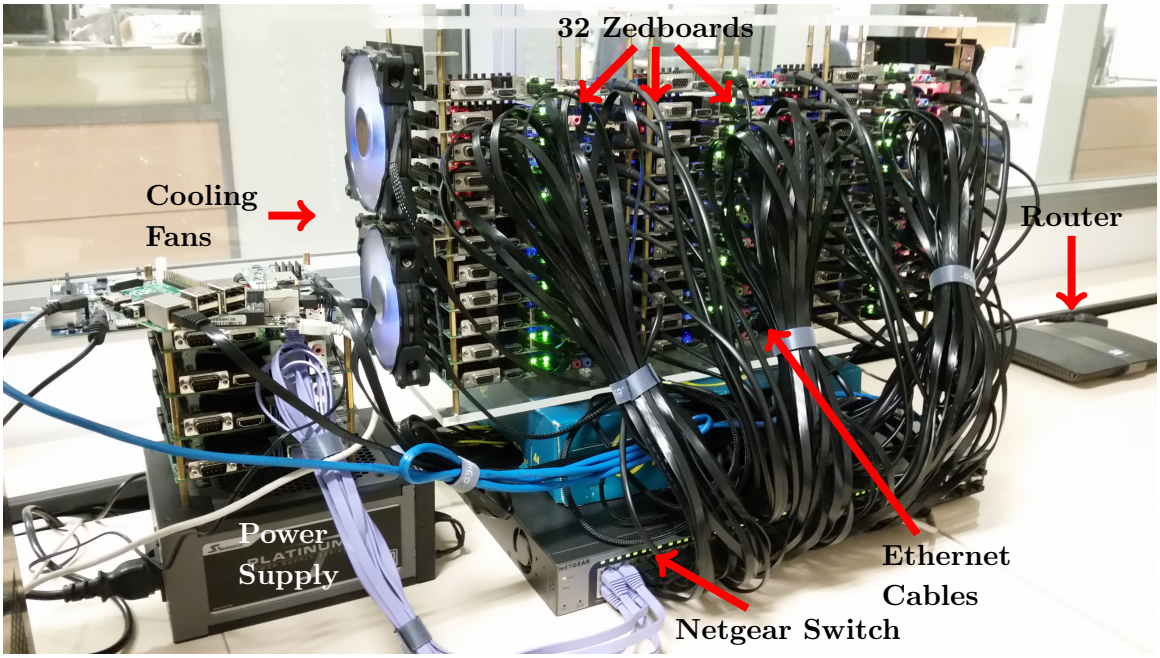

Figure 1. Zedwulf Cluster: 32 Zynq Z7020 SoC boards

of 192 NVIDIA Tegra-2 SoCs, interconnected using 1GbE network. The study concluded the lack of high-bandwidth I/O interfaces such as 10GbE/InfiniBand and the absence of hardware support for interconnection protocols on the Tegra-2's ARM Cortex-A9 processor as the sole limiting factors in adopting the SoC for HPC usage. While the present day performance gap between HPC-grade x86 processors and commercial ARM processors can be as high as an order of magnitude, large graph problems with low spatio-temporal locality can eliminate the performance gap between the two architectures while retaining the energy efficiency advantages. To investigate this claim, we prototype a Beowulf cluster composed of 32 Xilinx FPGA-based Zynq SoC boards, interconnected using a Gigabit Ethernet Switch. We map sparse-graph oriented irregular computations of varying dimensions to stress the memory and network throughputs of the cluster nodes. Fig. 1 shows a photograph of our "Zedwulf" (ZedBoard+Beowulf) cluster.

In this paper, we make the following key contributions:

- **Microbenchmarking of COTS SoCs**: We analyze the memory potential and network characteristics of various embedded SoCs using micro-benchmarking tools.
- **Prototype a 32-node Zynq SoC cluster**: We prototype physically a 32-node Zynq SoC cluster using the Xilinx Zedboard and Microzed platforms.
- **Communication optimization for sparse-graph access on the Zynq cluster**: We develop customized Message Passing Interface (MPI) routines and DMA engines optimized for irregular access exhibited by graph problems.
- **Performance and power evaluation of the Zynq cluster vs an x86 server node**: We benchmark our cluster for a few representative sparse graphs and compare against the Intel E5-2407 CPU.

1. Microbenchmarking COTS SoC Platforms

We first evaluate a range of COTS embedded SoC-based platforms listed in tab. 1 to assess their feasibility for scaling to larger-scale systems. Our characterization experiments focus on a

single chip and measure raw compute throughput, memory performance as well as MPI support for these systems.

Table 1. Comparing datasheet specifications and microbenchmarking of various COTS SoCs

	Zedboard	Microzed	Parallella	Intel Galileo 2	Raspberry Pi	Beaglebone Black
Technology	28nm	28nm	28nm	32nm	40nm	45nm
SoC	Xilinx Zynq 7020	Xilinx Zynq 7010	Xilinx Zynq 7010	Intel Quark X1000	Broadcom BCM2835	TI AM3359
Processor	ARMv7, FPGA	ARMv7, FPGA	ARMv7, FPGA, Epiphany III	i586	ARMv6	ARMv7
Clock Freq.	667 MHz CPU 250 MHz FPGA	667 MHz CPU 250 MHz FPGA	667 MHz CPU 250 MHz FPGA	400 MHz	700 MHz	1 GHz
On-chip Memory	32 KB L1 512 KB L2 560 KB FPGA	32 KB L1 512 KB L2 560 KB FPGA	32 KB L1 512 KB L2 240 KB FPGA	16 KB L1	16 KB L1 128 KB L2 -	32 KB L1 256 KB L2
Off-chip Memory	512 MB 32b DDR3-1066	1024 MB 32b DDR3-1066	1024 MB 32b DDR3-1066	256 MB 32b DDR3-800	512 MB 32b DDR2-400	512 MB 16b DDR3-606
DMIPS	1138	1138	1138	237	862	1778
Coremark	1591	1591	1782	526	1314	2457
Network[4]	57 MB/s	59 MB/s	32 MB/s	18 MB/s	10 MB/s	21 MB/s
L1 B/W	7.7 GB/s	7.7 GB/s	7.5 GB/s	2.8 GB/s	2.7 GB/s	7.6 GB/s
L2 B/W	1.4 GB/s	1.4 GB/s	1.4 GB/s	-	1.4 GB/s	3.4 GB/s
DRAM Seq.	654 MB/s	641 MB/s	537 MB/s	270 MB/s	187 MB/s	278 MB/s
DRAM Rnd.	32 MB/s	32 MB/s	28 MB/s	12 MB/s	10 MB/s	11 MB/s
Power	5 Watts	3.6 Watts	7.5 Watts	4 Watts	3.75 Watts	3.25 Watts

Recent academic studies have examined the feasibility of HPC systems based on mobile SoCs [3] for HPC-oriented workloads and investigated the status of networking support in these SoCs. Additionally, there are many contemporary hobbyist clusters built from **Apple TV** [4], **Raspberry Pi** [5], and **Beagleboard xM** [6] that use off-the-shelf devices for delivering proof-of-concept systems with high power efficiency. These studies are insightful but it remains to be seen if pure ARM-based SoCs have future prospects in the cluster computing space.

Our preliminary experiments on the **Intel Galileo 2** platform indicate the Quark SoC would not be competitive at this stage with its under-powered 400 MHz 32b CPU when compared to ARM-based embedded SoC platforms. It reported the lowest DMIPS score of 237 and had poor Ethernet throughput of 10 MB/s (100M Ethernet NIC). Occupying the lower-end of the ARM spectrum, the **Raspberry Pi** reported a 3x higher DMIPS/Coremark score than the Galileo 2. Nevertheless, its relatively slower DDR2 memory limits the overall performance gains. The **Beaglebone Black** further doubles the compute performance to 1778 DMIPS. However, the 16b 400 MHz DDR3 memory barely keeps up with its superior compute capabilities constraining overall performance. Besides, these devices are also limited by 100 Mb/s network links. The Zynq SoC-based platforms (Zedboard, Microzed and Parallella) overcome some of these shortcomings by coupling the Zynq SoC to a 1 Gb/s network link and a respectable 32b DDR3 1066 MHz memory. The **Zedboard** and **Microzed** delivered the highest sequential and random access memory bandwidths. Complemented by the Gigabit Ethernet connectivity, these platforms averaged bi-directional network throughput at a high 60 MB/s. Nonetheless, that corresponds only to a network efficiency of 24%. This behavior is attributed to the slower clock rate of the ARM cores (35% slower ARM CPU relative to the Beaglebone running at 1 GHz). In addition to the Zynq SoC, the **Adapteva Parallella** [7] platform also attaches an Epiphany floating-

[4] Intel MPI Benchmark Suite result for `MPI_Sendrecv` for all systems in 2-node configurations

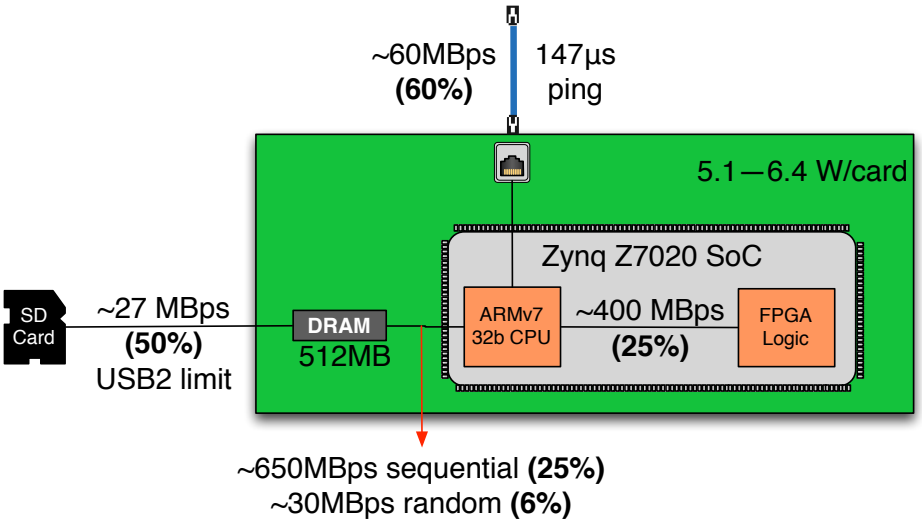

Figure 2. A Zynq node (Zedboard) with peak and achieved bandwidths

point co-processor as a separate chip thereby improving its compute capability substantially. We recorded comparable DMIPS and memory bandwidth scores on the Zedboard/Microzed, but the network throughput saturated at a disappointing 32 MB/s. The high local DRAM and remote MPI throughputs suggest that the Zedboard can become a viable candidate for energy-efficient operation for sparse irregular workloads. It is worth noting that these Zynq platforms are development systems with extraneous supporting logic for audio, video and configurable IOs that can be eliminated in a pure datacenter/HPC-focused design.

2. Zedwulf Organization

The Zedwulf cluster is composed of 32 Zedboards (Rev. D) or 32 Microzed (eval. kit), interconnected using a Netgear GS748T 48-port Gigabit Smart Switch. With a rated switching capacity of 96 Gb/s, the switch can sustain 2 Gb/s duplex bandwidth per 1GbE Ethernet link connecting each Zedboard. We powered the system using a Seasonic Platinum 1KW PSU from the PCIe EPS12 power rail with fuse protection. We stacked the Zedboards on top of each other in three columns with 10/11 boards on each column. We provided air cooling from 2 fans placed on either sides of the stack (4 fans total) as shown in fig. 1. While every Zedboard has a SanDisk Ultra 32 GB SD card attached to host the OS, the master node has an additional Samsung 840 Pro SSD attached to the USB2 port using a SATA-USB adapter. We setup the SSD as the primary secondary storage device for our cluster to hold our large graphs and it offers a convenient lower latency solution for quickly loading and distributing sub-graphs. A single Zynq node with various interface bandwidths is shown in fig. 2. We also built a 32-node Microzed cluster by simply replacing the Zedboard with Microzeds.

The Zynq is a heterogeneous multicore system architecture consisting of a dual-core ARM Cortex-A9 on the Processing System (PS) and a FPGA fabric on the Programmable Logic (PL). Residing on the same chip, the PS and PL are interconnected using AXI on-chip buses. This contrasts to traditional FPGA implementations, whereby the latter is connected to an x86 host using PCIe buses. This approach allows ARM processors to benefit from low-latency links to the FPGA which allow tightly-coupled CPU-based control of FPGA operation.

We configured each Zedboard to run Xillinux-1.3a, an Ubuntu-12.04 based Linux distribution with Xillybus drivers to communicate with the FPGA using an AXI 2.4 GB/s channel. We

compiled software libraries such as MPI and other utilities with `gcc-4.6.3` with appropriate optimization flags enabled. We use NFS (Network File System) to synchronize files (graphs) across all 32 nodes. We setup MPI to use Myrinet Open-MX patch to deliver a marginal improvement in network latency. We also choose MPICH over OpenMPI as it provided a 20–30% lower latency and higher bandwidth in our initial stress benchmarks.

3. Communication Optimization

Graph processing is a communication-dominated algorithm that can often be organized as lightweight computations on vertices and message-passing along edges. We map bulk-synchronous parallel (BSP) graph computations of the style used in neural network evaluation, page-rank calculations, and sparse matrix-vector multiplication. We map these evaluations to our cluster by careful optimization of local communication (irregular memory access) and remote communication (MPI access) and compare it against simple x86-based implementations that leverage multi-threading and compiler optimizations.

3.1. MPI Optimization

Partitioning the graph structure to fit across multiple *Processing Elements (PEs)* creates network traffic which connect local vertices to vertices present in other PEs. Unlike local edges, which connect vertices present within the same PE, updating remote edges is typically an order of magnitude slower as the data needs to be transferred from the origin PE to the target PE using the ARM CPUs to handle network packet transfers. Hence, there is an inherent need to reduce the time spent in fulfilling the network operations for maximizing performance gains while using distributed systems. We designed an optimized graph-oriented global scatter technique [8] using the Message Passing Interface (MPI) library.

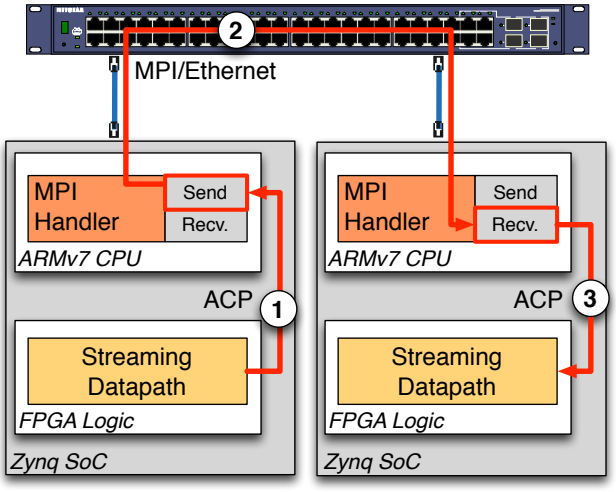

Figure 3. Sequence of steps for synaptic communication along edge of sparse graph. Step ① and Step ③ operate over the ACP links, while Step ② is managed by the MPI library over Ethernet

Our approach leveraged coalesced data transfers between PEs to take advantage of the network bandwidth, rather than being limited by the high network latencies. The high-level packet flow in the system is shown in fig. 3. We used **MPI_type_indexed** API to encode the *send*

and *receive* buffer displacements in an MPI friendly manner. We then employed **MPI_Sendrecv** as the building block of our scatter routine. The send-recv operations were scheduled in a periodic fashion to avoid network contention across MPI nodes. This coalesced approach of edge updates offered the speedup of **60**× when compared to performing fine-grained message transfers. We show a simplified code sketch of our MPI optimization in fig. 4.

```
// build MPI data structure from graph
for(j=0:total_proc-1) {
        send_type=MPI_Datatype();
        recv_type=MPI_Datatype();
        MPI_Type_indexed(send_count, send_addr,send_type);
        MPI_Type_indexed(recv_count, recv_addr,recv_type);
        MPI_Type_commit(send_t);
        MPI_Type_commit(recv_t);
}

// Loop over multiple bulk-synchronous steps
for(BSP steps) {
        // Manage local messages

        // Send MPI data between nodes
        for(j=0:total_proc-1) {
                // scheduling to avoid conflicts
                int target = (rank+j)%total_proc
                int source = (total_proc+rank-j)%total_proc;
                MPI_Sendrecv (send_buf, recv_buf, ...);
        }
        MPI_Barrier();

        // Do compute stuff
}
```

Figure 4. Basic MPI Communication Skeleton that shows how the `MPI_Datatype` is built and the mechanism of using `MPI_SendRecv` for sparse communication

First, we translate the sparse graph adjacency lists into MPI-compatible data types that encode the graph structure as a series of addresses and counts for send and receive between all-possible pairs of MPI nodes. This is done for those edges that cross compute node (SoC board) boundaries. We perform a coalesced transfer to one MPI target in a single function call to avoid MPI overheads of finer-grained messages. To achieve this coalesced transfer, we setup the `MPI_Datatype` using `MPI_Type_indexed` to encode a custom sequence of blocks with source and destination positions. We employ the Passive Target Communication paradigm here, using `MPI_Win_lock` and `MPI_Win_unlock` functions for executing Remote Memory Access (RMA) calls. We exploit opportunities for overlapping communication in the system by (1) having simultaneous epoch sessions in progress, ensuring load-balanced scheduling of message transfers in a cyclic fashion, and (2) replacing local `MPI_Put` with simple array-indirection.

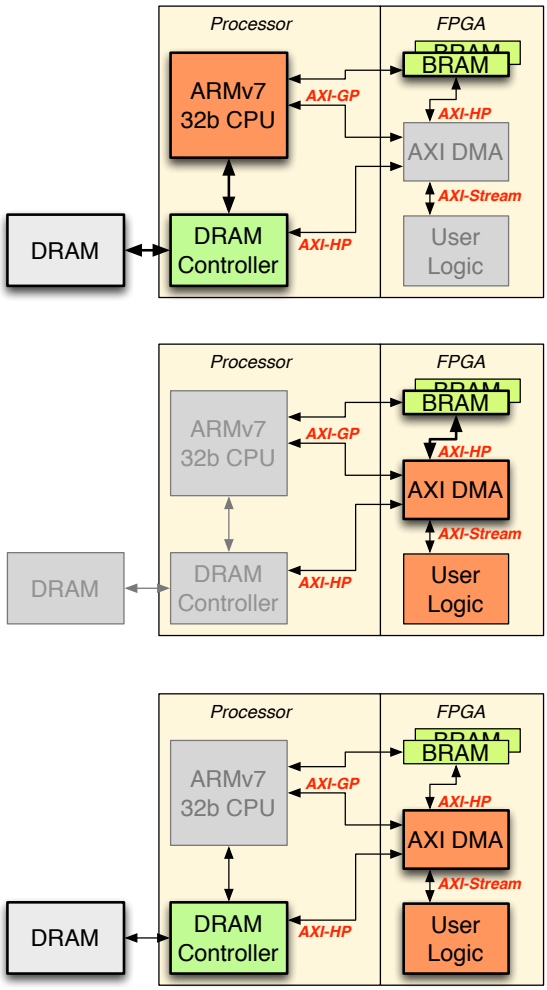

Figure 5. MMU Optimization: Scatter-Gather operation

3.2. Memory Access Optimization

For each vertex in the graph, the graph processor needs to fetch adjacent vertex data from local memory wherever possible. The graph data is conventionally stored in a compressed sparse format (row based or column based), which is a memory storage optimization for sparse graph structures. However, memory access patterns can still result in frequent cache misses under this memory organization scheme.

While the FPGA on the Zedboard has 560KB of on-chip memory, they can barely accommodate 100-1000s of graph vertex and edges. Using the off-chip DRAM memory carelessly would result in poor DRAM bandwidth utilization. Hence, we designed a Memory Management Unit (MMU) for Zedwulf to optimize irregular data transfers. We configure the AXI DMA IP block to use low-level AXI descriptor chains to encode the sparse graph access sequence. With our approach we are able to improve random DRAM access throughput for graph operations by as much as **3**–**4**×.

We operate the MMU in optimized Scatter-Gather mode [9]. This allows the `AXI_DMA` engine to avoid requiring frequent assistance from the CPU and enables somewhat independent operation. In this mode, instead of programming the internal registers for each DMA transfer, the CPU only needs to construct a one-off linked list of AXI descriptor commands for the complete

series of transfers. This can be done once at the start and reused repeatedly for iterative BSP-like graph algorithms. The descriptor chain is stored locally on the FPGA fabric in BlockRAMs and coupled to the AXI_DMA engines over an AXI-HP interface. It can even be constructed on-the-fly based on the compressed sparse-row representation of the graph, but we do not explore this at present.

```
XScuGic InterruptController;

struct axi_desc_t {
  u32 next;
  u32 base_addr;
  u32 control;
  u32 status;
};

struct axi_desc_t axi_desc[GRAPH_ACCESSES];

// Initialize graph access pattern as DMA descriptor chain
for (i=0; i<GRAPH_ACCESSES; i++)  {

    // create an entry in linked list
    axi_desc[i].base_addr = base_addr[i];
    axi_desc[i].control = length[i];
    Xil_Out32(BRAM_ADDR + i*ALIGN + NXTDESC, axi_desc[i].next);
    // copy other fields to BRAM
}

// Initialize DMA engine
Xil_Out32(DMAREG_ADDR + MM2S_CURDESC, BRAM_ADDR );
Xil_Out32(DMAREG_ADDR + MM2S_DMASR, 0x0000000);
Xil_Out32(DMAREG_ADDR + MM2S_DMACR, 0x5001);

// Perform DMA on the descriptor chain
Xil_Out32(DMAREG_ADDR + MM2S_TAILDESC, BRAM_ADDR + (GRAPH_ACCESSES-1)*ALIGN);
```

Figure 6. Scatter-Gather-Mode AXI_DMA device driver

In fig. 5 we show the three-step configuration flow for the Scatter-Gather DMA mode. In scatter-gather mode we represent the irregular list of accesses as a linked list of <base_addr>,<length> tuples stored in local on-chip FPGA BlockRAM. We instruct the DMA engine to get length bytes starting from base_addr location of the graph representation. Instead of forcing the interrupt after each transfer, we are able to perform a set of back-to-back transfers directly without interrupting the host until after the full sequence has been transferred. This ability to avoid frequent CPU interrupts coupled with FPGA-based storage of AXI descriptor chain provides low-latency turnaround times between consecutive DMA transactions. This is loaded once at the start over AXI-GP ports from the CPU. We represent this in fig. 6 in the InitializeDescriptors function. The address of the next descriptor is specified in each descriptor. The head and tail descriptors are provided to the DMA engine and it will process one descriptor after another.

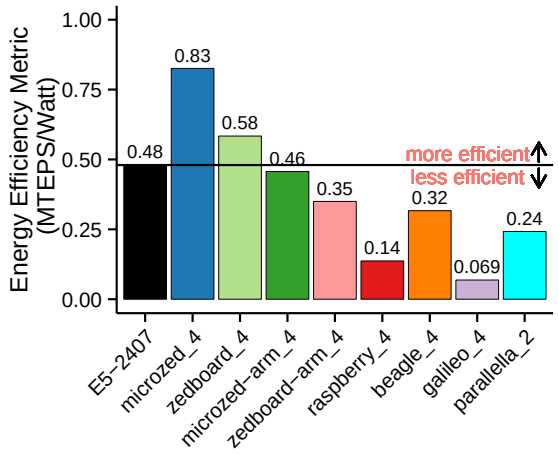

Figure 7. Performance-Power Tradeoffs across embedded SoCs platforms (4-node and 2-node SoCs) and a single x86 node ("-arm_4" versions exclude FPGA and only use ARM). Graph size is 32 million vertices and 32 million edges

4. Results

We analyze the performance and energy-efficiency of various embedded clusters for sparse graph processing. We perform bulk-synchronous evaluation on randomly-generated graphs (Erdos-Renyi [10] technique) with upto 32 million vertices and 32 million edges that can safely fit within the limited memory available on the embedded platform. For the first experiment, we setup 4-node clusters of each unique embedded platform (except Parallella with 2 nodes) and compare it against one x86 node. We scale our setup for the second experiment where we compare the 32-node Zedboard and 32-node Microzed clusters against a single x86 node. The code running on the single x86 node is parallelized using OpenMP pragmas to use all available cores (4 cores for the E5-2407). For completeness, our power measurements include the Ethernet switch and PSU along with the Zynq boards.

In fig. 7 we plot processing efficiency (MTEPS/W) across various embedded and x86 platforms. The Galileo 2 and Raspberry Pi clusters have the lowest performance while demanding high power usage. The Beagle cluster doubles the performance achieved while consuming 10% less power, thereby improving the power efficiency. The 2-node Parallella cluster nearly matches the performance of the 4-node Beaglebone but it needs more power for the extra Epiphany co-processor. The Zedboard and Microzed boards offer the highest energy efficiency when using the FPGA accelerators instead of simply relying on their ARM CPUs. The Microzed stands out with its 30% less power use over the Zedboard as it eschews unnecessary development support (audio, video, IO chips) in favor of a low-cost implementation.

In fig. 8 we show the performance (in MTEPS, millions of traversed edges per second) of the x86 node and the Zynq clusters plotted against their measured power consumption. We are able to marginally exceed the energy efficiency of the x86 node (0.48 MTEPS/W vs. 0.58 MTEPS/W) when using the Zedboard cluster. However, the lower-power and cheaper Microzed-based cluster is able to deliver a 1.7× improvement in energy efficiency (0.83 MTEPS/W) due to its lean design.

This measured 0.83 MTEPS/W energy efficiency figure is within striking distance of the 1.89 MTEPS/W[6] possible in the `SMALL DATA` category of the Green Graph500 list. We look

[6]http://green.graph500.org/lists.php, July 2015 list, University of Luxembourg

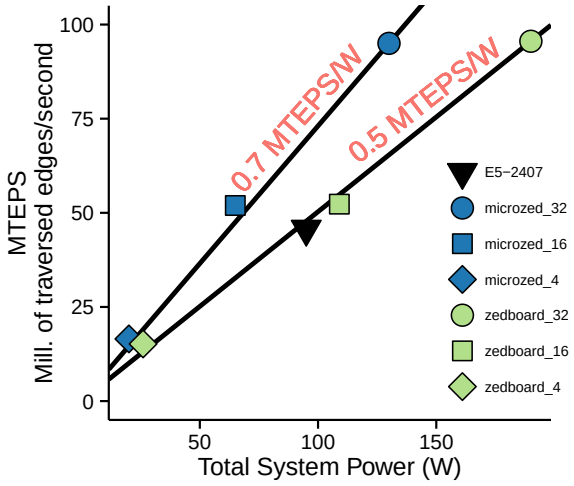

Figure 8. Energy Efficiency of Zynq FPGA cluster against an x86 node (4-node, 16-node and 32-node Zynq setups). Graph size is 32 million vertices and 32 million edges

forward to implementing the Graph500 benchmarks on larger problem sizes with larger cluster of Zynq nodes in the near future that builds upon this work.

5. Conclusions

We show how to use the Zynq SoC with ARMv7 32b CPUs supported by FPGA-accelerators to prototype energy-efficient HPC systems for sparse graph acceleration. For a range of graphs up to 32 million nodes and edges, we are able to deliver a performance of 91–95 MTEPS at an energy-efficiency of 0.58 MTEPS/Watt (32-node Zedboard), and 0.83 MTEPS/Watt (32-node Microzed) which exceeds the x86 efficiency of 0.48 MTEPS/Watt by as much as 1.7×. While the Zynq SoC we evaluated in this study is promising, performance gains were limited by (1) slow 1G Ethernet speeds of 50% peak, (2) limited DRAM capacity per node 512 MB, (3) poor CPU-FPGA link bandwidth of 400 MB/s, and (4) extraneous devices and interfaces for audio/video processing. Upgraded Zynq SoCs optimized for data-center processing that address these concerns can further improve performance and energy efficiency of these systems.

6. Future Work

While the Zynq SoC we evaluated in this study is promising, performance gains were limited by a variety of factors. The slow network transfers that saturate at only 50% of peak 1G Ethernet speed and MPI stack overheads result in a communication time that is roughly 2× worse than network performance of the x86. The limited DRAM capacity of 512 MB per node constrained the size of the largest graphs we could evaluate in this study. The poor CPU-FPGA link bandwidth of 400 MB/s meant that data-transfer time dominated FPGA runtimes. The Zedboard platform chosen in this study contains extraneous devices and interfaces that can be removed for HPC-like scenarios that reduces size, power and cost to be a better candidate for a future study. It may even be prudent to evaluate the smaller Z7010 SoC (cost $56/chip compared to $100/chip for the Z7020) with a smaller FPGA fabric for better balanced design. To improve the programmability of the FPGA design, the use of arrays of soft-processor tiles [11] overlayed on top of the FPGA but fully-customized to particular graph problems would be a promising approach. The Parallella

platform with specialized high-performance I/O banks could be used as a superior interconnect alternative to Ethernet for sparse low-latency communication between SoC chips.

References

1. Richard M. Russell. The CRAY-1 Computer System. *Commun. ACM*, 21(1):63–72, January 1978. DOI: 10.1145/359327.359336.
2. Nikola Rajovic, Alejandro Rico, Nikola Puzovic, Chris Adeniyi-Jones, and Alex Ramirez. Tibidabo: Making the case for an ARM-based HPC system. *Future Generation Computer Systems*, 2013. DOI: 10.1016/j.future.2013.07.013.
3. Nikola Rajovic, Paul M Carpenter, Isaac Gelado, Nikola Puzovic, Alex Ramirez, and Mateo Valero. Supercomputing with commodity CPUs. In *the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, New York, New York, USA, 2013. ACM Press. DOI: 10.1145/2503210.2503281.
4. Karl Fürlinger, Christof Klausecker, and Dieter Kranzlmüller. The AppleTV-cluster: Towards energy efficient parallel computing on consumer electronic devices. *Whitepaper, Ludwig-Maximilians-Universitat*, 2011. DOI: 10.1007/978-3-642-23447-7_1.
5. Simon J Cox, James T Cox, Richard P Boardman, Steven J Johnston, Mark Scott, and Neil S O'Brien. Iridis-pi: a low-cost, compact demonstration cluster. *Cluster Computing*, 17(2):349–358, June 2013. DOI: 10.1007/s10586-013-0282-7.
6. E Principi, V Colagiacomo, S Squartini, and F Piazza. Low power high-performance computingon the Beagleboard platform. In *Education and Research Conference (EDERC), 2012 5th European DSP*, pages 35–39, 2012. DOI: 10.1109/ederc.2012.6532220.
7. Linley Gwennap. Adapteva: More Flops, Less Watts. *Microprocessor Report*, pages 1–5, June 2011.
8. P. Moorthy and N. Kapre. Zedwulf: Power-performance tradeoffs of a 32-node zynq soc cluster. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 68–75, May 2015. DOI: 10.1109/fccm.2015.37.
9. N. Kapre, Han Jianglei, A. Bean, P. Moorthy, and Siddhartha. Graphmmu: Memory management unit for sparse graph accelerators. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 113–120, May 2015.
10. Paul Erdös and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960.
11. N. Kapre. Custom fpga-based soft-processors for sparse graph acceleration. In *Application-specific Systems, Architectures and Processors (ASAP), 2015 IEEE 26th International Conference on*, pages 9–16, July 2015. DOI: 10.1109/asap.2015.7245698.

Received June 3, 2015.