Scale-Out vs Scale-Up: A Study of ARM-based SoCs on Server-Class Workloads

REZA AZIMI, TYLER FOX, WENDY GONZALEZ, and SHERIEF REDA,
Brown University, USA

ARM 64-bit processing has generated enthusiasm to develop ARM-based servers that are targeted for both data centers and supercomputers. In addition to the server-class components and hardware advancements, the ARM software environment has grown substantially over the past decade. Major development ecosystems and libraries have been ported and optimized to run on ARM, making ARM suitable for server-class workloads. There are two trends in available ARM SoCs: mobile-class ARM SoCs that rely on the heterogeneous integration of a mix of CPU cores, GPGPU streaming multiprocessors (SMs), and other accelerators, and the server-class SoCs that instead rely on integrating a larger number of CPU cores with no GPGPU support and a number of IO accelerators. For scaling the number of processing cores, there are two different paradigms: mobile-class SoCs that use scale-out architecture in the form of a cluster of simpler systems connected over a network, and server-class ARM SoCs that use the scale-up solution and leverage symmetric multiprocessing to pack a large number of cores on the chip. In this article, we present ScaleSoC cluster, which is a scale-out solution based on mobile class ARM SoCs. ScaleSoC leverages fast network connectivity and GPGPU acceleration to improve performance and energy efficiency compared to previous ARM scale-out clusters. We consider a wide range of modern server-class parallel workloads to study both scaling paradigms, including latency-sensitive transactional workloads, MPI-based CPU and GPGPU-accelerated scientific applications, and emerging artificial intelligence workloads. We study the performance and energy efficiency of ScaleSoC compared to server-class ARM SoCs and discrete GPGPUs in depth. We quantify the network overhead on the performance of ScaleSoC and show that packing a large number of ARM cores on a single chip does not necessarily guarantee better performance, due to the fact that shared resources, such as last-level cache, become performance bottlenecks. We characterize the GPGPU accelerated workloads and demonstrate that for applications that can leverage the better CPU-GPGPU balance of the ScaleSoC cluster, performance and energy efficiency improve compared to discrete GPGPUs.

CCS Concepts: • **Computer systems organization** → **Heterogeneous (hybrid) systems**;

Additional Key Words and Phrases: ARM computing, GPGPU acceleration, scale-out clusters

ACM Reference format:

Reza Azimi, Tyler Fox, Wendy Gonzalez, and Sherief Reda. 2018. Scale-Out vs Scale-Up: A Study of ARM-based SoCs on Server-Class Workloads. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 3, 4, Article 18 (August 2018), 23 pages.

<https://doi.org/10.1145/3232162>

This project is partially supported by NSF grant 1305148 and 1438958.

Authors' addresses: R. Azimi, Brown University, School of Engineering, 184 Hope Street, Providence, RI, 02906; email: reza_azimi@brown.edu; T. Fox, Brown University, School of Engineering, 184 Hope Street, Providence, RI, 02906; email: tyler_fox@brown.edu; W. Gonzalez, Brown University, School of Engineering, 184 Hope Street, Providence, RI, 02906; email: wendy_gonzalez@brown.edu; S. Reda, Brown University, School of Engineering, 184 Hope Street, Providence, RI, 02906; email: sherief_reda@brown.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 ACM 2376-3639/2018/08-ART18 \$15.00

<https://doi.org/10.1145/3232162>

1 INTRODUCTION

Power consumption is the limiting factor when looking to improve the performance at the chip and cluster level. Emerging architectures, such as low-power ARM processors, enable a new major direction for increasing the performance of large-scale clusters, given a power constraint. Compared to the available processors in the market, ARM processors have three main advantages:

- (1) ARM processors can be customized by design for particular workload characteristics compared to the available solutions that must be purchased as general processors.
- (2) ARM cores have a simpler design and consume much less power compared to the available solutions. Given a fixed power budget, a higher number of ARM cores can be used. Therefore, using ARM processors can improve both performance and energy efficiency for modern workloads that have a high degree of parallelism.
- (3) ARM processors provide another source of supply, which creates competitions in the market and improves the cost efficiency of clusters.

Examining existing and planned server-based ARM System-on-a-Chip (SoC) processors reveals that upcoming server-class SoCs are trending toward a scale-up solution that uses Symmetric Multi-Processing (SMP) to include a large number of CPU cores on the chip. 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. In addition to CPU cores, these SoCs include IP blocks for a memory controller and high-end network and I/O connectivity.

The makeup of server-class SoCs is different from mobile-class ARM SoCs that emphasize heterogeneous integration of Graphical Processing Unit (GPU) Streaming Multiprocessors (SMs) at the expense of fewer CPU cores. While these GPU cores have historically been dedicated solely to graphics, modern mobile-class ARM SoCs incorporate general-purpose GPUs (GPGPU) that can be programmed to accelerate workloads. Compared to the discrete GPGPUs, the integrated GPGPUs tend to have lower specifications, such as slower clock speeds and fewer SMs. For comparable core counts with scale-up solutions, these low-end mobile-class ARM SoCs must use a scale-out architecture in the form of a cluster connected over a network.

In this article, we present the ScaleSoC cluster for ARM server computing based on a scale-out architecture that uses GPGPU-accelerated ARM SoCs connected with a fast network. A fast network reduces the overhead of communication across nodes and allows for heavy computation to be efficiently offloaded to GPGPU units. We characterize the performance and energy efficiency of each computing component of the ScaleSoC cluster and compare it with similar counterparts. The CPU performance of ScaleSoC is studied using the server-class ARM SoCs, and GPGPUs available on the ScaleSoC are analyzed in contrast with discrete GPGPUs.

We consider a broad range of CPU and GPGPU-accelerated server-class workloads, including latency-sensitive transactional workloads, MPI-based CPU and GPGPU-accelerated scientific applications, and emerging artificial intelligence (AI) workloads for deep learning. We seek answers to two broad, important questions: (1) What are the main architectural and system organization factors that impact the performance and energy efficiency of each class of workload? and (2) What are the characteristics of each workload class that benefit the most from each architecture? While previous efforts have made progress toward answering similar questions, they focus exclusively on comparing one type of SoC architecture against x86 and only focus on the CPU cores of ARM SoCs [7, 10, 23, 29, 34, 36, 38, 39]. However, this work is not about ARM versus x86; rather, we focus on the ScaleSoC cluster and compare it with different ARM-based architectures and system organizations. We believe the ScaleSoC cluster opens a new direction for ARM-based computing. The contributions of this article are as follows:

- This is the first study to analyze different scaling paradigms for ARM SoCs. We propose the ScaleSoC cluster organization as a scale-out solution and analyze it in contrast to scale-up solutions for a broad range of server-class workloads. To overcome the lack of benchmarks to stress the available GPGPUs on mobile-class ARM clusters, we collected a set of benchmarks. We show that a fast network for communication and GPGPU acceleration open a new direction for ARM computing.
- We show that despite running at a lower frequency and having higher network overhead compared to scale-up systems, there exists a class of GPGPU-accelerated workloads that can exploit the scale-out architecture. For GPGPU-accelerated scientific workloads that scale well, scale-out clusters improve both performance and energy efficiency compared to scale-up solutions, as they are able to use more processing units for a fixed power budget.
- For image inference using deep neural networks, we look at both popular deep learning frameworks: Caffe and Tensorflow. We show that for this emerging class of AI workload, scale-out clusters improve both performance and energy efficiency due to having a better CPU-GPGPU balance than scale-up systems that use discrete GPGPUs.
- Our results show that despite the industrial trends to use scale-up architecture in designing ARM processors, shared resources such as caches become the performance bottleneck of many-core ARM SoCs to the extent that they perform worse than scale-out solutions running at lower frequencies and with network overhead.

The organization of this article is as follows. In Section 2, we give the background and motivate our study. Section 3 describes the hardware and software organization that we used to analyze ScaleSoC cluster. In Section 4, we compare ScaleSoC cluster with scale-up systems. Finally, we review related work in Section 5 and summarize the main conclusions and directions for future work in Section 6.

2 MOTIVATION AND BACKGROUND

Using ARM cores for server-class workloads is based on the philosophy of obtaining improved performance and energy efficiency by using cores that deliver less performance individually in exchange for significantly lower power consumption. The lower per-core power consumption allows a higher number of cores to be used given the same power budget of a traditional server. We observed that industrial designs use scale-up ARM SoCs while previous academic ARM studies are done using scale-out architectures. Thus, we study the scaling debate for ARM SoCs.

Network communication has a major role in scale-out architectures. Previous attempts to build high-performance 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 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.

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 [38]. Second, the programming model becomes extremely complicated when a cluster of GPGPUs is 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 [37]. Thus, Mont-Blanc only focused on evaluating the performance of a single GPGPU node instead of the whole cluster. Due to the lack of workloads, 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. To overcome the workload problem, we collected and ported a set of representative workloads, clusterSoCBench, that we made available for other researchers [3].

Based on our observations on modern mobile-class ARM SoCs, we developed ScaleSoC as a scale-out cluster organization for GPU-accelerated ARM SoCs. ScaleSoC is different from previous scale-out ARM clusters [34, 37, 38] in two major ways:

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

Unlike previous efforts to study the ARM vs x86 debate, and only focusing on the CPU component of mobile class ARM SoCs, we investigate the scale-up vs scale-out debate using a wide range of server-class workloads to determine how to best design future ARM SoCs.

3 SCALESOC: ORGANIZATION AND ANALYSIS

3.1 Hardware Organization

We use 16 Jetson TX1 boards to build our ScaleSoC cluster. Each TX1 SoC has four Cortex A57 CPU cores running at 1.73GHz¹ and two Maxwell SMs, for a total of 256 CUDA cores running at 0.9GHz. Each Jetson board has 4GB 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 11.72GB/s and 21GB/s, respectively [14, 27]. Each Jetson board also has 16GB 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 article, 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, eight nodes means eight TX1 Jetson boards are used to obtain results.

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 ScaleSoC 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. The storage server and all nodes are connected using the 10GbE NIC with a Cisco 350XG managed switch that has a bisection bandwidth of 120Gb/s. For experiments with 1GbE, a 48-port Netgear switch is used.

3.2 Software Stack

We consider a large range of benchmarks that are representative of HPC/AI applications and latency-sensitive transactional workloads. For latency-sensitive transactional workloads, we use memcached and web serving using the Apache webserver to present stateful and stateless transactional workloads, respectively [16, 18]. We use the data caching load generator from Cloudsuite as the client for memcached [17]. We developed our own scripts as clients for web serving to stress

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

Table 1. Summary of the GPGPU Accelerated Workloads Collected in ClusterSoCBench

tag	problem description	input size
hpl	Solving linear equations ($Ax = b$) [32]	$N = 20,000$
cloverleaf	Solving compressible Euler equations [41]	4,080 cells and 100 steps
tealeaf2d	Solving linear heat conduction equation in 2D [42]	4,000 2D cells and 10 steps
tealeaf3d	Solving linear heat conduction equation in 3D [42]	250 3D cells and 5 steps
jacobi	Solving poisson equation on a rectangle [35]	16,800×16,800 matrix size
alexnet	Parallelized Caffe to classify ImageNet images using the AlexNet model [20, 22]	1,000 images
googlenet	Parallelized Caffe to classify ImageNet images using the GoogleNet model [20, 40]	1,000 images

the system in an open-loop fashion. We use separate machines to generate the loads and make sure that the load on the clients does not create client-side delay. Our Apache server is serving pages from PmWiki that stores articles in flat text files [30]. We chose PmWiki to remove the need for the database and focus on stressing the CPU components of our solutions and not IO. Our web serving clients only request the same page to assure data exists in the OS file caches. We make sure our servers are warmed up before measuring the performance.

To stress our cluster with traditional scientific workloads, we use NAS Parallel Benchmarks (NPB) with different class sizes to evaluate the CPU performance of ScaleSoC cluster. We used OpenMPI as our MPI library. Unless otherwise noted, we use all the CPU cores on each node to run our benchmarks. NPB is compiled using the `-O3` flag on all systems. We also used the CPU version of `hpl` and `stream` from the HPCC suite [27].

The lack of a standard benchmark suite is one of the barriers to evaluate the performance of GPGPU-accelerated clusters. We identified and collected a set of benchmarks (ClusterSoCBench) that is able to stress GPGPU-accelerated ARM clusters [3]. Table 1 shows the list of benchmarks found in ClusterSoCBench and the input sizes we used. Except *hpl*, which is the most common benchmark to evaluate HPC clusters, we gather four more benchmarks from scientific domain. All workloads were compiled with `-O3` optimization. ClusterSoCBench can be found on GitHub to facilitate research in the areas that demand the benchmarks [3]. To analyze the performance of a single node, we used workloads from both the ScaleSoCBench and Rodinia benchmarks suites [13].

For the emerging AI domain, we consider image inference using deep neural networks on representative deep learning frameworks: Caffe [20] and Tensorflow [5]. Tensorflow is designed to work on distributed systems; Caffe, on the other hand, is a single node framework. To stress the cluster using Caffe, we developed our own scripts that distribute images to be classified in parallel across all nodes. For image inference, the AlexNet and GoogleNet deep neural network models [22, 40] are used to classify the ImageNet dataset.

In our experiments, we used standard techniques, such as fixing CPU affinity and choosing performance DVFS governors, to get the best performance of the benchmarks as they come, without modifying or hand-tuning any of the code/libraries for specific systems. This decision was made to make the comparison of systems fair and to account for the fact that architecture-dependent optimizations are in different stages for different systems. Our main goal is to study different architectures, not to compare different optimization techniques for different architectures. Further performance and energy efficiency improvements are expected when more aggressive architecture optimization is added. We use MPI to leverage multiple cores on each node. Although the hybrid use of MPI+OpenMP can also be used to leverage the multi-core architecture, we stick with MPI, as not all benchmarks can take advantage leverage of hybrid MPI+openMP. Furthermore, the hybrid MPI+openMP configuration changes when comparing systems with different numbers of nodes

Fig. 1. The experimental setup overview of ScaleSoC cluster. Sixteen TX1 boards are connected with both 10GB and 1GB switches.

and cores per node. Our goal is to fix the software configurations to be as similar as possible to study different architectures.

We installed Ubuntu 14.04 with kernel 3.10.96 on all 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 the source and 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. We used PHP version of 5.5 for our Apache server. We used Tensorflow 1.0 alpha for our experiments. We used the same libraries, software stack, and compiler options for both the scale-out and scale-up solutions to ensure a fair comparison.

3.3 Performance and Power Measurement

We analyze the core performance of our systems by fully instrumenting all servers/nodes to record performance-monitoring counters for both the CPU and GPGPU cores. Performance-monitoring counters are collected using Linux perf, GPGPU events and metrics are collected using nvprof. Performance-monitoring counters are collected on different runs than the power/performance measurements. For CPU performance counters, on each run, we collected the same number of counters as actual available PMU registers. All performance counters were collected over many runs to avoid multiplexing. The CPU affinity of all processes are set to fixed logical cores in all runs.

The power consumption of each server platform is measured by sensing the external current at the 120V AC socket with a sampling rate of 10Hz. When measuring the power consumption of the cluster, we measure the power of all nodes used in each experiment. Figure 1 shows the experimental setup overview of ScaleSoC cluster. A separate machine is used to collect the power measurements from the power meter using USB serial connection. We consider two metrics for energy efficiency: (1) the total energy consumption and (2) the floating-point operations per second (FLOPS) per watt.

Fig. 2. Speedup gained by using the 10GbE NIC compared to using the 1GbE for different cluster sizes.

Fig. 3. Normalized energy consumption when the 10GbE NIC is used compared to using the 1GbE for different cluster sizes.

3.4 Benefit of Faster Network

Network plays an important role in scale-out clusters as nodes must use network for data transfers instead of shared memory that is used in scale-up solutions. To upgrade the network on our Jetson TX1 boards that are equipped with the standard 1GB ethernet NIC, we leverage the PCIe bus available on these boards and upgrade the network to 10GbE.² Compared to the on-board 1GbE controller, the addition of the 10GbE card improves the average throughput between two TX1 nodes from 0.53Gb/s to 3.13Gb/s, measured using the *iperf* tool, and the average latency of the ping-pong latency test from 0.4ms to 0.05ms, measured using Latency-Bandwidth benchmark [27].

Figures 2 and 3 show the speedup and energy consumption when the 10GbE cards are being used compared against the standard 1GbE on the Jetson boards. Results are average over all benchmarks. We use MPI to leverage all cores on the system. As an example, results for 16 nodes utilize all 64

²The ARM environment is still immature and, to upgrade the network, we had to modify both the TX1 kernel and the network driver. Initially the throughput of the 10GB network card would often drop to zero and hang when stressed with a heavy load. This was found to be an issue with the input-output memory management unit (IOMMU) in the TX1 kernel, which is responsible for connecting the direct memory access (DMA) mapping-capable input-output bus to the system's main memory without involving the CPU. After fixing the kernel issue, we observed large packet loss when the 10GbE card was being used. We found disabling the use of paged buffers in the driver fixed the packet loss issue [1].

Table 2. The Upper Bound of Fast Network Improvement for Various Workloads

benchmark	GPGPU accelerated	improvement	benchmark	GPGPU accelerated	improvement
hpl	Yes	3.94×	ep	No	1.01×
jacobi	Yes	3.40×	ft	No	2.34×
cloverleaf	Yes	2.53×	is	No	1.81×
tealeaf2d	Yes	2.15×	lu	No	1.44×
tealeaf3d	Yes	3.53×	mg	No	1.26×
bt	No	1.18×	sp	No	1.34×
cg	No	1.76×	hpl	No	1.11×

Results are obtained by comparing the simulated execution time of workloads under ideal network scenario and execution time using the 10GbE network.

cores available on ScaleSoC. Adding these cards on the PCIe slots adds to the power consumption of the cluster ($\approx 5W$ per card). Depending on the network utilization of the workloads, improvement in runtime is attained, which increases the overall energy efficiency of the cluster. Both speedup and energy efficiency further increase in larger clusters when using the 10GbE network, as the higher amount of inter-node communication in larger clusters results in the network having a greater impact. *bt* and *cg* are examples of network-intensive workloads in which both speedup and energy efficiency are improved as cluster size increases. On the other hand, for workloads, such as *ep*, that do not utilize the network, 10GbE cards decrease the energy efficiency because execution time remains the same and the extra 10GbE cards increase the power consumption. On average, for the 16-node cluster, we achieve a $2\times$ speedup and 15% improvement in energy efficiency when the 10GbE is used.

To further analyze the benefit of the fast network, we used simulation to obtain the upper-bound of fast network improvements compared to the available 10GbE cards. We used DIMEMAS to simulate our workloads with the *ideal network* scenario in which network latency is assumed to be zero and unlimited bandwidth is available [12]. GoogleNet and AlexNet are excluded, as DIMEMAS only simulates MPI workloads. Table 2 shows the upper-bound of fast network improvement compared to the available 10GbE cards. Our results show that GPGPU-accelerated workloads benefit more from the fast network, as network communication has a higher overhead for these workloads. Because GPUDirect technology is not supported on Jetson TX1 boards, any network communication must be handled by the CPU first and then transferred to the GPGPU through the main memory.

Conclusion 1: Previous mobile-class ARM clusters used standard 1GbE with the assumption that ARM cores are not strong enough to leverage the extra bandwidth. Our results show that, similar to server class ARM SoCs that leverage fast network accelerators, processors on modern mobile-class ARM SoCs are strong enough to benefit from faster network controllers instead of the standard 1GbE. Using 10GbE, ScaleSoC cluster reduces the runtime of a wide range of benchmarks by $2\times$ on average.

3.5 Benefit of GPGPU Acceleration

Traditionally, GPGPU-accelerated scientific 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, as both system and workload characteristics need to be considered [15]. It is interesting to provide estimates for the case when some work is offloaded to the CPU cores. Figure 4 shows the energy efficiency of *hpl* when the ratio of workload between the one CPU core and GPGPU changes, normalized to the case where all calculations are offloaded

Fig. 4. 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.

Table 3. Throughput and Energy Efficiency Using the CPU and GPGPU Versions of *hpl* and Their Collocation for Different Network Speeds

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+1G	60	84	502	389
CPU+GPU+10G	136	222	663	545

The hybrid CPU-GPU results are estimated using three CPU Cores for the CPU Version and one CPU Core + GPGPU for the GPGPU version.

to the GPGPU. 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 both GPGPU SMs. However, as the cluster size grows, offloading work to the CPU has less effect on energy efficiency because network communication causes a higher overhead for the GPGPU, as shown in Table 2. Based on these results, it is expected that, by using the GPGPU and all CPU cores at the same time, performance and energy efficiency would both improve.

As the GPGPU-accelerated *hpl* implementation does not use all CPU cores and GPGPU SMs at the same time using hybrid MPI+OpenMP, we performed the following experiment to provide a good estimation of the maximum performance for the case where all 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 three CPU cores. The GPGPU-accelerated *hpl* uses the same number of logical MPI processes as number of nodes in the cluster to use the GPGPU. The CPU version of *hpl* uses the number of nodes multiplied by the three remaining cores as the number of logical MPI processes to leverage all remaining cores on the cluster. Table 3 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 improve the throughput and energy efficiency by 1.4× compared to the best results obtained using the CPU and GPGPU alone. Additionally, the faster 10GbE network improves

Table 4. Configuration Comparison of the Cavium Server and ScaleSoC Cluster

	Cavium ThunderX	ScaleSoC
number of nodes	1	16
ISA	64-bit ARM v8	64-bit ARM v8 & PTX
tech	28nm	20nm
CPU	2 × 48 cores	4-Cortex A57
CPU freq	2.0GHz	1.73GHz
GPGPU	-	2 Maxwell SM
L1 (I/D) size	78KB/32KB	48KB/32KB
L2 size	16MB	2MB
L3 size	-	-
DRAM	128GB DDR4-2133	4GB LPDDR4-1600
max power	≈350W	≈350W

the throughput and energy efficiency by 2.4× and 1.4×, respectively, on average, compared to the standard 1GbE. These results highlight the benefit of each component of the proposed ScaleSoC for ARM computing. Tibidabo and SnowBall ARM cluster [34, 38] achieved 120 and 206MFLOPS/W, respectively, while ScaleSoC cluster achieved 524MFLOPS/W. Comparing with previous ARM cluster, proposed cluster organization shows new directions for ARM computing.

Conclusion 2: Our results show that leveraging the available GPGPUs on the mobile-class ARM SoCs brings dramatic improvements to the performance and energy efficiency of ARM clusters. For the standard hpl benchmark, the throughput and energy efficiency of our cluster are improved by 40%.

4 COMPARISONS TO SCALE-UP SOLUTIONS

In this section, we first introduce the organization of scale-up systems we used for comparison; then, we compare ScaleSoC results to the scale-up solutions for various server class workloads.

4.1 Organization of Scale-Up Solutions

For meaningful comparison, compared systems must be configured as similarly as possible. Although previous works studied ARM versus x86 [7, 10, 23, 29, 34, 36, 38, 39], we compare the CPU performance of ScaleSoC only with scale-up ARM servers. When ARM servers are compared with x86, differences can be caused by the difference in architectures, instruction sets, and software stacks even if the fabrication technology is the same. Conclusions cannot be inferred as general rules since each factor is not studied individually.

For experimental study, finding two ARM systems on the market that have the same instruction set and fabrication technology is not a trivial task. We compare the CPU performance of our 16-node ScaleSoC cluster with an existing ARM-based server which uses a many-core (scale-up) architecture for ARM based SoCs and has the same instruction set (ARMV8). We use a dual-socket Cavium ThunderX server for comparison. Table 4 compares the configurations of the ARM server and ScaleSoC cluster. The Cavium-based server contains two Cavium ThunderX SoCs, making it a 96-core machine that can run at a maximum of 2.0GHz. 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 (350W).

Table 5. Configuration Comparison of the Discrete GPGPU Cluster with the ScaleSoC Cluster

	MSI GTX 960	NVIDIA TX1
Number of nodes	2	16
Cores	8 Maxwell SMs	2 Maxwell SMs
GPGPU freq	1.31GHz	0.99GHz
L2 size	1.04MB	0.26MB
Memory	4GB GDDR5	4GB LPDDR4 ³
Memory bandwidth	112GB/s	25GB/s
Max power	≈350W	≈350W

We acknowledge that the Cavium SoC is fabricated with 28nm technology while TX1 SoCs use 20nm technology. At the time of this study, there is no scale-up ARM server in the market with the ARMV8 instruction set fabricated with 20nm technology; therefore, speculation is the only option. However, speculating the performance and power consumption of a different fabrication generation based on the measurement of its successor/predecessor is not a trivial task and speculation is always highly error prone. As an example, when ARM introduced Cortex A57 as the successor to Cortex A15, ARM's internal projection was that A57 can achieve 25–50% better IPC at a cost of 20% higher power consumption for the same fabrication technology. However, comparing the results of Cortex A15 and A57 fabricated by Samsung with the same technology shows a wide range of performance and power consumption results between two systems that are highly dependant on workload characteristics [2]. Thus, based on the available systems on the market, we choose to compare ScaleSoC with the Cavium server since (1) both systems entered the market approximately at the same time (3Q15), (2) both systems have the same instruction set (ARMV8), (3) we fix our power budget for a fair comparison (350 W), and (4) we use the same libraries, software stack, and compiler options on the Cavium server to ensure a fair comparison. As fabrication technology is out of our control, we try to compensate by choosing systems with the same market availability dates and power budgets.

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. The ScaleSoC cluster, made of mobile-class SoCs, however, takes a different approach to GPGPU acceleration by utilizing the GPGPUs integrated on the SoCs.

We compare the GPGPU performance of our ScaleSoC 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. Unfortunately, due to driver incompatibility issues with the ARM environment, we had to host the discrete GPGPUs on Xeon servers (E5-2630 v3) with 16GB of DDR3 main memory. Unless otherwise noted, we quantified the effect of different hosts for our GPGPU-accelerated workloads to be negligible. Both clusters roughly use the same total power (≈350 W). The power tax of Xeon servers (≈150 W) is comparable with other systems, as our Cavium server consumes ≈200 W without any load. Our discrete GPGPU cluster is connected to the same 10GbE switch and to the same file server. We use one discrete GPGPU per host server, since some of our workloads only support this model. Table 5 compares the configuration of the discrete MSI GTX 960 GPGPU to

the integrated GPGPU on TX1 cluster. Each TX1 SoC has two Maxwell SMs, which are equivalent to 256 CUDA cores running at 0.9GHz, whereas the GTX 960 has eight SMs (1024 CUDA cores) running at 1.3GHz. While the integrated TX1 GPGPU shares its 4GB of main memory between the CPU and two GPU SMs, the discrete GTX 960 has 4GB of dedicated GPGPU memory shared between eight SMs. We use the same software stack and libraries to ensure a fair comparison. Regarding our notation, two GTX means two discrete GTX 960 cards were used to obtain results.

4.2 Comparison

To ensure a meaningful comparison, we compare each component of our ScaleSoC cluster with a similar component of existing solutions.

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

We evaluate and analyze the performance of ScaleSoC cluster on four major classes of server-class workloads:

- The GPGPU-accelerated scientific workload.
- The emerging deep neural network workload.
- The latency-sensitive transactional CPU workload.
- The classical MPI-based scientific CPU workloads.

4.2.1 The GPGPU-Accelerated Scientific Workloads.

Single Node Results: First, we compare the performance of a single node of the ScaleSoC cluster to the discrete GPGPU hosted on a Xeon server. This comparison enables us to understand the performance of a single node, which is later needed to analyze the performance of the cluster. Table 6 shows a comparison of the performance, power, and energy consumption with the discrete GPGPU. As the results confirm, a single TX1 is designed for a different design trade-off point in performance and power consumption. Compared to a single TX1 node, the discrete GPGPU delivers a $3.4\times$ speedup on average while consuming $4.7\times$ more power, meaning the single TX1 node 60% more energy efficient. The main reason for the performance difference between these two systems is the slower clock speed of the TX1 GPGPU and the $4\times$ higher CUDA core counts of the GTX960 GPGPU.

Although we chose the largest problem size available for benchmarks selected from the Rodinia suite; they have shorter runtimes and stress the GPGPU cores less than the other workloads. Thus, the system with the discrete GPU only improves the performance by $2.8\times$, while consuming $4.4\times$ more power compared to a single TX1 node, making the TX1 84% more energy efficient. On the other hand, other benchmarks that can stress the systems more have worse results compared to the benchmarks selected from Rodinia suite on average. While these benchmarks show a $4.5\times$ speedup on the discrete GPGPU, a single TX1 node still can improve the energy efficiency by 16%.

Cluster Results: Figure 5 shows the runtime and energy consumption of ClusterSoCBench scientific workloads for different ScaleSoC cluster sizes, normalized to the results obtained using two GTX cards. A cluster size of eight nodes has the same number of CUDA cores and a cluster size of 16 nodes has double the number of CUDA cores but consumes the same amount of power. We profiled the time of GPGPU kernels on GTX cards using *nvprof*. Comparing the total time of benchmarks to the time spent on the GPGPU, we observed that, on average, 92% of the time is

³Shared between CPU and GPGPU cores.

Table 6. Runtime, Power, and Energy Consumption of GPGPU-Accelerated Scientific Workloads on a Single TX1 Node Normalized to a Single Discrete GPGPU Card

benchmark	normalized runtime($\times$)	normalized power($\times$)	normalized energy($\times$)
heartwall	2.81	0.21	0.60
leukocyte	2.73	0.25	0.65
lud	1.39	0.18	0.26
needle	1.67	0.26	0.41
srad v1	3.46	0.21	0.71
srad v2	3.94	0.21	0.84
sc omp	2.91	0.24	0.72
lavaMD	3.77	0.25	0.92
hpl	6.11	0.27	1.69
jacobi	3.17	0.20	0.63
cloverleaf	5.18	0.20	1.05
Tealeaf2d	3.04	0.19	0.59
Tealeaf3d	5.04	0.18	0.92

Fig. 5. Runtime and energy consumption of ClusterSoCBench scientific workloads running on 8- and 16-node ScaleSoC clusters, normalized to two discrete GPGPUs.

spent on GPGPU computation. As a single Xeon core is stronger than a single TX1 core, any performance difference must be related to the GPGPU cores. Figure 5 shows that there are three major classes of workloads when characterizing the energy consumption and performance trade-offs of scale-up and scale-out systems.

- When comparing the performance of two GTX cards with eight nodes of the ScaleSoC cluster for the same number of CUDA cores, all workloads consume less energy but deliver lower performance. As GPGPU cores on ScaleSoC run at a lower frequency, these results are expected.

Fig. 6. ScaleSoC cluster throughput, memory, and GPGPU utilization of ScaleSoC for Caffe and TensorFlow. Results are normalized with respect to Tensorflow's values.

Fig. 7. Normalized throughput and unhalted CPU cycles per second of image inference using distributed Caffe for different scale-out cluster sizes, normalized to the discrete GPGPUs.

- Workloads, such as *tealeaf2d*, *tealeaf3d*, and *cloverleaf*, consume more energy and deliver worse performance when a large number of nodes are used. Using more nodes does not necessarily improve performance, yet energy consumption still increases as nodes are added.
- Workloads, such as *hpl* and *jacobi*, improve both performance and energy efficiency when running on the full 16-node ScaleSoC cluster. This is because for the same power budget, the higher degrees of parallelism of these workloads exploit the larger number of nodes to reduce the runtime and improve the energy efficiency.

4.2.2 The Emerging Deep Neural Network Workloads. For image inference using deep neural networks, we use pre-trained AlexNet and GoogleNet models for the ImageNet dataset. We first compare the performance of popular frameworks, Caffe and Tensorflow. Memory and GPU utilization are measured using the *dram_utilization* and *achieved_occupancy* metrics of *nvprof*. Throughput is measured by number of images classified per unit of time. As Figure 6 shows, the ScaleSoC cluster performs better when Caffe framework is used compared to Tensorflow for the same model and image data. This is due to the higher memory requirement of the Tensorflow framework compared to Caffe, which results in lower GPGPU utilization. Therefore, we choose to use the Caffe framework for comparing our ScaleSoC cluster against discrete GPGPUs, as it is more suitable given the limited memory available on our boards.

Figure 7 shows the comparison of image inference on the ScaleSoC cluster and discrete GPGPUs using different class sizes using the Caffe framework. The ScaleSoC cluster improves both the performance and energy efficiency using the *AlexNet* and *GoogLeNet* models. Performance is im-

proved by 1.4 $\times$ and 2 $\times$ and energy efficiency improved by 1.5 $\times$ and 2 $\times$ for *AlexNet* and *GoogleNet*, respectively, when running on the 16-node ScaleSoC cluster. We notice the massive improvement for this type of workload is due to higher CPU utilization for these two emerging AI applications compared to the other benchmarks from scientific domains. For image inference, the CPU must decode JPEG images to prepare raw data for the forward path computation of deep neural networks done by the GPGPUs. The JPEG decompression has control-dominant characteristics that makes it more suitable to run on CPU rather than the GPGPU SMs. Figure 7 shows that the *AlexNet* and *GoogleNet* speedup and unhalted CPU cycles per second of the different ScaleSoC cluster sizes normalized to the discrete GPGPU system. Figure 7 shows even for the same GPGPU SM counts (eight nodes), *GoogleNet* can leverage 64% more CPU cycles per second due to larger core count per SM of the ScaleSoC cluster. The ScaleSoC cluster has a better CPU-GPGPU balance, which benefits this emerging type of applications.

Conclusion 3: Our results show that the GPGPUs available on the ARM SoCs have comparable performance compared to the traditional discrete GPGPUs for the same power budget. There exists a class of workloads that have enough parallelism to exploit the additional number of processors offered by using low-power components given the same power budget.

Conclusion 4: Compared to discrete GPGPUs, our results show that the better CPU-GPU balance of the scale-out solution achieves better performance and energy efficiency for image classification workloads using deep neural networks. This is because the JPEG decompression must be done on the CPU for the forward pass of the neural networks. Comparing the results of *GoogleNet* network running on Caffe, eight nodes of ScaleSoC cluster have the same GPU SM count as two discrete GPGPUs, consume half the power and 74% less energy, and can leverage 64% more CPU cycles per second, which results to improve the image classification throughput by 27%.

4.2.3 The Latency-Sensitive Transactional CPU Workloads. Latency-sensitive transactional workloads are associated with Service Level Objectives (SLOs) that define the requirement of response time. As modern online services have a complex multi-layered architecture, SLOs are defined on the tail of response time distribution rather than as the average response time for each micro-service. The goal is to maximize the server throughput given the constraint on the tail latency of response time. We choose two representative applications to evaluate the performance of ScaleSoC and the Cavium server for transactional workloads: (1) *memcached* that is an in-memory key-value store app and (2) *web serving* using the Apache web server.

Although both *web-serving* and *memcached* are transactional workloads, they have distinct characteristics that represent different aspects of latency-sensitive workloads [25]. The major differences between the two are that *web-serving* has CPU-bound requests, which are stateless and have median response time in order of milliseconds mostly spent in user-space, while *memcached* requests are stateful and are responded to with a short-living thread that has a median latency of microseconds, mostly spent in kernel-space.

We define the SLO as the 99.9 percentile of response time and choose a 200 ms budget for *web-serving* and a 5ms budget for *memcached* [25]. Clients for both *memcached* and *web-serving* both use exponential distribution for inter-arrival time distribution of requests. To stress *memcached*, 80% of requests are gets and the rest are sets. The average get size recorded was between 700 to 800 bytes. The rest of the parameters for the *memcached* client remained the same as the original client in Cloudsuite [17]. We configured the *memcached* server on both the ScaleSoC cluster and the Cavium server to use 48GB of main memory and warmed them up sufficiently. Both systems have an average cache hit ratio of 85% throughout our experiments.

Table 7 shows the maximum average throughput of both systems given the SLO constraint. For *web-serving*, the Cavium Server has 45% higher throughput compared to the ScaleSoC cluster.

Table 7. *Web Serving* and *Memcached* Throughput for ScaleSoC Cluster Compared to the Cavium Server Given the Defined SLO Constraint

workload	SLO	ScaleSoC throughput	Cavium Server throughput
Memcached	99th percentile latency <5ms	44,000 RPS	38,000 RPS
Web Server	99th percentile latency <200ms	960 RPS	1,400 RPS

Results for *web-serving* are expected as *web-serving* requests are CPU-bound and the Cavium server has higher CPU and DRAM frequencies and a larger number of CPU cores than the ScaleSoC cluster, as shown in Table 4.

To run *memcached* on the Cavium server, we used all the known techniques to improve the performance of this benchmark on many-core machines, such as changing the Linux real time scheduler to FIFO, increasing the priority of the *memcached*, separating *memcached* process and interrupt processing core affinities, and considering NUMA zones [25]. Although these techniques improved the performance of the Cavium server substantially (about 73%), ScaleSoC still improves the *memcached* throughput by 15% compared to the Cavium server despite having lower CPU and DRAM frequencies and lower core counts. To make sure the NIC is not the bottleneck for the Cavium server, we repeated our test with multiple clients and using separate NICs. *Memcached* is known to stress the front-end components of CPU cores [10, 26]. Thus, we look at the front-end performance counters and observe 7× and 3× more branch miss prediction and L1 instruction cache misses, respectively, for the Cavium server compared to the ScaleSoC cluster.

Our results show that for the latency-sensitive transactional workloads which are CPU-bound, the Cavium server performs better due to better specs. For workloads, such as *memcached*, where an immense number of short-lived requests must be served simultaneously, shared core resources such as the branch predictor and L1 instruction cache pollution become the performance bottleneck of many-core scale-up architectures such as the Cavium server.

4.2.4 The Classical MPI-Based Scientific CPU Workloads. We use the NPB benchmark suite to analyze the performance of ScaleSoC and the Cavium server for classical scientific workloads. 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 introduces a large amount of contention between threads. To provide a meaningful discussion, we run *ep*, *bt*, and *sp* with 64 MPI processes as well to compare the performance of the same number of cores on both scale-out and scale-up ARM SoCs.

Running a multi-threaded process on a 96-core machine introduces new challenges. Task affinity of the parent process is usually fixed 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 this technique improves the average runtime by 1.6× and the average standard deviation of the runtime reduces from 29.3 seconds to 1.38 seconds across 10 different runs. We also used OpenMP to leverage the multi-core architecture of the the Cavium server. Compared to our OpenMP results on 96 cores, we observed that MPI improves performance by 13%, on average. Similar results are reported for NPB in previous studies [28]. We used MPI for both scale-up and scale-out systems to leverage the multi-core architecture, as it achieves better performance while the software configuration remains the same for both systems.

Table 8 gives the runtime, power, and energy consumption of running workload size class C on the Cavium server, normalized to the ScaleSoC cluster. We choose to run class C because it is

Table 8. Traditional Scientific Applications Result for the Cavium Server Compared to the ScaleSoC Cluster with Class Size C

benchmark	normalized runtime (×)	normalized power (×)	normalized energy (×)
bt	1.30	0.99	1.29
cg	0.94	0.93	0.87
ep	1.04	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

the largest problem size that all benchmarks can be run on both systems. Results show a broad range of performance for both systems. As an example, *ft*'s runtime decreases by 23%, while *mg*'s runtime is increased by 2.5× when running on Cavium server compared to the ScaleSoC cluster. The results are surprising as the ScaleSoC cluster substitutes the internal memory traffic of the Cavium SoC with network traffic, which has higher overhead. In addition, as shown in Table 4, the Cavium server has a faster CPU and memory speed. To understand the reason for these results, we run different workload sizes (A to D) and collect the performance counters on both systems. *ft*'s results for class D are excluded because we could not run it on the ScaleSoC cluster due to memory limitations. 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 [31]. Therefore, we collected 12 counters that are part of ARMv8 PMUv3, and did not collect any additional counters that are only available for specific systems.

After collecting the raw counter values from the systems under the test, we added additional metrics, such as the cache miss ratios and branch miss prediction ratio 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 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 that affect the relative performance of two systems [6]. Since we observe that three principal components explain 95% of the variance of the observation matrix, we use them to calculate the coefficients of regression. The top three performance counters that have the highest coefficient of regression values are then chosen, since they have the largest impact on the model.⁴ Figure 8 shows the relative runtime and value of each of these three chosen events/metrics. To summarize our observations in Figure 8:

- (1) Benchmarks, such as *ft* and *is*, that are network bound perform poorly on ScaleSoC cluster as problem size increases due to high network overhead.
- (2) Benchmarks, such as *bt*, *cg*, *lu*, and *sp*, perform poorly on ScaleSoC cluster at smaller problem sizes but as problem size increases, the poor performance of branch predictor and L2 cache of the Cavium server make them better suited to run on the ScaleSoC cluster.

⁴Using only these 3 variables instead of all 16 only reduces the r-squared value of our regression from 0.93 to 0.9. The Mean Squared Error (MSE) increased from 0.016 to 0.022, which is negligible, and shows that these top three variables are enough to explain the results.

Fig. 8. Relative runtime and events/metrics of the Cavium server compared to the ScaleSoC cluster chosen using PLS.

- (3) Other benchmarks, such as *ep* and *mg*, always perform poorly on the Cavium server because of poor CPU design choices for Cavium SoC. *Ep* has the highest L2 miss ratio and *mg* has the highest speculatively executed instructions ratio of all benchmarks.

Analyzing PLS chosen performance counters gives us the architectural conclusion that the branch predictor and L2 cache of the Cavium server are the bottleneck of performance compared to the TX1 nodes in ScaleSoC cluster. The higher L2 miss ratio is the result of the Cavium ThunderX having less L2 cache data per core and the fact that all cores are connected to the same L2 cache via the Cavium Coherent Processor Interconnect (CCPI) which keep both sockets coherent. Based on the previous Cavium designs (Octeon III), it is estimated that the chip has a short pipeline length to avoid large branch misprediction penalties [4]. However, our results show that the Gshare branch predictor used in the Cavium server has a higher miss ratio than the simple two-level branch predictor of the Cortex A57. Poor branch predictor performance results in a high number of speculatively executed instructions, depending on where the miss prediction gets resolved. Executing a high number of speculatively instructions consumes CPU cycles and reduces performance. We observe that the branch predictor is the bottleneck for the Cavium server when workloads stress the frond-end units (such as *memcached*). However, to confirm the L2 cache is the bottleneck, we perform the following experiment.

We run different class sizes of NPB with 32 instances (36 for *bt* and *sp*) and bind them all on one socket for one set of tests and later divided equally between two sockets of Cavium server. Note that the same number of logical cores are used in both sets of experiments. We would expect the one-socket experiment to perform better since cache coherency traffic is all local. However, the results show the two-socket configuration performs up to $1.8\times$ better than the one-socket configuration. Figure 9 shows the normalized runtime and L2D cache misses of the one-socket experiments with respect to the two-socket experiments for various classes of the NPB benchmark suite. The correlation coefficient for the presented data in Figure 9 is recorded to be 0.71. The two-socket configuration provides more L2D cache per core, reduces the number of cache misses, and improves performance.

Conclusion 5: Our results show that despite the industrial trends to use scale-up many-core architecture for designing ARM SoCs, their overall performance suffers from the poor performance of architectural resources, such as last-level cache, to the extent that they perform worse than scale-out solutions that have network overhead and run at a lower frequency. In our results, we observed that in the case of stressing the front-end component of processors using *memcached*, the scale-up solution has $7\times$ and $3\times$ more misses for branch predictor and L1I cache, respectively,

Fig. 9. Normalized runtime and L2D cache misses of the Cavium server when using only one of the two sockets, compared to using both sockets for running different class sizes of the NPB benchmark suite. In both sets of experiments, the number of MPI processes is the same. The only difference is the scheduling of processes to only one and then two sockets of the Cavium server.

Table 9. The Throughput and Energy Efficiency of Our Cluster and Existing Solutions

	CPU only		GPGPU accelerated	
	Cavium server	ScaleSoC cluster	Two GTX	ScaleSoC cluster
throughput (GFLOPS)	82.57	149.10	95.96	97.47
efficiency (MFLOPS/W)	225.94	378.27	355.41	351.74

For the Cavium server, all 96 cores are used to get the results.

compared to our scale-out cluster, ScaleSoC. We also observed 6.8 $\times$, 1.3 $\times$, and 1.56 $\times$ more branch miss prediction misses, speculatively executed instructions, and last-level data cache misses, respectively, on average, for the scale-up solution running classical HPC benchmarks.

To summarize our comparison and recommendation for future ARM-based clusters, our results show that adding faster network connectivity is critical. ARM SoCs are now capable of taking advantage of the higher available bandwidth and only continue to become more performant. Furthermore, with frameworks, such as CUDA, that simplify programming, GPGPU acceleration becomes a more promising direction to increase the performance and energy efficiency of ARM-based clusters. Table 9 summarizes our cluster’s performance and energy efficiency results for the standard *hpl* benchmark and compares every component of our cluster to the existing solutions alone. Table 9 demonstrates that a large number of ARM cores on a single chip does not necessarily guarantee better performance. Adding the GPGPUs to the makeup of the SoCs enables another direction for ARM computing. We quantified comparable results for the same family and power budget with traditional discrete GPGPUs.

5 RELATED WORK

A number of studies recently appeared that focus on the use of low-power, mobile-class ARM SoCs in the HPC domain [23, 36–39]. Rajovic et al. designed an HPC cluster, Tibidado, with 128

nodes, where each node is based on a mobile 32-bit Nvidia Tegra2 SoC featuring dual Cortex-A9 cores [36, 38, 39]. The study points to several limitations (e.g., lack of ECC protection and high network bandwidth) that arise from using mobile platforms. Mont-Blanc is the latest prototype that uses mobile-class ARM SoCs [37]. Mont-Blanc is based on Cortex A15 (ARMV7) and uses 1GbE for network communication. Unlike with Tibidabo, where its integrated 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 server-class ARM SoCs, a recent study compares the performance and power of the X-Gene 1 SoC against the standard Intel Xeon and the recent Intel Phi [7]. The result concludes that these systems present different tradeoffs that do not dominate each other, and that the X-Gene 1 provides a higher energy efficiency, measured in performance/watt. Azimi et al. evaluated the performance and energy efficiency of X-Gene 1 SoC and x86 Atom for scale-out and high-performance computing benchmarks [10]. They discussed the impact of the SoC architecture, memory hierarchy, and system design on the performance and energy efficiency outcomes.

Data center applications typically have lower communication needs between its threads, which enables them to scale gracefully on parallel clusters. Ou et al. analyzed the energy efficiency of three data center applications (web server, in-memory database and video transcoding) and concluded that the ARM cluster is between 1.2 and 9.5 times more energy efficient than an x86 workstation that uses an Intel Core-Q9400 processor [33]. For I/O-dominated workloads, the FAWN cluster couples lightweight Atom x86 processors in a well-balanced system with a solid-state HDD and 100Mbps Ethernet [8]. Compared to traditional disk-based clusters, FAWN achieves two orders of magnitude of improvements in queries per joule. Attempts to replicate the same success with complex database workloads have led to poor results compared to traditional high-end x86 servers [24].

As for the x86 versus ARM debate, [9, 11, 19], Blem et al. compare 32/64-bit x86 against 32-bit ARMv7 SoC-based platforms using several workloads that are representative of mobile, desktop and server domains [11]. The analysis mostly focuses on the SPEC CPU06 benchmarks, with additional results from two server applications (a web search and a web server). By analyzing the number of instructions and instruction mix and their impact on performance and power, the comparison concludes that ISA effects are indistinguishable and that it is the better branch predictor and larger caches that give x86 processors a lead in performance over ARM processors. The study also concludes that the ARM and x86 systems are engineered for different runtime and power consumption trade-offs. Jundt et al. compared the x86 versus XGene 1 and used hardware performance counters to find the bottleneck of performance [21]. However, using this methodology across different architectures neglects the fact that there is no guarantee an event advertised as the same generic event in fact really measures the same phenomena on different architectures [31].

To the best of our knowledge, none of the previous works considered both the scale-out and scale-up architectures to analyze the performance and energy efficiency of ARM SoC clusters. Furthermore, this is the first work that characterizes a broad range of server-class workloads including both CPU and GPGPU-accelerated workloads for ARM SoC clusters in depth.

6 CONCLUSIONS

In this article, we investigated the performance and energy efficiency of different scaling solutions for ARM SoCs. Compared to previous scale-out ARM clusters, ScaleSoC cluster leverages faster network connectivity and GPGPU acceleration to improve its performance and energy efficiency. We considered a broad range of workloads, including latency-sensitive transactional CPU workloads, MPI-based CPU and GPGPU-accelerated scientific workloads, and emerging deep neural networks that represent modern server-class workloads. We compared the performance

of ScaleSoC cluster with server-class ARM SoCs and discrete GPGPUs using a broad range of server-class workloads.

For the standard *hpl* benchmark, our results showed that faster network can improve the performance and energy efficiency by 2× and 1.4×, respectively. GPGPU acceleration can further improve the throughput and energy efficiency by 40 percent compared to the best performance of using the CPU and GPGPU alone. This is the first work that investigated the performance of GPGPU cores available on the mobile-class ARM SoCs in depth. We showed the available GPGPU SMs on mobile-class ARM SoCs opens a new direction for ARM computing and delivers comparable results with discrete GPGPUs for the same architecture and power budget. We showed that image inference applications using emerging deep neural networks can leverage the better CPU-GPGPU balance of the ScaleSoC cluster, leading to better performance and energy efficiency compared to discrete GPGPUs. Finally, we showed that the scale-up ARM SoCs suffer from shared resource contentions, which limits their performance.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their constructive comments.

REFERENCES

- [1] Reza Azimi. 2017. 10G Ethernet for Jetson TX1 using PCI-e x4. Retrieved May 25, 2017 from <https://devtalk.nvidia.com/default/topic/965204/jetson-tx1/10g-ethernet-for-jetson-tx1-using-pci-e-x4/>.
- [2] Andrei Frumusanu. 2017. ARM A53/A57/T760 Investigated—Samsung Galaxy Note 4 Exynos Review. Retrieved May 25, 2017 from <https://www.anandtech.com/show/8718/the-samsung-galaxy-note-4-exynos-review/5>.
- [3] Reza Azimi. 2017. ClusterSoCBench: set of benchmarks to stress the cluster of GPGPUs. Retrieved May 25, 2018 <https://github.com/scale-lab/ClusterSoCBench>.
- [4] Johan De Gelas. 2017. Investigating Cavium’s ThunderX: The First ARM Server SoC With Ambition. Retrieved May 25, 2017 from <http://www.anandtech.com/show/10353/investigating-cavium-thunderx-48-arm-cores/7>.
- [5] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*, Vol. 16. 265–283.
- [6] Hervé Abdi. 2003. Partial least square regression (PLS regression). In *Encyclopedia for Research Methods for the Social Sciences*. Sage, 792–795.
- [7] David Abdurachmanov, Brian Bockelman, Peter Elmer, Giulio Eulisse, Robert Knight, and Shahzad Muzaffar. 2015. Heterogeneous high throughput scientific computing with APM X-Gene and Intel Xeon Phi. In *Journal of Physics: Conference Series*, Vol. 608. IOP Publishing, 012033.
- [8] David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. 2009. FAWN: A fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. ACM, 1–14.
- [9] Rafael Vidal Aroca and Luiz Marcos Garcia Gonçalves. 2012. Towards green data centers: A comparison of x86 and ARM architectures power efficiency. *Journal of Parallel and Distributed Computing* 72, 12 (2012), 1770–1780.
- [10] Reza Azimi, Xin Zhan, and Sherief Reda. 2015. How good are low-power 64-bit SoCs for server-class workloads? In *Proceedings of the , 2015 IEEE International Symposium on Workload Characterization (IISWC’15)*. IEEE, 116–117.
- [11] Emily Blem, Jaikrishnan Menon, and Karthikeyan Sankaralingam. 2013. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA’13)*. IEEE, 1–12.
- [12] Barcelona Supercomputing Center. 2017. BSC tool. Retrieved May 25, 2017 from <https://tools.bsc.es/>.
- [13] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the . IEEE International Symposium on Workload Characterization, 2009. (IISWC’09)*. IEEE, 44–54.
- [14] Ben Cumming. 2016. CUDA-stream. Retrieved May 25, 2017 from <https://github.com/bcumming/cuda-stream>.
- [15] Kapil Dev and Sherief Reda. 2016. Scheduling challenges and opportunities in integrated CPU+GPU processors. In *Proceedings of the 14th ACM/IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia’16)*. 78–83. DOI : <http://dx.doi.org/10.1145/2993452.2994307>
- [16] Lars Eilebrecht. 1998. *Apache Web-Server: Installation & Modulbeschreibungen, Konfiguration & Administration, Sicherheitsaspekte, Apache-SSL;[für Apache 1.2 und 1.3]*. Internat. Thomson Publ.

- [17] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafae, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *ACM SIGPLAN Notices*, Vol. 47. ACM, 37–48.
- [18] Brad Fitzpatrick. 2004. Distributed caching with memcached. *Linux Journal* 2004, 124 (2004), 5.
- [19] Mateusz Jarus, Sébastien Varrette, Ariel Oleksiak, and Pascal Bouvry. 2013. Performance evaluation and energy efficiency of high-density HPC platforms based on Intel, AMD and ARM processors. In *European Conference on Energy Efficiency in Large Scale Distributed Systems*. Springer, 182–200.
- [20] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*. ACM, 675–678.
- [21] Adam Jundt, Allyson Cauble-Chantrenne, Ananta Tiwari, Joshua Peraza, Michael A. Laurenzano, and Laura Carrington. 2015. Compute bottlenecks on the new 64-bit ARM. In *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing*. ACM, 6.
- [22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.
- [23] Zdravko Krpić, Goran Horvat, Drago Žagar, and Goran Martinović. 2014. Towards an energy efficient SoC computing cluster. In *Proceedings of the 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO'14)*. IEEE, 178–182.
- [24] Willis Lang, Jignesh M. Patel, and Srinath Shankar. 2010. Wimpy node clusters: What about non-wimpy workloads?. In *Proceedings of the 6th International Workshop on Data Management on New Hardware*. ACM, 47–55.
- [25] Jialin Li, Naveen Kr Sharma, Dan R. K. Ports, and Steven D. Gribble. 2014. Tales of the tail: Hardware, os, and application-level sources of tail latency. In *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 1–14.
- [26] Kevin Lim, David Meisner, Ali G. Saidi, Parthasarathy Ranganathan, and Thomas F. Wenisch. 2013. Thin servers with smart pipes: Designing SoC accelerators for memcached. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 36–47.
- [27] Piotr R. Luszczek, David H. Bailey, Jack J. Dongarra, Jeremy Kepner, Robert F. Lucas, Rolf Rabenseifner, and Daisuke Takahashi. 2006. The HPC challenge (HPCC) benchmark suite. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. Citeseer, 213.
- [28] Damián A. Mallón, Guillermo L. Taboada, Carlos Teijeiro, Juan Tourino, Basilio B. Fraguera, Andrés Gómez, Ramón Doallo, and J. Carlos Mourino. 2009. Performance evaluation of MPI, UPC and OpenMP on multicore architectures. In *European Parallel Virtual Machine/Message Passing Interface Users Group Meeting*. Springer, 174–184.
- [29] Jahanzeb Maqbool, Sangyoon Oh, and Geoffrey C. Fox. 2015. Evaluating ARM HPC clusters for scientific workloads. *Concurrency and Computation: Practice and Experience* 27, 17 (2015), 5390–5410.
- [30] Patrick R. Michaud. 2017. PmWiki. Retrieved May 25, 2017 from www.pmwiki.org.
- [31] Andrzej Nowak, David Levinthal, and Willy Zwaenepoel. 2015. Hierarchical cycle accounting: A new method for application performance tuning. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'15)*. IEEE, 112–123.
- [32] Nvidia. 2016. High performance linpack for CUDA. Retrieved May 25, 2017 from <https://developer.nvidia.com/rdp/assets/cuda-accelerated-linpack-linux64>.
- [33] Zhonghong Ou, Bo Pang, Yang Deng, Jukka K. Nurminen, Antti Ylä-Jääski, and Pan Hui. 2012. Energy-and cost-efficiency analysis of arm-based clusters. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'12)*. IEEE, 115–123.
- [34] E. Padoin, D. De Olivera, Pedro Velho, P. Navaux, Brice Videau, Augustin Degomme, and Jean-Francois Mehaut. 2013. Scalability and energy efficiency of HPC cluster with ARM MPSoc. In *Proceedings of the 11th Workshop on Parallel and Distributed Processing*. IEEE, 1–4.
- [35] Parallel forall. 2016. Jacobi for CUDA. Retrieved May 25, 2017 from <https://github.com/parallel-forall/code-samples/tree/master/posts/cuda-aware-mpi-example>.
- [36] Nikola Rajovic, Paul M. Carpenter, Isaac Gelado, Nikola Puzovic, Alex Ramirez, and Mateo Valero. 2013. 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, 40.
- [37] Nikola Rajovic, Alejandro Rico, Filippo Mantovani, Daniel Ruiz, Josep Oriol Vilarrubi, Constantino Gomez, Luna Backes, Diego Nieto, Harald Servat, Xavier Martorell, et al. 2016. The Mont-Blanc prototype: An alternative approach for hpc systems. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 444–455.
- [38] Nikola Rajovic, Alejandro Rico, Nikola Puzovic, Chris Adeniyi-Jones, and Alex Ramirez. 2014. Tibidabo: Making the case for an ARM-based HPC system. *Future Generation Computer Systems* 36 (2014), 322–334.

- [39] Nikola Rajovic, Lluís Vilanova, Carlos Villavieja, Nikola Puzovic, and Alex Ramirez. 2013. The low power architecture approach towards exascale computing. *Journal of Computational Science* 4, 6 (2013), 439–443.
- [40] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–9.
- [41] UK-MAC. 2016. Cloverleaf for CUDA. Retrieved May 25, 2017 from https://github.com/UK-MAC/CloverLeaf_CUDA.
- [42] UK-MAC. 2016. Tealeaf for CUDA. Retrieved May 25, 2017 from <https://github.com/UK-MAC/TeaLeaf>.

Received October 2017; revised May 2018; accepted June 2018