Understanding the Role of GPGPU-accelerated SoC-based ARM Clusters

Reza Azimi, Tyler Fox and Sherief Reda
Brown University
Providence, RI
firstname_lastname@brown.edu

***Abstract*—The last few years saw the emergence of 64-bit ARM SoCs targeted for mobile systems and servers. Mobile-class SoCs rely on the heterogeneous integration of a mix of CPU cores, GPGPU cores, and accelerators, whereas server-class SoCs instead rely on integrating a larger number of CPU cores with no GPGPU support and a number of network accelerators. Previous works, such as the Mont-Blanc project, built their prototype ARM cluster out of mobile-class SoCs and compared their work against x86 solutions. These works mainly focused on the CPU performance. In this paper, we propose a novel ARM-based cluster organization that exploits faster network connectivity and GPGPU acceleration to improve the performance and energy efficiency of the cluster. Our custom cluster, based on Nvidia Jetson TX1 boards, is equipped with 10Gb network interface cards and enables us to study the characteristics, scalability challenges, and programming models of GPGPU-accelerated workloads. We also develop an extension to the Roofline model to establish a visually intuitive performance model for the proposed cluster organization. We compare the GPGPU performance of our cluster with discrete GPGPUs. We demonstrate that our cluster improves both the performance and energy efficiency of workloads that scale well and can leverage the better CPU-GPGPU balance of our cluster. We contrast the CPU performance of our cluster with ARM-based servers that use many CPU cores. Our results show the poor performance of the branch predictor and L2 cache are the bottleneck of server-class ARM SoCs. Furthermore, we elucidate the impact of using 10Gb connectivity with mobile systems instead of traditional, 1Gb connectivity.**

***Index Terms*—GPGPU acceleration, energy efficiency, ARM computing, distributed systems.**

I. Introduction

ARM 64-bit processing has generated enthusiasm to develop ARM-based servers that are targeted for both high-performance computing (HPC) clusters and data centers. Examining existing and upcoming server-based, ARM System-on-a-Chip (SoC) designs reveals that upcoming SoCs are trending toward including larger numbers of CPU cores. For instance, Applied Micro's X-Gene 1 contains 8 ARM cores, the planned X-Gene 3 will have 32 cores, and Cavium's ThunderX SoC packs 48 ARMv8 cores per socket [2]. In addition to CPU cores, these SoCs include IP blocks for a memory controller and 10Gb network and I/O connectivity.

The makeup of server-class SoCs is different from mobile-class ARM SoCs. Mobile-class ARM SoCs emphasize heterogeneous integration that uses fewer CPU cores at the expense of Graphical Processing Unit (GPU) cores. While these GPU cores have historically been dedicated solely to graphics, new mobile-class ARM SoCs incorporate general-purpose GPU (GPGPU) cores that can be programmed for scientific applications.

In this paper, we propose a novel cluster organization using mobile-class ARM SoCs that is different than previous approaches in two unique ways: (1) its SoCs use general-purpose GPUs in addition to traditional CPU cores at the cluster level to increase performance and energy efficiency, and (2) it uses 10Gb controllers for communication between nodes in the cluster instead of traditional 1Gb network connectivity. We show that modern, mobile-class ARM SoCs are capable of utilizing the additional bandwidth afforded by 10Gb connections. We believe the availability of general-purpose GPUs and faster network connectivity opens a new direction for HPC ARM clusters. To analyze the proposed cluster organization, we build our own custom cluster using mobile SoC boards that we outfit with 10Gb network controllers. We then use our cluster to understand the impact of node architecture and cluster organization on the performance and energy efficiency of HPC applications, including classical scientific applications and emerging Artificial Intelligence (AI) applications for deep learning. The contributions of this paper are as follows.

- We propose a novel cluster organization that exploits GPGPU acceleration and faster network connectivity to improve the energy efficiency and performance of the ARM-based cluster compared to previous designs.
- Previous works, such as the Mont-blanc prototype, only looked at the CPU performance of the cluster, despite the fact that their cluster was equipped with GPGPUs [19]. This is mainly due to the complexity and lack of workloads that can be used for GPGPU-accelerated ARM clusters. We identified and ported a collection of benchmarks (ClusterSoCBench) to the ARM environment that can be used to evaluate the CPU+GPGPU performance of the whole cluster.
- We extend the Roofline model for the proposed cluster organization to establish a visually intuitive performance model that can be used to bound its performance [26].
- We quantify the main factors that affect the scalability of our GPGPU-accelerated workloads and compare them to the traditional CPU workloads. We also study different CUDA memory management models and show the benefits and disadvantages of each.
- Compared to server-class ARM SoCs, we demonstrate that a large number of ARM cores on a single chip does not necessarily guarantee better performance due to the poor performance of the branch predictor and L2 cache for the server-class ARM SoCs.
- We compare the GPGPU performance of our cluster

against discrete GPGPUs. We observe that our cluster provides better CPU/GPGPU balance, which improves the performance and energy efficiency of image classification using deep neural networks, since the JPEG images are decoded on the CPU and fed to the GPGPU for processing using the neural network.

The organization of this paper is as follows. In Section II, we review related work and background on CUDA programming models. In Section III, we describe and analyze our proposed cluster organization. In Section IV, we compare our cluster against other systems. Finally, we summarize the main conclusions in Section V.

II. Related work

A. ARM-Based Clusters

A number of studies recently appeared that focus on the use of low-power, ARM-based SoC platforms in the HPC domain [19], [18], [21], [20], [16], [4], [13], [5]. Because an ARM-based mobile core exhibits lower performance than a traditional server-class core, an ARM-based cluster requires a larger number of cores to deliver the same performance as fewer, more powerful x86 cores. As such, strong scalability becomes an important requirement. Rajovic *et al.* designed Tibidabo, an HPC cluster with 128 nodes, where each node is based on a mobile, 32-bit Nvidia Tegra2 SoC, featuring dual Cortex-A9 cores [18], [21], [20]. The study points to a number of limitations, such as low network bandwidth and the lack of error protection, that arise from using mobile platforms. For the *hpl* benchmark, which exhibits good weak scaling, Tibidabo achieves an efficiency of 120 MFLOPS/W with an average of $\frac{120}{128\times 2} = 0.47$ MFLOPS/W per core. In a similar direction, Padoin *et al.* compare the energy efficiency of a number of ARM-based platforms and demonstrate an efficiency of up to 206.6 MFLOPS/W [16]. Mont-Blanc is the latest prototype to use mobile-class ARM SoCs [19]. Mont-Blanc is based on Cortex A15 (ARMV7) and uses 1GbE for network communication. Unlike with Tibidabo, whose GPUs were not programmable, the integrated GPGPUs used in the Mont-Blanc cluster are programmable using OpenCL. However, the Mont-Blanc study only evaluates the CPU performance of the cluster. For 64-bit ARM-based platforms, a recent study compares the performance and power of the X-Gene SoC against the standard Intel Xeon and the recent Intel Phi [4]. This study concludes that these systems present different trade-offs that do not dominate each other, and that the X-Gene 1 provides one of the higher energy efficiencies, measured in performance/watt. Azimi *et al.* evaluated the performance and energy efficiency of an ARMv8 X-Gene 1 SoC and an x86 Atom for a number of scale-out and high-performance computing benchmarks [5]. They analyzed the impact of the SoC architecture, memory hierarchy, and system design on the performance and energy efficiency outcomes.

B. CUDA Programming Model Background

Traditionally, GPGPUs act as coprocessors that operate on a set of data that is allocated, copied, and then later freed by the CPU. The GPGPU programming model hides the latency of the data transfers by concurrently transferring one stream of data and executing another stream whose data is ready. In terms of hardware, integrating the GPGPU cores on the same die gives mobile-class SoCs a truly unified memory architecture system that can further reduce the latency of data transfers by eliminating the slow movement of data between the CPU and GPGPU memory over the PCIe bus, which is necessary for discrete GPGPUs.

As for the programming model, CUDA offers three types of memory management between the CPU (host) and the GPGPU (device) [7]. Below is an overview of each type of CUDA memory transfer model:

Host and device memory: This is the conventional method in which the host and device have different main memory address spaces. The host and device can only access data that explicitly exists in their respective address space. Programmers need to copy data from the host's address space to the device's address space and vice versa using explicit `cudaMemcpy` calls. Even in systems with a unified memory architecture, such as Nvidia's TX1, where the host and device share the same main memory, the address spaces for the host and device are still separate when this memory management model is used and copying is needed.

Zero-copy: Zero-copy enables device threads to directly access host memory. The design goal of zero-copy was to avoid superfluous memory copies in systems with a unified memory architecture, such as the TX1, where the host and device memory are physically the same.

Unified memory: Unified memory creates a pool of managed memory shared between the host and device and automatically migrates the data between the host and device memory addresses to exploit the locality. Unified memory is designed to make CUDA programming easier, increase the total memory accessible for both the host and device, and offer the benefit of storing data in local memory by automatically migrating the data. Unlike zero-copy, unified memory is designed for both unified memory and discrete GPGPUs.

III. Proposed architecture and analysis

Using ARM for HPC is based on the philosophy of obtaining improved energy efficiency by using nodes that deliver less performance individually in exchange for much lower power consumption. The lower per-node power consumption allows a higher number of nodes to be used given the same power budget as a traditional server. We propose a novel cluster organization for mobile-class ARM SoCs that is different from previous work in two major ways:

1) We advocate for the use of faster 10GbE network instead of the available standard 1GbE on mobile boards.
2) We advocate for the use of general-purpose graphical processing units that are available on mobile-class ARM SoCs to increase the performance and energy efficiency of the whole cluster.

Previous attempts to build high-performance computing clusters using mobile-class SoCs have utilized 1GbE network connectivity for communication between nodes for two main reasons. First, early mobile boards did not offer expansion

tag	description	input size
hpl	High performance Linpack solving Ax=b [15]	N=10000
cloverleaf	Solves compressible Euler equations [24]	4080 cells and 100 steps
tealeaf2d	Solves the linear heat conduction equation in 2D [25]	4000 x and y cells and 10 steps
tealeaf3d	Solves the linear heat conduction equation in 3D [25]	250 x, y and z cells and 5 steps
jacobi	Solves Poisson equation on a rectangle [17]	16800×16800 matrix size
alexnet	Parallelized Caffe to classify ImageNet images using the AlexNet model [10], [11]	1000 images
googlenet	Parallelized Caffe to classify ImageNet images using the GoogleNet model [10], [23]	1000 images

TABLE I
SUMMARY OF THE GPGPU ACCELERATED WORKLOADS.

PCIe slots, which meant that it was not possible to add an additional network adapter to upgrade the network connectivity. Second, the CPU cores of older boards were not capable of driving enough network traffic to take advantage of the additional network bandwidth provided by 10GbE network adapters. Unlike previous efforts, our nodes, based on the Nvidia Jetson TX1, feature ×4 PCIe slots that we used to install 10GbE network controllers to make the cluster much more competitive with modern clusters.

Furthermore, previous cluster organizations using mobile-class SoCs only focused on studying the CPU cores available on the SoCs for two main reasons. First, the GPUs available on the SoC of these clusters were not designed for general-purpose computing. The Tibidabo cluster is an example of such efforts [20]. Second, the programming model becomes extremely complicated when a cluster of GPGPUs are used to solve a problem. While CUDA provides an easier framework to program GPGPUs, the GPGPUs available in works such as Mont-Blanc were not CUDA programmable [19]. Thus, Mont-Blanc only focused on evaluating the performance of a single GPGPU node instead of the whole cluster. Mont-Blanc only accounted for the GPGPU performance of the entire cluster using back-of-the-envelope calculations; as a result, it did not consider many details that affect performance. Using representative workloads, we take an in-depth look at the performance of the GPGPU-accelerated cluster as a whole and quantify the improvements in energy efficiency brought about by using GPGPU acceleration across the cluster. Our results also show that for suitable workloads, mobile GPGPU accelerators available on our cluster can demonstrate even better performance than discrete GPGPUs of the same architecture family for a given power budget.

In the rest of this section, we look at the different aspects of the proposed cluster organization. We first provide the details of the proposed cluster organization and our software stack. We then analyze the effect of the network and GPGPU-accelerated computing in detail.

A. Cluster Organization

Hardware organization: We use 16 Jetson TX1 boards to build our cluster. Each TX1 SoC has 4 Cortex A57 CPU cores running at 1.73 GHz[1] and 2 Maxwell streaming multiprocessors (SM), for a total of 256 CUDA cores running at 0.9 GHz. Each Jetson board has 4 GB of LPDDR4-1600 main memory that is shared between the CPU and GPGPU cores. Using the `stream` benchmark, we measured the maximum memory bandwidth to the CPU and GPGPU cores, which we found to be 11.72 GB/s and 21 GB/s, respectively [12], [8]. Each Jetson board also has 16 GB of eMMC storage on which the kernel and OS are installed. We used an NFS mounted file server on all 16 TX1 nodes for extra storage to collect all logs and traces; however, the binaries were all available locally. The file server uses SSDs for storage. Throughout the paper, when discussing data transfer between the CPU and GPU, the CPU is referred to as the *host* and the GPGPU is referred to as the *device*. Regarding the annotations in the figures and tables, unless otherwise noted, we refer to each Jetson TX1 board as a *node*. For example, 8 nodes means 8 TX1 Jetson boards are used to obtain results.

10GbE network tuning: mobile-class development boards, such as the Jetson TX1, come standard with a 1Gb Ethernet Network Interface Controller (NIC). While 1GbE is more than enough for typical mobile use, it is rarely sufficient for the demands of cluster-based computing. To make our cluster's network competitive with traditional clusters, we connect a Startech PEX10000SFP PCIe 10GbE network card to the PCIe x4 slot on each of the Jetson TX1 boards. Compared to the on-board 1GbE controller, the addition of the 10GbE card improves the average throughput between two TX1 nodes from 0.53 Gb/s to 3.13 Gb/s, measured using the `iperf` tool, and the average latency of the ping-pong latency test from 0.4 ms to 0.05 ms, measured using Latency-Bandwidth benchmark [12]. The storage server and all nodes are connected using the 10GbE NIC with a Cisco 350XG managed switch that has a bisection bandwidth of 120 Gb/s. For experiments with 1GbE, a 48-port Netgear switch is used.

Software stack: We consider a large range of benchmarks that are representative of HPC/AI applications. The lack of a standard benchmark suite is one of the barriers to evaluating the performance of GPGPU-accelerated clusters. We identified several CUDA benchmarks that are parallelized with the Message Passing Interface (MPI) model to analyze the proposed cluster organization. We ported them to work with the ARM environment. Table I lists the ported workloads and the input sizes we used. Input sizes were chosen to fully utilize the main memory of a TX1 node. All workloads were compiled with `-O3` optimization. For AI applications, we consider Caffe [10], which is representative of emerging deep learning frameworks. We developed our own scripts to distribute jobs that process and classify images in parallel across all nodes using the AlexNet [11] and GoogleNet [23] deep neural network models. All of our GPGPU-accelerated workloads (ClusterSoCBench) can be found on Github to facilitate research in the areas that demand the benchmarks [1].

In addition, we use the NAS Parallel Benchmarks (NPB) to evaluate our cluster's CPU performance with a workload size of class C. Class size C is chosen since it is the largest class

[1]TX1 documentation states the CPU frequency is 1.9 GHz, however our boards run at a maximum of 1.73 GHz.

size that can fit into the memory of a single TX1 node (except `ft`). NPB is compiled using the `-O3` flag on all systems. For the experiment with the CPU version of the `hpl` benchmark, we used the one available in HPCC suite [12].

For all of our experiments, we run the benchmarks as they come, without modifying or hand-tuning any of the code/libraries for specific systems. This decision is made to make the comparison of systems from different vendors fair and to account for the fact that architecture-dependent optimizations are in different stages for different vendors' systems. Further performance and energy efficiency improvements might be attained when more aggressive architecture optimization is considered.

OS and compilers: We installed Ubuntu 14.04 with kernel 3.10.96 on all of our TX1 nodes using the Jetpack 24.1 installer. Jetpack 24.1 installs CUDA 7.0 along with the Ubuntu operating system. GCC is upgraded to 5.4 on all nodes. OpenMPI 1.10 and OpenBLAS 0.2.19 stable versions are compiled from source and are used for MPI and BLAS libraries. To build Caffe, Python 2.7, Boost 1.54, Google protobuf 2.5, CUDNN 5, and HDF5 1.8 are used, and all Python dependencies are installed using `pip`.

Power and performance measurement: We fully instrumented all servers to record a wide array of measurements, including performance-monitoring counters, network traffic statistics, and power consumption. Performance-monitoring counters are collected using Linux `perf`, GPGPU events and metrics are collected using `nvprof`, network traffic statistics are collected using `nmon`, and workload traces are collected using Extrae [6]. Performance-monitoring counters both for the CPU and GPGPU are collected on different runs from the power/performance measurements. For CPU performance counters, we collected the same number of counters as actual available Performance Monitoring Unit (PMU) registers on each run. All performance counters were collected over many runs to avoid multiplexing. The CPU affinity of all processes is set to fixed logical cores in all runs.

The power consumption of each system is measured by sensing the external current at the 120 V AC socket with a sampling rate of 10 Hz. We consider two metrics for energy efficiency: (1) the total energy consumption, and (2) the floating-point operations per second (FLOPS) per watt.

B. Analysis of the Proposed Cluster Organization

In the rest of this section, we look at the different aspects of the proposed cluster organization: 1) we study the effect of network choice on our cluster; 2) we examine the communication characteristics of the GPGPU-accelerated workloads; 3) we extend the Roofline model for the proposed cluster organization; 4) we study the scalability of our workloads; 5) we analyze different memory management models offered by CUDA for our cluster; and 6) we quantify the energy efficiency of simultaneously using CPU and GPGPU cores.

1) The network choice: Figures 1 and 2 show the speedup and energy consumption when the 10GbE cards are used compared to when the standard 1GbE is used. Adding these cards on the PCIe slots adds to the power consumption of the cluster (about 5 W per node); however, a massive improvement

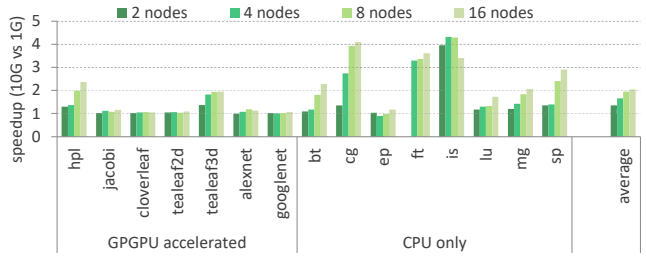

Fig. 1. Speedup gained by using the 10GbE NIC compared to 1GbE for different cluster sizes.

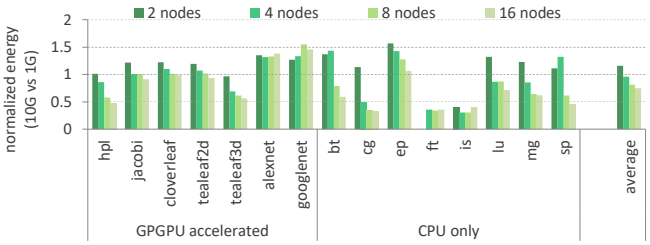

Fig. 2. Normalized energy consumption when the 10GbE NIC is used compared to using 1GbE for different cluster sizes.

in runtime is attained for network-intensive workloads, which increases the overall energy efficiency of the cluster. On average, for the 16-node cluster, we achieve a 2× speedup and 25% improvement in energy efficiency when the 10GbE is used. Figures 1 and 2 show the performance and energy efficiency advantages of using faster network connectivity increases as the cluster size grows. This is because inter-node communication increases with cluster size, causing the network to have a greater impact. Our results validate the benefits of using 10GbE network controllers instead of standard 1GbE connectivity in modern clusters made from mobile-class SoCs.

2) GPGPU communication characteristics: In highly parallel systems, such as GPGPUs, data transfer becomes a major performance bottleneck in a similar way to last level cache (LLC) stalls for multi-processor CPUs. At the cluster level, part of the data must be transferred from other nodes. As GPUDirect technology is not supported on TX1 boards, communication must be handled by the CPU and then transferred to the GPU through main memory. Transferring data through network has different overhead than transferring data from the main memory to the GPGPU; therefore, these two types of data transfers must be accounted for separately. To analyze the performance of the cluster, we looked at the DRAM traffic to the GPGPU and the network traffic between nodes. The DRAM traffic includes both the network data transferred from other nodes and the data that exists locally in the main memory.

Figure 3 shows the average DRAM and network traffic on a logarithmic scale for our workloads running on 8 TX1 nodes for 10GbE and 1GbE. We labeled the points with the name of the workload followed by the NIC used in the experiment. Benchmarks that cause high network traffic also cause high DRAM traffic because they have larger overall data footprints. However, different network and DRAM traffic ratios result in different communication patterns in our workloads. We observe three different types of communication patterns in the benchmarks.

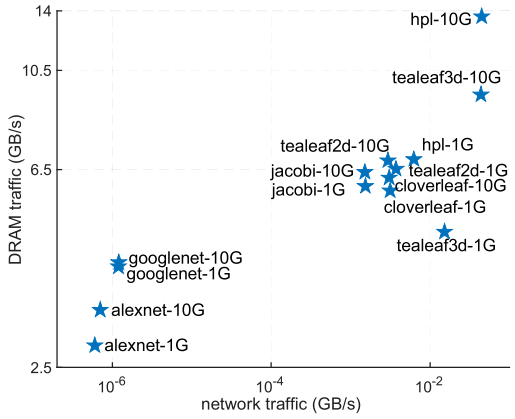

Fig. 3. Average DRAM and network traffic for different workloads and network speeds. The points are labeled with the name of the benchmark followed by the network speed.

- For *Tealeaf3d* and *hpl*, DRAM traffic increases by 93% and 99% respectively when 10GbE is used instead of 1GbE. The lower DRAM traffic of 1GbE experiments relative to the 10GbE experiments shows that *hpl* and *tealeaf3d* have large data footprints, meaning that a slow network limits data transfers to the GPGPUs and starves them of data to process. This aligns with our observation in Figure 1, in which *tealeaf3d* and *hpl* show the largest speedup when the 10GbE NIC is used, compared to the 1GbE.
- *Tealeaf2d*, *cloverleaf*, and *jacobi* show moderate network and DRAM traffic usage; therefore, no appreciable speedup is gained by using 10GbE connectivity.
- The third type of communication pattern is when workloads have a relevantly large DRAM to network traffic ratio. This is the case when most of the GPGPU data is available locally on the node, and therefore little communication over the network is needed. *Alexnet* and *googlenet* are examples of such workloads that have different communication characteristics than the scientific workloads. This emerging class of workloads benefit the most from using clusters of mobile-class GPGPUs.

It is crucial to understand that network transfers alone do not characterize the performance of the cluster. Performance of the cluster is a function of how well the GPGPUs' execution engines are fed data. For example, there is a case that our benchmarks do not cover in Figure 3: both small network and DRAM traffic, which leads to GPGPU starvation. It is the programmer's duty to avoid this case. For the presented workloads, the total DRAM traffic and network traffic are correlated and both contribute to performance. Thus, their ratio must be considered when analyzing the workload performance.

3) Roofline Model extension: The well-known Roofline model describes the peak computational capacity of a single chip in a visually intuitive way [26]. The Roofline model is based on three components: 1) communication, 2) computation, and 3) locality. In the standard model, the term communication describes the transfer of data from the DRAM to the caches, computation is defined as the number of floating-point operations, and locality is associated with operational intensity and is defined as the ratio of floating-point operations to the total bytes transferred from the DRAM.

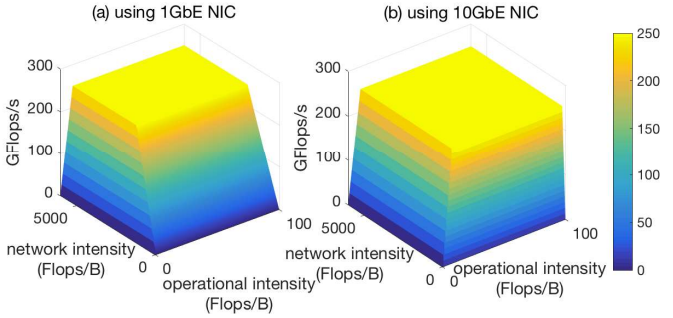

Fig. 4. Proposed Roofline model extension for different network speeds: a) using 1GbE NIC b) using 10GbE.

As mentioned in the previous subsection, there are two major types of data transfers in an integrated GPGPU-accelerated cluster: 1) the network data that is transferred between nodes and 2) the data that is transferred between the host and device memory on a single node. These data transfers happen with different speeds and must be measured separately. The similarity of data characteristics we observed for the GPGPU-accelerated workloads running on our cluster to the Roofline model motivates us to extend the Roofline model. In our extension to the Roofline model for integrated GPGPU-accelerated clusters, we define *communication* as the data transferred over the network between nodes, *computation* as the floating-point operations performed by the GPGPU, and *locality* as the data transferred through the DRAM to the GPGPU. To use this model correctly, we define the *operational intensity* as the ratio of total floating-point operations to the number of bytes transferred from the main memory to the GPGPU. We also define *network intensity* as the ratio of total floating-point operations to the number of bytes transferred over the network. Equations (1) and (2) define the *operational intensity* and *network intensity*, respectively. Equation (3) defines the peak performance calculated by the proposed Roofline model extension.

$$\textit{operational intensity} = \frac{\text{FLOPS throughput}}{\text{DRAM traffic}} \tag{1}$$

$$\textit{network intensity} = \frac{\text{FLOPS throughput}}{\text{NIC traffic}} \tag{2}$$

$$\text{peak performance} = \text{MIN}\Big(\text{peak computational capacity},\ \text{peak memory bandwidth} \times \textit{operational intensity},\ \text{peak network bandwidth} \times \textit{network intensity}\Big) \tag{3}$$

The Roofline model assumes complete overlap between both communication and computation, and therefore describes the theoretical peak performance for a system. However, it is still useful to have a visually intuitive performance model to compare workloads and determine how to improve performance. Figure 4 shows the theoretical peak performance when 1GbE and 10GbE NICs are used. The integrated GPGPU on each TX1 node has a peak theoretical computational capacity of 250 double-precision GFLOPs. Figure 4 shows how the attainable peak performance is limited by the choice of network.

Table II shows the measured performance, operational and network intensity, percentile of theoretical peak performance,

benchmark	operational intensity (FLOP/B)	network intensity (FLOP/B)	1GbE			10GbE		
			throughput (GFLOPS)	percentile of peak(%)	limit	throughput (GFLOPS)	percentile of peak(%)	limit
hpl	0.56	169.7	3.82	27.33	network	7.61	54.45	operational
jacobi	0.30	1275.77	1.80	23.99	operational	1.93	25.67	operational
cloverleaf	0.01	20.32	0.06	23.48	operational	0.06	25.01	operational
tealeaf2d	0.03	74.01	0.21	26.10	operational	0.22	27.18	operational
tealeaf3d	0.09	19.45	0.44	25.36	network	0.85	37.34	operational
alexnet	0.47	2155245.79	1.32	11.10	operational	1.57	13.22	operational
googlenet	0.82	2794432.12	3.34	16.28	operational	3.41	16.61	operational

TABLE II
EXTENDED ROOFLINE MODEL AND MEASURED PARAMETERS FOR DIFFERENT NETWORK SPEEDS USING 8 NODES.

and limiting intensity for each benchmark and network option. Table II quantifies how much the choice of network affects the performance of each node in the cluster, specifically for the workloads that exhibit high network traffic. The limiting intensity specifies which intensity, operational or network, limits the theoretical peak performance the most, given the peak memory and network bandwidth. Compared to other benchmarks, the large operational and network intensities of *jacobi*, *alexnet* and *googlenet* show that these benchmarks are more compute bound. Of the selected benchmarks, *hpl* comes closest to reaching the peak performance value due to its large operational and network intensities and high throughput, relative to the other scientific workloads. *Hpl* is the most commonly used benchmark to measure the performance of HPC systems and is highly optimized. Using a faster network does not change the operational intensity or network intensity, as these parameters are workload-dependent; that is, the total FLOPs, memory requests and data transferred over the network remain the same.

4) Scalability: Studying the scalability of workloads is essential to high-performance computing clusters with large number of nodes. We used the methodology proposed by Roses *et al.* to study the scalability of our workloads [22]. Traces of the workloads running on our cluster were collected using the Extrae tool for different cluster sizes [6]. All workloads except *hpl* use iterative approaches. Traces for these benchmarks are chopped using the PARAVER trace visualization tool [6]. For *hpl*, we use the whole trace as one big phase. Parallel efficiency for strong scaling can then be defined as shown in Equation (4) [22].

$$\eta = \frac{\text{speedup}}{P} = LB \times Ser \times Trf \tag{4}$$

where P is the number of processing units, LB describes how well the load is balanced between different nodes, Ser indicates the dependencies in the workload, and Trf reflects the effect of data transfers between nodes on the performance. The maximum value for η is 1, which means the workload achieves perfect scaling. In addition to the trace analysis, for calculating Ser DIMEMAS, a high-level network simulator is needed to simulate the traces for the ideal network scenario in which latency is assumed to be zero and unlimited bandwidth is available [6].

We study the scalability of the GPGPU-accelerated workloads to understand the scalability bottlenecks for our scientific workloads. *Alexnet* and *googlenet* are excluded, as these benchmarks do not communicate to solve the problem; rather, each individual image is classified using a single node.

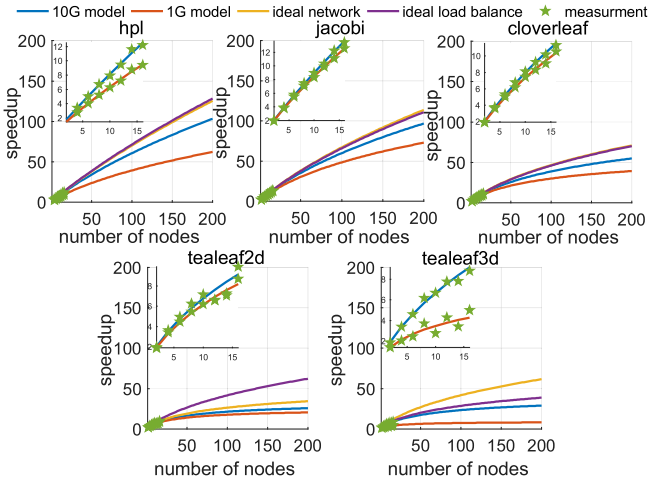

Fig. 5. Scalability of the GPGPU-accelerated benchmarks. 10G/1G model designates the extrapolations fitted to the data collected when 10GbE and 1GbE NICs have been used, respectively; ideal network is the case when traces are simulated assuming unlimited bandwidth between nodes; ideal load balance is when the load is perfectly distributed among nodes.

Figure 5 shows the extrapolated speedups for up to 200 nodes for different benchmarks and zooms in for 1 to 16 nodes to compare the models with the data we measured from the cluster. For these points, the fitting was done with an average r-squared (coefficient of determination) of 0.84. Our results show *hpl* and *jacobi* have better scalability compared to the *cloverleaf*, *tealeaf2d* and *tealeaf3d*. In addition to extrapolation for different network speeds, we simulated two additional scenarios to understand the scalability bottleneck for each workload. First, we simulated the traces using the ideal network in which the network is assumed to have zero latency and unlimited bandwidth. Speedups are improved on average by 1.28×, while the two most network-bound applications, *hpl* and *tealeaf3d*, see speedup improvements of 1.47×.

Another source of inefficiencies for scalability is the balance of work among different nodes. If the work is not distributed evenly between nodes, some nodes finish their tasks sooner than others and must wait to communicate with other nodes. The injected wait time for the load imbalance reduces the parallel efficiency of the cluster. For the second scenario, we simulated the case of perfect load balance between nodes by artificially making $LB = 1$. Note that in the case of ideal load balance, we used the traces with of 10GbE network and not the simulated ideal network. This decision is made to allow us to study the effects of each factor in isolation. On average, the speedups improve by 1.3×, while *tealeaf2d*

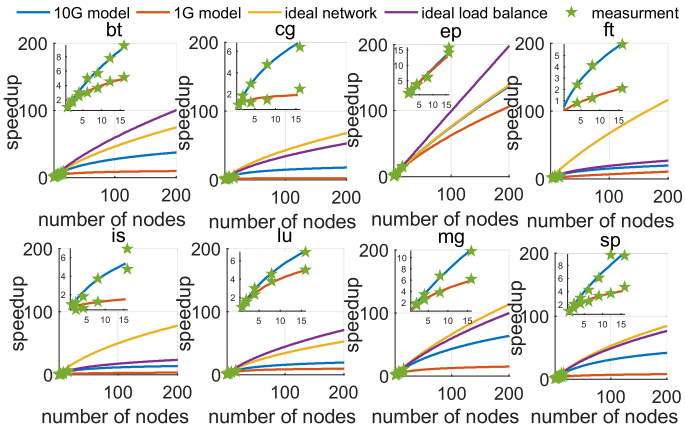

Fig. 6. Scalability of the NPB suite workload. 10G/1G model designates the extrapolations fitted to the data collected using the 10GbE and 1GbE NICs, respectively; ideal network is when traces are simulated with unlimited bandwidth; ideal load balance is when the load is perfectly distributed among nodes.

experiences an average of 1.88× speedup when its load is completely balanced among nodes. Even considering these two ideal cases, the scalability of the *cloverleaf*, *tealeaf2d*, and *tealeaf3d* is far from *hpl* and *jacobi* due to the smaller Ser factor which get affected by the data synchronization between host and device.

We also analyzed the scalability of our CPU workloads using the same methodology we used to analyze GPGPU scalability. Figure 6 shows scalability for the NPB suite. Again, it zooms in on the region showing 1 to 16 nodes to compare the models and the measured data. For these points, the fitting was done with an average r-squared (coefficient of determination) of 0.76. Figure 6 shows *bt*, *ep*, *mg*, and *sp* demonstrate better scalability compared to *cg*, *ft*, *is*, and *lu*.

To explain this behavior, we looked at the same ideal cases we simulated for GPGPU workloads. The ideal network improves the speedups on average by 2.12× for NPB suite, while the two most network-bound applications, *ft* and *is*, see speedup improvements of 3.62×. Thus, for *ft* and *is*, we concluded that high network traffic is the cluster bottleneck. An ideal load balance improves the speedup on average by 1.68×, while *cg* and *lu* experience an average of 2.14× speedups when their load is completely balanced between nodes. Therefore, the poor load balance among nodes is the bottleneck of *cg* and *lu* running on the cluster. CPU workloads show higher speedup improvements than the GPGPU-accelerated workloads when ideal cases are considered, as the CPU workloads do not have the data transfer overhead between the host and the device. This reduces the efficiency of Ser and makes it a more dominating factor.

5) Impact of GPGPU memory management models: We modified the *jacobi* benchmark to use different memory management methods, including host and device (H & D) copy, zero-copy, and unified memory. We analyze the performance differences between the different methods. Table III shows the runtime, L2 utilization, L2 read throughput, and memory stalls for different memory management methods and cluster sizes, normalized to the traditional host and device method. Unified memory results in the same performance as

		H & D	zero-copy	unified memory
8 nodes	runtime	1.00	7.21	0.99
	L2 usage	1.00	0.12	1.00
	L2 read throughput	1.00	0.09	1.00
	memory stalls	1.00	1.16	1.00
16 node	runtime	1.00	6.52	0.98
	L2 usage	1.00	0.12	1.00
	L2 read throughput	1.00	0.08	1.01
	memory stalls	1.00	1.14	0.99

TABLE III
RUNTIME, L2 USAGE, L2 THROUGHPUT, AND MEMORY STALLS OF JACOBI FOR DIFFERENT PROGRAMMING MODELS, NORMALIZED TO THE HOST AND DEVICE MEMORY MODEL.

the separate host and device memory model, as it automatically copies data between the host and device memory to leverage the locality. On the other hand, zero-copy increases the runtime by 6.8× for the TX1 cluster on average. This performance loss was unexpected for the TX1 cluster. We examined the performance-monitoring events using `nvprof` to try and determine the cause. The results in Table III show low L2 utilization, low read throughput, and high memory stalls when zero-copy is used. This clearly indicates that the cache hierarchy is completely bypassed when zero-copy is used. We also confirmed our findings with Nvidia. In the case of the TX1, caching is bypassed for zero-copy to maintain cache coherency. This results in large performance losses. On the other hand, unified memory is able to utilize the cache hierarchy and offers greater programming convenience than the traditional host and device copy method; however, data is still transferred between the host and device address spaces, albeit transparently.

6) Simultaneous CPU-GPGPU usage: Our GPGPU-accelerated benchmarks offload the heavy-duty calculations to the GPGPU and use a single CPU core for the communication and data transfers. Workload scheduling in heterogeneous systems is not a trivial task; both system and workload characteristics need to be considered [9]. It is interesting to provide estimates for the case when some work is offloaded to the CPU cores. Figure 7 shows the energy efficiency of *hpl*, measured in MFLOPS/W, when the ratio of workload between the one CPU core and GPGPU changes, normalized to the case where all calculations are offloaded to the GPGPU. As expected, as the fraction of work performed by the GPGPU decreases, the energy efficiency also decreases, since a single CPU core is less energy efficient than the GPGPU SMs. Results show that, on average, a single CPU core is 25% less energy efficient than the GPGPU SMs available on a single TX1. Based on these results, it is expected that by using the GPGPU and all of the CPU cores at the same time, the performance and energy efficiency would both improve.

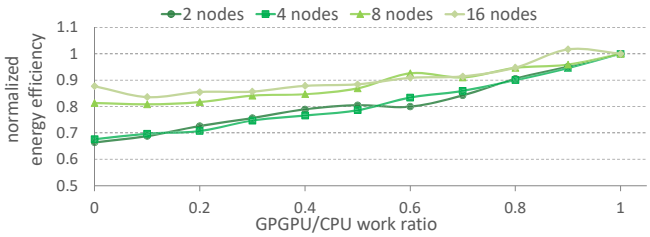

Fig. 7. Normalized energy efficiency of *hpl* when different ratios of CPU-GPGPU work is assigned, compared to the case where all of the load is on the GPGPU. Only one CPU core is being used per node.

configuration	throughput (GFLOPS)		energy efficiency (MFLOPS/W)	
	8 nodes	16 nodes	8 nodes	16 nodes
CPU+1G	43	54	410	261
CPU+10G	98	149	496	378
GPU+1G	30	37	287	190
GPU+10G	65	97	354	352
CPU+GPU[2]+1G	60	84	502	389
CPU+GPU[2]+10G	136	222	663	545

TABLE IV
THROUGHPUT AND ENERGY EFFICIENCY USING THE CPU AND GPGPU VERSIONS OF *hpl* AND THEIR COLLOCATION FOR DIFFERENT NETWORK SPEEDS.

As the *hpl* implementation does not use all of the CPU cores and the GPGPU at the same time, we performed the following experiment to provide a good estimation of the maximum performance for the case where all the CPU cores are used along with the GPGPUs. We ran the CPU and GPGPU versions of *hpl* together at the same time. To minimize contention on each node, we reserved one CPU core for the GPGPU data transfers and then simultaneously ran the CPU version of *hpl* on the remaining 3 CPU cores. Table IV summarizes the achieved throughput (GFLOPS) and energy efficiency (MFLOPS/W) using all CPU cores only, the GPGPU-accelerated version, and the CPU collocated with the GPGPU version, as explained above, for different cluster sizes and network speeds. Simultaneously using the GPGPU and CPU improves the energy efficiency by 1.5× compared to the best results obtained using the CPU and GPGPU alone. This highlights the benefit of the proposed cluster organization for ARM computing.

IV. COMPARISON TO OTHER SYSTEMS

To gain a better understanding of the performance and energy efficiency of the proposed cluster organization, we compare our cluster with other solutions. To ensure a meaningful comparison, we compare each component of our cluster with a similar component of existing solutions:

1) We compare the ARM mobile CPU of our cluster to emerging server-class ARM servers that rely on integrating many CPU cores.
2) We compare the integrated GPGPUs of our cluster with traditional discrete GPGPUs.

A. Many-Core ARM Server Comparison:

We compare the CPU performance of our 16-node cluster with an existing ARM-based server. We use a dual-socket Cavium ThunderX server for comparison. Table V compares the configurations of the ARM server and our cluster. The Cavium-based server contains two Cavium ThunderX SoCs, making it a 96-core machine that can run at a maximum of 2.0 GHz. We compare our 16-node TX1 cluster against one Cavium server, as both our cluster and the Cavium server consume approximately the same amount of power at max load (350 W).

We used the same libraries, software stack, and compiler options on the Cavium server to ensure a fair comparison. The number of processes for NPB workloads other than *ep*, *bt*, and *sp*, must be a power of two. Therefore, we run our benchmarks with 64 MPI processes, since 128 processes introduce a large amount of contention between threads. In order to provide a meaningful comparison, we run *ep*, *bt*, and *sp* with 64 MPI processes as well to compare the performance of the same number of cores on both systems. Running a multi-threaded process on a 96-core machine introduces new challenges. Task affinity of the parent process is usually fixed in order to avoid the overhead of migrations to different cores; however, we observed that fixing the task affinity of the parent process alone is not enough, since the migration of child threads across the fixed number of cores still introduces large overhead. Therefore, the task affinity of each MPI process must be fixed to one core. We found that doing this reduces the standard deviation of the average runtime from 29.3 seconds to 1.38 seconds across 10 different runs.

[2]Estimated using 3 CPU cores for the CPU version and 1 CPU core + GPGPU for the GPGPU version.

	Cavium ThunderX	NVIDIA TX1
number of nodes	1	16
ISA	64-bit ARM v8	64-bit ARM v8 & PTX
tech	28 nm	20 nm
CPU	2 × 48 cores	4 Coretex A57
CPU freq	2.0 GHz	1.73 GHz
GPGPU	-	2 Maxwell SM
L1 (I/D) size	78KB/32KB	48KB/32KB
L2 size	16 MB	2 MB
L3 size	–	–
SoC TDP	120 W	15 W

TABLE V
CONFIGURATIONS OF THE MANY-CORE ARM SERVER COMPARED TO OUR CLUSTER.

Table VI gives the runtime, power, and energy consumption of the Cavium server, normalized to our TX1 cluster. It shows a broad range of performance results for both systems. As an example, *ft*'s runtime increases by 23%, but *mg* is improved by 2.5× when running on the TX1 cluster. The improved results are surprising, as TX1 cluster substitutes the internal memory traffic of the Cavium SoC with network traffic, which has higher overhead. In addition, TX1 has lower CPU and DRAM frequencies than the Cavium server.

To understand the performance differences, we first looked at the benchmarks that perform better on the Cavium server. These benchmarks which are *cg*, *ft*, *is*, and *lu*, showed poor scalability in our scalability analysis due to high network traffic and poor load balance among nodes. To understand the benchmarks that perform worse on the Cavium server, we collect the ARM performance counters that are available on both the Cortex A57 and the Cavium ThunderX [3].

After collecting the raw counter values from the systems under the test, we added additional metrics, such as the miss ratios using the collected raw events. Then, we constructed an observation matrix, X, where each row contains our relative value of events/metrics for each benchmark on the Cavium server compared to our cluster. The response vector, Y, is

[3]An event even with the same name can look at different phenomena across different systems or even different versions of hardware from the same vendor. This problem was pointed out by Andrzej *et al.* and must be considered when performance monitoring counters are used for analysis across systems [14]. Therefore, we collected twelve counters that are part of ARMv8 PMUv3, and did not collect any additional counters that are only available for specific systems. Note that without a loss of generality, our analysis method is applicable when even more information and counters are available.

benchmark	normalized runtime	normalized power	normalized energy
bt	1.30	0.99	1.29
cg	0.94	0.93	0.87
ep	1.06	1.04	1.11
ft	0.77	1.05	0.81
is	0.76	0.98	0.75
lu	0.97	1.02	0.99
mg	2.50	0.96	2.40
sp	1.48	0.97	1.43

TABLE VI
NORMALIZED RUNTIME, POWER CONSUMPTION, AND ENERGY CONSUMPTION OF THE CAVIUM SERVER COMPARED TO 16-NODE TX1 CLUSTER FOR NPB SUITE.

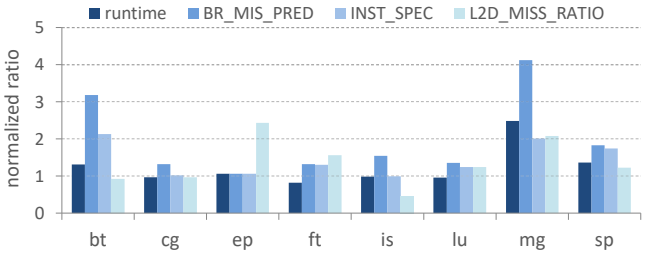

Fig. 8. Relative runtime and events/metrics chosen using PLS.

constructed based on the relative performance of the Cavium server to the TX1 cluster.

We used the statistical Partial Least Squares (PLS) methodology to identify the main components in our observation matrix that affect our response vector (relative performance of two systems) [3]. We observe that three principal components explain 95% of the variance of the observation matrix; thus, we use three components to model our response vector using linear regression. The top three variables that have the highest coefficient of regression values are then chosen. These are branch miss prediction, number of speculatively executed instructions, and the L2 miss ratio. Figure 8 shows the relative runtime and value of each of these three chosen events/metrics. Analyzing the chosen performance counters gives us the architectural conclusion that the branch predictor and L2 cache are the performance bottlenecks compared to the Cortex A57 designs. The higher L2 miss rate is a result of the Cavium ThunderX having less L2 cache data per core as well as contention between large number of threads running on a single machine. Based on previous Cavium designs (Octeon III), it is estimated that the chip has a short pipeline length to avoid large branch miss-prediction penalties [2]. However, our results show there are workloads for which the Cavium server exhibits higher branch miss prediction and speculatively executed instructions, which is a natural consequence of branch prediction misses.

To summarize, we observed two major classes of benchmarks comparing the performance of Cavium server and our TX1 cluster:

1) Benchmarks, such as *cg*, *ft*, *is*, and *lu*, which do not scale well, perform poorly on TX1 cluster mainly due to high network traffic and poor load balance among nodes.
2) Other benchmarks such as *bt*, *ep*, *mg*, and *sp*, perform better on the TX1 cluster due to the poor performance of the Cavium CPU cores. *Ep* has the highest L2 miss ratio. *Mg* proved to perform the worst on the Cavium server out of all of the scientific applications, as it proved to have the highest branch miss prediction, highest number of speculatively executed instructions, and the second worst L2 miss ratio. *Sp* showed relatively better branch prediction performance, but a higher L2 miss ratio compared to *bt*.

	MSI GTX 960	NVIDIA TX1
number of nodes	2	16
Cores	8 Maxwell SM	2 Maxwell SM
GPGPU freq	1.31 GHz	0.99 GHz
L2 size	1.04 MB	0.26 MB
Memory	4 GB GDDR5	4 GB LPDDR4 [4]
Memory bandwidth	112 GB/s	25 GB/s
TDP	130 W	15 W [4]

TABLE VII
CONFIGURATIONS OF THE DISCRETE AND SOC-CLASS GPGPUS.

B. Discrete GPGPU Comparison:

It is important to remember that GPGPUs are obviously not exclusive to mobile-class SoCs. The typical approach to GPGPUs for the purpose of accelerating mathematical operations has been to simply connect a discrete GPGPU to a workstation or server, usually via the PCIe slot on the motherboard. Our cluster made of mobile-class TX1 SoCs, however, takes a different approach to GPGPU acceleration by utilizing the GPGPUs integrated on the SoCs.

We compare the GPGPU performance of our 16-node TX1 cluster with two MSI GTX 960 discrete GPGPUs. For a meaningful comparison between the discrete and integrated GPGPUs, we picked a discrete GPGPU from the same family as the integrated GPGPU (Maxwell), and constructed a cluster using two of them. Using this configuration, we now have two clusters: one cluster of 16 TX1 nodes and one cluster of two discrete GPGPUs, each hosted on their own server, connected by 10GbE. Both clusters roughly use the same total power (350 W). Table VII compares the configuration of the discrete MSI GTX 960 GPGPU to the integrated GPGPU on TX1 cluster. Each TX1 SoC has 2 Maxwell Streaming Multiprocessors (SM), which are equivalent to 256 CUDA cores running at 0.9 GHz, whereas the GTX 960 has 8 SM (1024 CUDA cores) running at 1.3 GHz. While the integrated TX1 GPGPU shares the 4 GB of main memory between the CPU and GPU, the discrete GTX 960 has 4 GB of dedicated GPGPU memory.

Unfortunately, due to driver incompatibility issues with the ARM environment, we had to host the discrete GPGPUs on Xeon servers (E5-2630 v3). Our discrete GPGPU systems are equipped with 10GbE network connectivity and are connected to the same fileserver. We use the same software stack and libraries to ensure a fair comparison. Regarding our notation, 2 GTX means two discrete GTX 960 cards were used to obtain results. The power tax of Xeon servers (≈150 W) is comparable with other systems, as our Cavium server consumes ≈200 W without any load. We use one discrete GPGPU per host server, since some of our workloads only support this model.

Figure 9 shows the runtime and energy consumption of our GPGPU-accelerated workloads for different TX1 cluster sizes, normalized to the results obtained using two GTX cards. Figure 9 shows that there are three major classes of workloads

[4] Shared between CPU and GPGPU cores.

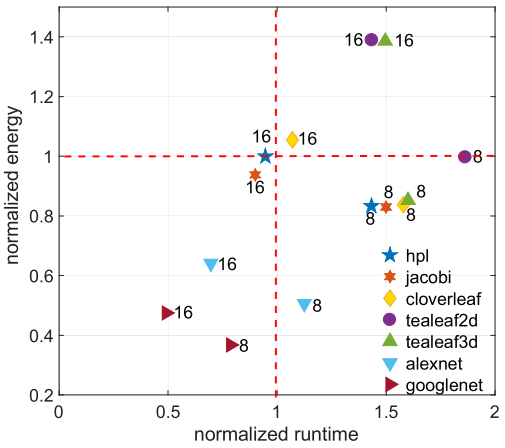

Fig. 9. Runtime and energy consumption of the TX1 cluster for different cluster sizes, normalized to two discrete GPGPUs.

when characterizing the energy consumption and performance trade-offs compared to the discrete GPGPU solution.

- When comparing the performance of 2 GTX cards with 8 TX1 nodes for the same number of CUDA cores, all workloads except *googlenet* consume less energy while also delivering less performance. Individual mobile-class ARM SoCs are mainly designed to deliver less performance, but consume much less power, leading to better energy efficiency. This is because, historically, increasing battery life has been the main design initiative.
- As shown in Figure 5, workloads such as *tealeaf2d*, *tealeaf3d*, *cloverleaf* do not scale-well. As the result, these workloads consume more energy and deliver less performance when a large number of nodes is used. Using a higher number of nodes does not necessarily improve performance despite the fact that energy consumption increases as nodes are added.
- There are workloads that deliver more performance while consuming less energy, since, for the same power budget, a higher number of nodes can be used. *Hpl*, *jacobi*, *alexnet*, and *googlenet* are examples of these types of workloads that scale well. As a result, both performance and energy efficiency improve at large cluster sizes.

Figure 9 shows running *alexnet* and *googlenet* on the proposed cluster organization improves both the performance and energy efficiency. We notice larger CPU utilization for these two emerging AI applications compared to the other benchmarks from scientific domains. For image classification, the CPU must decode JPEG images to prepare raw data for the GPGPUs to perform forward path computation on the deep neural network. Figure 10 shows the *alexnet* and *googlenet* speedups and unhalted CPU cycles per seconds of the different scale-out cluster sizes, normalized to the scale-up system. Figure 10 shows that even for the same GPGPU SM count (8 nodes), *googlenet* can leverage 64% more CPU cycles per seconds due to higher core count of the TX1 cluster. The proposed cluster organization gives a better CPU-GPGPU balance, which benefits these types of emerging applications.

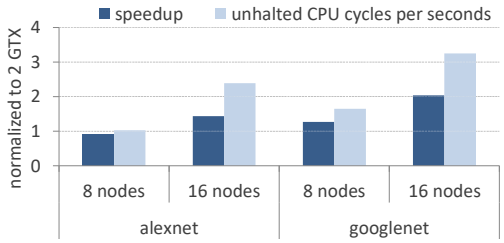

Fig. 10. AI workload speedup and unhalted CPU cycles per second for different scale-out cluster sizes normalized to the scale-up system.

V. Conclusions

In this paper, we proposed a novel cluster organization for mobile-class ARM SoCs. Our proposed cluster organization is based on two novel components. First, mobile-class ARM SoCs are connected to 10Gb network cards. Second, the integrated GPGPUs available on these SoCs are leveraged to increase the performance and energy efficiency of the cluster. When designing future ARM-based clusters, our results show that adding faster network connectivity is critical. Furthermore, with frameworks, such as CUDA that simplify programming, GPGPU acceleration becomes a more promising method for increasing the performance and energy efficiency of ARM-based clusters. Adding the GPGPUs to the makeup of the SoCs enables another direction for ARM computing. To the best of our knowledge, this is the first work to analyze integrated GPGPUs of ARM SoCs at the cluster level in depth. Our set of collected benchmarks, ClusterSoCBench, provide a path toward establishing a standardized, optimized benchmark suite capable of stressing different architectural characteristics to facilitate research in this area [1].

We also extended the Roofline model to visually describes the peak computational capacity for the proposed cluster organization. Using our cluster, we studied the scalability and characteristics of our workloads. Our results show that, on average, faster network connectivity improves the performance and energy efficiency of the cluster by 2× and 25%, respectively, across a broad range of HPC/AI benchmarks when compared to the standard 1GbE. We showed that using the proposed cluster improves the energy efficiency of the standard High Performance Linpack benchmark, *hpl*, compared to using only the CPU cores and 1GbE. We also compared our cluster with other existing solutions, such as traditional discrete GPGPUs and server-class many-core SoCs. Our results showed that a large number of ARM cores on a single chip does not necessarily guarantee better performance. We quantified comparable results for the same family and power budget with traditional discrete GPGPUs. Finally, we showed image classification applications using deep neural networks can leverage better CPU-GPGPU balance of SoC based clusters leading to better performance and energy efficiency compared with discrete GPGPUs of same family and power budget.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive comments. This project is partially supported by NSF grant 1305148 and 1438958.

REFERENCES

[1] "ClusterSoCBench: set of benchmarks to stress the cluster of GPGPUs." https://github.com/scale-lab/ClusterSoCBench, 2017.

[2] "Investigating Cavium's ThunderX: The First ARM Server SoC With Ambition," http://www.anandtech.com/show/10353/investigating-cavium-thunderx-48-arm-cores/7, 2017.

[3] H. Abdi, "Partial least square regression (pls regression)." in *Encyclopedia for research methods for the social sciences*. Sage, 2003, pp. 792–795.

[4] D. Abdurachmanov, B. Bockelman, P. Elmer, G. Eulisse, R. Knight, and S. Muzaffar, "Heterogeneous high throughput scientific computing with APM x-gene and intel xeon phi," *CoRR*, vol. abs/1410.3441, 2014.

[5] R. Azimi, X. Zhan, and S. Reda, "How good are low-power 64-bit socs for server-class workloads?" in *Workload Characterization (IISWC), 2015 IEEE International Symposium on*, pp. 116–117.

[6] B. S. Center, "BSC tools." https://tools.bsc.es/, 2017.

[7] S. Cook, *CUDA programming: a developer's guide to parallel computing with GPUs*. Newnes, 2012.

[8] B. Cumming, "Cuda-stream," https://github.com/bcumming/cuda-stream, 2016.

[9] K. Dev and S. Reda, "Scheduling challenges and opportunities in integrated cpu+gpu processors," in *Proceedings of the 14th ACM/IEEE Symposium on Embedded Systems for Real-Time Multimedia*, ser. ESTIMedia'16, 2016, pp. 78–83.

[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[12] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, "The hpc challenge (hpcc) benchmark suite," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, p. 213.

[13] J. Maqbool, S. Oh, and G. C. Fox, "Evaluating arm hpc clusters for scientific workloads," vol. 27, no. 17. Wiley Online Library, 2015, pp. 5390–5410.

[14] A. Nowak, D. Levinthal, and W. Zwaenepoel, "Hierarchical cycle accounting: a new method for application performance tuning," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*, pp. 112–123.

[15] Nvidia, "High performance linpack for CUDA." https://developer.nvidia.com/rdp/assets/cuda-accelerated-linpack-linux64, 2016.

[16] E. Padoin, D. De Olivera, P. Velho, P. Navaux, B. Videau, A. Degomme, and J.-F. Mehaut, "Scalability and energy efficiency of hpc cluster with arm mpsoc," in *Proc. of 11th Workshop on Parallel and Distributed Processing*, 2013.

[17] Parallel-forall, "Jacobi for CUDA." https://github.com/parallel-forall/code-samples/tree/master/posts/cuda-aware-mpi-example, 2016.

[18] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero, "Supercomputing with commodity cpus: Are mobile socs ready for hpc?" in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 40.

[19] N. Rajovic, A. Rico, F. Mantovani, D. Ruiz, J. O. Vilarrubi, C. Gomez, L. Backes, D. Nieto, H. Servat, X. Martorell *et al.*, "The mont-blanc prototype: An alternative approach for hpc systems," in *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 2016, pp. 444–455.

[20] N. Rajovic, A. Rico, N. Puzovic, C. Adeniyi-Jones, and A. Ramirez, "Tibidabo: Making the case for an arm-based hpc system," vol. 36. Elsevier, 2014, pp. 322–334.

[21] N. Rajovic, L. Vilanova, C. Villavieja, N. Puzovic, and A. Ramirez, "The low power architecture approach towards exascale computing," vol. 4, no. 6. Elsevier, 2013, pp. 439–443.

[22] C. Rosas, J. Giménez, and J. Labarta, "Scalability prediction for fundamental performance factors," vol. 1, no. 2, 2014, pp. 4–19.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[24] UK-MAC, "Cloverleaf for CUDA." https://github.com/UK-MAC/CloverLeaf_CUDA, 2016.

[25] ——, "Tealeaf for CUDA." https://github.com/UK-MAC/TeaLeaf, 2016.

[26] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," vol. 52, no. 4. ACM, 2009, pp. 65–76.