Revisiting the Case of ARM SoCs in High-Performance Computing Clusters

Tyler Fox

Submitted on: May 2, 2017

Submitted in partial fulfillment of the requirements of the degree of Bachelor of Science *with Honors* in Computer Engineering

School of Engineering, Brown University

Prepared under the direction of

Prof. Sherief Reda, Advisor

Prof. Iris Bahar, Reader

By signing below, I attest that the undergraduate thesis listed above meets the criteria for Honors, and has been successfully presented to the faculty at the Undergraduate Research Symposium.

_____

Advisor's Signature

_____

Reader's Signature

BROWN

_____

Honors Chair's Signature

Table of Contents

Abstract

Over the course of the past decade, the explosive popularity of embedded devices such as smartphones and tablets have given rise to ARM SoCs, whose characteristically low power consumption have made them ideal for these types of embedded devices. Recent maturation in the ARM SoC market, which has seen the advent of more powerful 64-bit ARM SoCs, has enabled the development of server-class machines that make use of ARM CPUs. These servers typically use several heavy-duty ARM CPU cores opposed to the typical heterogeneous setup of mobile-class SoCs, which typically integrate low-power CPU cores accompanied by GPU cores and hardware accelerators. In this paper, we aim to characterize high-performance computing (HPC) clusters built using mobile-class ARM SoCs and determine whether they offer performance and efficiency benefits when compared to dedicated ARM servers and traditional x86 servers. For this thesis, we developed a custom HPC cluster of mobile-class NVIDIA Jetson TX1 developer boards with 10Gb network connectivity. We measure and analyze the performance, power consumption, and energy efficiency of the different setups using a variety of HPC and big data applications, ranging from classical numerical benchmarks to emerging deep learning applications. We also evaluate the benefits of using 10Gb network connectivity between TX1 nodes over the more traditional 1Gb connectivity. We show which HPC applications are best-suited for evaluation using these types of clusters and give insights on how to best utilize the ARM architecture when creating clusters for a variety of different applications; namely, good cache design and efficiently using silicon area by including a GPGPU opposed to additional CPU cores are key to creating efficient ARM systems. Finally, we explore the differences between our cluster's distributed GPUs and a traditional system with a centralized, discrete GPU to determine which types of workloads are best-suited for the distributed environment.

1. Introduction

The recent ubiquity of 64-bit ARM CPUs has given newfound motivation for companies to develop ARM-based servers intended for HPC clusters or data centers. To date, almost all of these machines have simply followed the trend set by traditional x86 servers: include several higher-performing (compared to mobile cores), CPU cores in a single machine. This trend becomes even more obvious for ARM servers; because a modern ARM core is less powerful than a modern Intel x86 core, ARM servers must incorporate even more cores than traditional servers to have competitive performance. For example, the Applied Micro X-Gene 1 contains 8 ARMv8 CPU cores, the upcoming Applied Micro X-Gene 3 will have 32 cores, and GIGABYTE has released a rack server that contains 2 Cavium ThunderX ARM SoCs, which each contain 48 CPU cores, for a total of 96 CPU cores. In addition to CPU cores, these SoCs also include IP blocks for a memory controller and 10Gb network connectivity.

Mobile-class ARM SoCs differ from the aforementioned server-class ARM SoCs in that they trade the several higher-frequency CPU cores for fewer lower-frequency CPU cores, while adding GPU cores. In typical consumer products like smartphones and tablets, the GPU cores have traditionally been used exclusively for graphics; however, these GPUs are general-purpose GPUs (GPGPUs) that can be programmed for scientific applications and used to quickly perform numerical calculations. These GPGPU cores typically run at a lower frequency than the CPU cores, but greatly outnumber the number of CPU cores on the system, making them ideal for computations that can be run in parallel.

We compared our cluster against these server-class ARM systems to show that a cluster made of heterogeneous SoCs containing CPU and GPU cores is able to achieve better performance and energy efficiency than homogeneous systems for certain types of HPC and machine learning

workloads and that SoCs with more powerful CPU cores can be hampered by poor network-on-chip designs, causing to performance to significantly deteriorate.

In chapters 5.A, 6.A, 7.A.i, and 7.B.i we re-visit the use of ARM SoCs for HPC application in an attempt to determine the optimal type and composition of ARM SoCs for HPC clusters as well as which HPC applications benefit the most from ARM SoCs and can be used to compare various ARM systems. While previous attempts to answer these questions have been made [3], [13], [17], [24], [27], [28], [29], they all exclusively evaluated the CPUs. What's more, some were written before the introduction of 64-bit ARM SoCs, when the ARM SoC market was in its infancy. As the types workloads have evolved, placing an ever-growing emphasis on machine learning in cloud computing, it has become clear that HPC clusters stand to significantly benefit from the addition of GPGPUs. As a result, we have reason to believe that the integration of GPGPUs could present a new opportunity for ARM HPC clusters.

To explore the practicality of this type of cluster, we built our own custom cluster using NVIDIA Jetson TX1 boards, based on the NVIDIA TX1 ARM SoC. The cluster we created differs from previous systems in two key ways: its SoCs use GPGPU cores alongside their ARM CPU cores when applicable, and the nodes are connected with 10Gb network adapters instead of traditional 1Gb Ethernet. Unlike previous clusters built in the early days of ARM SoCs, modern mobile ARM SoCs such as the TX1 are potent enough to take advantage of the bandwidth of 10Gb network. This custom cluster was then compared to both a system comprised of two X-Gene 1 nodes and a more traditional server-class machine containing an Intel Xeon x86 CPU. These three systems were used to evaluate the benefits and shortcomings of the different cluster architectures when operating on a variety of HPC workloads.

We also evaluate the performance and energy efficiency of our cluster compared to a traditional x86 server based on a modern Xeon processor. We demonstrate that while the Xeon server is a more potent machine, there are certain classes of workloads that can take advantage of the distributed environment provided by the heterogeneous cluster. In particular, certain HPC and deep learning applications that can utilize the GPGPU of the mobile-class ARM SoCs tend to perform better on our cluster and, in some cases, are capable of outperforming the more powerful Xeon-based machine in both performance and energy efficiency.

Finally, we assess the GPU performance of our TX1 cluster in the context of typical GPU setups on traditional workstations and clusters. We seek to determine the advantages and disadvantages of distributed GPUs compared to a centralized GPU, testing and analyzing both configurations using multiple workloads that each stress a different part of the system.

The organization of this thesis is as follows. In chapter 2, we review related work. In chapter 3, we describe our experimental infrastructure and the HPC workloads that we used in our experiments. In chapter 4, we discuss the impact of 10Gb network connectivity in a mobile-class cluster. In chapter 5, we analyze the memory hierarchy of mobile-class and server-class ARM systems. In chapter 6, we describe our methodology and results for CPU evaluation of server-class and mobile-class ARM SoCs compared to x86. In chapter 7, we examine the impact of heterogeneity in HPC clusters. In chapter 8, we quantify the impact of our 10Gb network and GPGPU acceleration using the HPL benchmark. Finally, we summarize the main conclusions in chapter 9.

2. Related Work

There have been previous studies that experimented with using systems based on ARM SoCs for high-performance computing applications [3], [13], [17], [24], [26], [72[27], [28], [29]. Many of these studies found that scalability is often a defining metric when determining whether an ARM-based system will perform well for a certain application. Because ARM CPUs are designed with power consumption in mind, a single ARM core offers less performance than an x86 core intended for use in servers. This means that systems made using ARM SoCs will naturally require more cores than a comparable x86 system to achieve equivalent performance. Rajovic *et al.* designed Tibidabo, an HPC cluster made of 128 ARM nodes. Specifically, each node was based on a mobile 32-bit NVIDIA Tegra 2 SoC, powered by a dual Cortex-A9 cores [27], [28], [29]. The Tibidabo studies identified several limitations of using ARM-based nodes for HPC applications at the time, such as their low network bandwidth, restricting communication between nodes, and lack of error-correcting code memory to detect and correct common types of data corruption. The performance of the Tibidabo cluster was evaluated using the HPL Linpack benchmark, which is a common benchmark to compare supercomputers' performance. The cluster was able to achieve an efficiency of 120 MFLOPS/W, which equates to an average of 0.47 MFLOPS/W per core.

Other studies have continued to this space with more breadth. A study conducted by Padoin *et al.* involved testing three multi-core ARM-based systems and comparing their energy efficiency [24]. The three SoCs tested were the Texas Instruments PandaBoard (based on the OMAP 4430 processor), ST-Ericsson Snowball, and NVIDIA Tegra 2, which all make use of dual Cortex-A9 cores. Of the systems tested, the Snowball SoC was able to achieve an energy

efficiency of 206.6 MFLOPS/W, improving on the 120 MFLOPS/W reached in the Tibidabo study.

New studies emerged following the introduction of 64-bit ARM-based platforms. A recent study evaluated the X-Gene 1 SoC, a 64-bit ARM SoC, and compared it to a traditional Intel Xeon server as well as the more recent Intel Phi [3]. The conclusion here was that these three different systems each possess their own unique advantages as well as tradeoffs. No one system completely dominated the other two across the board. Furthermore, the X-Gene 1 was found to be one of the more energy-efficient servers in terms of performance per watt.

Another study compared 32-bit ARMv7 SoCs to traditional 32-bit and 64-bit x86 systems using many different benchmarks representative of workloads across mobile, desktop, and server domains [6]. Their experiments primarily center on the SPEC CPU06 benchmarks, which are a set of CPU-intensive benchmarks that stress a system's processor and memory subsystem, and also include results from a web search and web server test application. They collect the number of instructions and instruction mix and then analyze their impact on performance and power consumption. The study found that the effects of the different instruction set architectures (ISA) do not directly impact efficiency; rather the superior branch prediction and larger cache sizes of x86 systems are the main reason they performed better than their ARM counterparts. They also conclude that ARM and x86 systems are simply designed with different runtime and power consumption trade-offs.

The most important characteristic of datacenter applications is their need to be able to scale for use in large clusters; as such, they typically have low communication needs between threads. Ou *et al.* measure the energy efficiency of three datacenter applications (web server, in-memory database, and video transcoding). They found that the ARM clusters they tested were able to

achieve 1.2x and 9.5x better energy efficiency that a reference x86 workstation based on an Intel Core-Q9400 processor [23]. For workloads dominated by I/O, the FAWN cluster utilizes weaker Intel Atom x86 processors, a solid-state hard drive, and 100Mbps Ethernet [4]. When compared to traditional disk-based clusters, the FAWN cluster performs two orders of magnitude better, when measured in queries per Joule. There have been attempts to replicate this success using complex database workloads, but these results have not shown the same sort of promise for ARM clusters when compared to high-end x86 servers [14].

3. Infrastructure

A. Hardware Setup

To evaluate high-performance computing applications on the different classes of ARM machines, we constructed two 64-bit ARM clusters. The first cluster was comprised of eight NVIDIA TX1 nodes. Each of the eight nodes was outfitted with a solid-state drive connected to its SATA port for storage and a 10Gb network adapter for network connectivity between nodes. The second cluster was made of two Applied Micro X-Gene 1 nodes. Each of these nodes also used a solid-state drive for storage and 10Gb connection for inter-node network connectivity. The detailed specifications of the individual nodes that make up each cluster can be found in Table 1.

	NVIDIA TX1	**APM X-Gene 1**
ISA	64-bit ARM v8 & PTX	64-bit ARM v8
Node Tech	20 nm	40 nm
Cores	4 A57 CPU + 256 GPU	8 CPU
Frequency	1.9 GHz	2.40 GHz
Dispatch Width	3	2/4
L1 (I/D) Size	48KB/32KB	32KB/32KB
L2 Size	2MB	256KB x 4
L3 Size	-	8MB
DRAM	4GB LPDDR4-1600	16GB DDR3-1600
Nominal Power	20W	60W

Table 1: Configurations of Individual Nodes

The X-Gene 1 SoC contains eight ARMv8 cores that each have a maximum frequency of 2.4 GHz [35]. The TX1 SoC has four ARMv8 cores that each have a maximum frequency of 1.9 GHz. The TX1 SoC, however, also includes 256 Maxwell CUDA GPU cores in addition to its aforementioned CPU cores. Applied Micro's X-Gene 1 specifications claim an out-of-order four instruction issue width; however, it can issue a maximum of two integer instructions. The TX1's ARM core, on the other hand, is able to dispatch three instructions per cycle. One of the important differences between the two SoCs is that the X-Gene SoC is manufactured using planar 40nm transistor technology, while the TX1's SoC is manufactured using 20nm transistor technology. This difference in manufacturing technology gives the TX1 SoCs an inherent advantage when it comes to area density and power consumption.

The X-Gene SoC's cache memory subsystem is larger than that of the TX1, as it contains L1, L2, and L3 caches compared to just L1 and L2 caches on the TX1. While the TX1's L2 cache is double the size of that of the X-Gene (2MB vs. 4x256KB), the X-Gene also contains a large 8MB L3 cache, which is nonexistent on the TX1. Each X-Gene node contains 16GB of DDR3 ECC memory at a frequency of 1600MHz. As a mobile-class SoC, the each TX1 uses 4GB of low-power DDR4 (LPDDR4) memory at the same 1600MHz frequency.

We also constructed a reference server that contains an Intel Xeon Haswell CPU with 16GB of memory and a 10Gb network controller for connectivity to compare our cluster of TX1s to a traditional x86 system. This x86 system serves as a reference point for modern performance and efficiency in the server space. Full specifications of the reference x86 Xeon node compared to a TX1 node are shown below in Table 2.

	NVIDIA TX1	Xeon Haswell Server
ISA	64-bit ARM v8 & PTX	x86-64
Lithography	20 nm	22 nm
CPU/GPU Cores	4 A57 CPU + 256 GPU	Xeon E5-2630 v3 (8 Cores) GTX 960 GPU (only in chapter 7)
Frequency	1.9 GHz	2.40 GHz (3.20GHz Turbo)
L1 (I/D) Size	48KB/32KB	256KB/256KB
L2 Size	2MB	2MB
L3 Size	-	20MB
DRAM	4GB LPDDR4-1600	16GB DDR4-1600
Nominal Power	20W	90W

Table 2: Configuration of a TX1 Node compared to our reference x86 Xeon Node

Version 4.9.4 of gcc was used to compile all benchmarks with consistent optimization flags across all systems. The Ubuntu 14.04 operating system was used on the X-Gene and TX1 systems, while Ubuntu 16.04 was used on the x86 Xeon server. CUDA 7.0 was used in all experiments except TensorFlow, which needed CUDA 8.0 to function properly on the TX1 cluster.

B. Cluster organization

Mobile-class development boards such as the Jetson TX1 typically come standard with 1Gb Ethernet ports. While 1Gb network connectivity is more than enough for typical mobile use, this is rarely sufficient for the demands of cluster-based computing. To make our reference cluster's networking setup competitive with traditional clusters, we connected a Startech PEX10000SFP PCIe 10Gb fiber network card to the PCIe x 4 slot on each of the Jetson TX1 nodes. As the X-Gene ships with both 10Gb and 1Gb ports, an additional network adapter was not required. All

10Gb ports were connected to the same Cisco SG350XG-24F 24-port switch with a switching capacity of 480 Gbps. All 1Gb ports were connected to a Netgear 1Gb network switch. Using this described network infrastructure, we constructed two clusters: one made up of eight Jetson TX1 boards (shown in Figure 1), and another made up of two X-Gene 1 nodes.

It is worth noting that slight modifications were needed to make the Startech 10Gb network cards function properly on the TX1. Our initial testing of the 10Gb network cards showed that communication between two TX1 nodes would experience packet loss on average. After some troubleshooting, we observed that modifying the network card driver to disable the use of paged buffers would fix this issue. The patch needed for the driver to alleviate this problem is included in Appendix A. We also experienced an issue where 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's kernel, which is responsible connecting the direct memory access (DMA) mapping-capable input-output bus to the system's main memory without involving the CPU. The patch applied to fix this issue is also included in Appendix A. After applying these small patches to the network driver and TX1 kernel, the 10Gb network cards function properly.

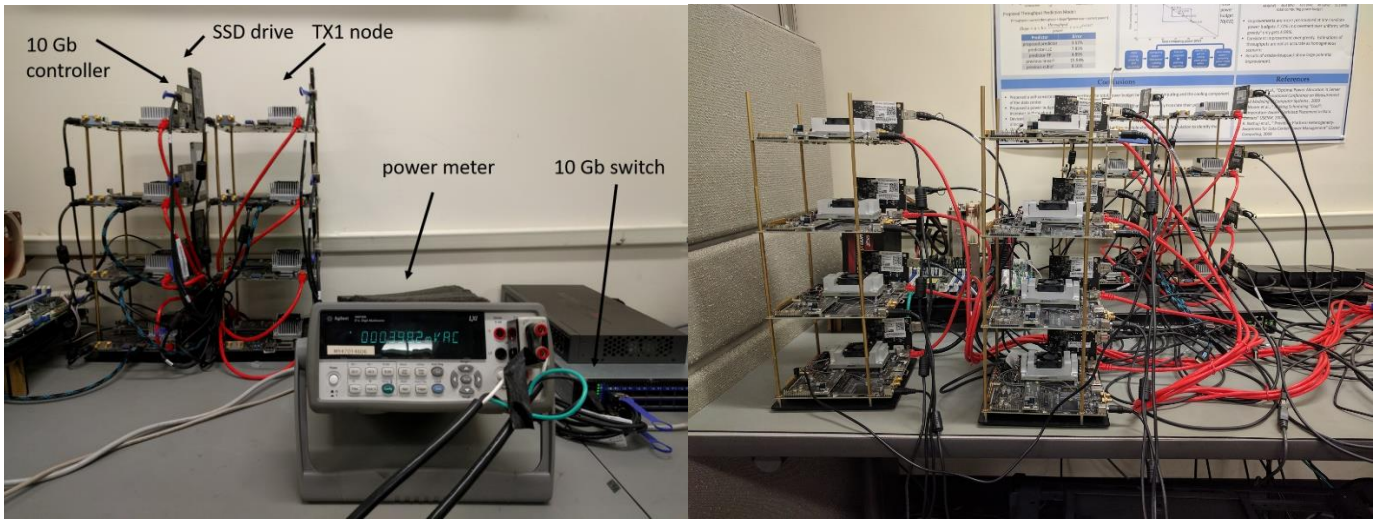

Figure 1: Cluster of 8 TX1 nodes equipped with SSDs and 10Gb network controllers shown with our power meter and 10Gb network switch (left). Full cluster of 16 TX1 nodes (right).

C. Workloads

We compiled a wide range of benchmarks for our experiments that are representative of modern workloads. Several of the workloads we chose utilize the message passing interface (MPI). MPI is a standardized, portable message-passing system used to facilitate communication in parallel computing architectures [9]. The MPI standard dictates both the syntax and semantics of a set of library routines that can be used by writing programs in C, C++, and Fortran that take advantage of the standard. MPI is often used in modern cluster computing to communicate between nodes when operating on parallel workloads. We used an Open MPI version 1.10, an open-source implementation of MPI, in our experiments.

i. CPU Benchmarks

The NAS Parallel Benchmarks (NPB) were used to evaluate cluster performance. NPB are a set of benchmarks intended for measuring the performance of highly parallel computers, often arranged in a cluster. NPB are developed and maintained by the NASA Advanced Supercomputing Division (NAS). We used NPB 3.3 for our testing, which makes use of MPI for

communication between nodes (as of NPB 2.4 in 2002). Our tests included the original eight NPB benchmarks: MultiGrid (MG), Conjugate Gradient (CG), Fast Fourier Transform (FT), Integer Sort (IS), Embarrassingly Parallel (EP), Block Tridiagonal (BT), Lower-Upper Gauss-Seidel Solver (LU) and Scalar Pentadiagonal (SP). The MG benchmark approximates the solution to a three-dimensional discrete Poisson equation. CG uses the 'inverse iteration with the conjugate gradient' method to estimate the smallest eigenvalue of a large sparse symmetric positive-definite matrix. The FT benchmark uses the Fast Fourier Transform (FFT) to solve a 3D partial differential equation. IS uses the bucket sort algorithm to sort small integers. EP uses the Marsaglia polar method to generate independent Gaussian random variates. BT, SP, and LU all solve a synthetic system of nonlinear partial differential equations, but use three different algorithms. The NPB benchmarks are available for CPUs only—there is no CUDA version available to make use of GPGPUs on ARMv8 machines.

ii. Single-Node GPU Benchmarks

To evaluate the single-node effects of incorporating a GPGPU in the TX1, we used the Rodinia benchmark set [31], which can be compiled to run on either the CPU-only or on the CPU+GPU. The Rodinia benchmark set is a collection of benchmarks provided by the University of Virginia that aims to test heterogeneous computer systems that contain accelerators such as GPGPUs. In order to produce the most meaningful comparisons, we chose the Rodinia benchmarks that have long runtimes: Leukocyte, Heart Wall, LU Decomposition, Needleman-Wunsch, SRAD, Streamcluster, and LavaMD. The Leukocyte application detects white blood cells in the first frame of a video, then tracks the cells through subsequent frames. The Heart Wall application tracks the movement of a mouse heard through a sequence of 104 ultrasound images that each have a resolution of 609x590. LU Decomposition is an algorithm used to calculate the solutions for a set

of linear equations using matrices. Needleman-Wunsch is a method for performing DNA sequence alignments using 2D matrices. SRAD, one of the first stages of the Heartwall application, is a diffusion method based on partial differential equations for ultrasonic and radar imaging applications. Streamcluster takes a stream of input points and finds a predetermined number of medians such that each point is assigned the center nearest it. The CUDA version of this benchmark is further optimized to parallelize the *pgain()* function, which is responsible for computing the cost savings of opening a new center. The LavaMD application computes particle potential and relocation within a large, three-dimensional space that is caused by mutual forces between particles.

iii. Multi-Node GPU Benchmarks

We also use deep learning workloads running on Caffe and TensorFlow 1.0 alpha to evaluate the systems, as they represent an increasingly prevalent class of high-performance computing applications. Caffe is an extremely popular open-source deep learning framework from UC Berkeley that is known for its speed and modularity. It supports several different types of neural network designs, and is currently used for a wide variety of applications in both research and industry. Caffe is capable of running on CPUs and also supports GPU acceleration on CUDA-enabled GPUs using NVIDIA's CUDA Deep Neural Network (cuDNN). TensorFlow is a machine learning library that was originally developed by the Google Brain team for internal use within Google. It was later open-sourced, and new versions are still being developed and periodically released. In addition to its ability to run on the CPU or GPU, TensorFlow offers a distributed framework that allows for training models across multiple nodes. TensorFlow's distributed training shares the weight parameters between nodes, allowing tasks to run on multiple machines, utilizing the GPUs on each.

These two deep learning frameworks are used to compare performance of image classification on the mobile, heterogeneous cluster with the server-class cluster, and eventually with a reference x86 server. The images used are from ImageNet, a very large database of images intended for use in visual object recognition research. The set of images is organized according to the WordNet English lexical database [34]. WordNet groups different parts of speech into 'synsets,' which are described as groups of cognitive synonyms. WordNet contains over 100,000 synsets, and ImageNet provides approximately 1,000 images for each synset. The sets images in each synset are annotated by humans, providing a very large, reliable set of images that have associated concepts from the WordNet hierarchy. This makes the ImageNet database ideal for evaluating image classification performance.

For our experiment, two neural network models, AlexNet and GoogleNet, are used to classify a subset of the images from the aforementioned ImageNet database. AlexNet is a convolutional neural network model designed by Alex Krizhevsk y, Ilya Sutskever and Geoff Hintonm, that is capable of labeling, or classifying, images. AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, achieving a competition-best top-5 error rate of 15.3%. GoogleNet is another convolutional neural network intended for image classification. As its name suggests, it was created by Szegedy *et al.* from Google. GoogleNet's primary contribution was its significantly reduced number of parameters in the network (4 million compared to AlexNet's 60 million). These contributions helped GoogleNet win ILSVRC 2014.

TensorFlow is also used to perform training of a convolutional MNIST model and a CIFAR-10 model. MNIST is a data set of handwritten digits often used to train models to recognize these handwritten digits. CIFAR-10 is a dataset of 60,000 32x32 images in that span over 10 different classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. This dataset is used

to train models to recognize these types of images and classify them. As of version 0.8, TensorFlow supports distributed training. This framework allows multiple nodes in a cluster to be used to train a model, with parameters and weights being shared between all nodes.

Chapter 7.C examines different types of workloads capable of running on multiple GPUs to gain an understanding of which types of workloads are better suited for centralized or distributed GPGPU configurations. Five benchmarks, HPL, a Jacobi solver, Cloverleaf, and Tea Leaf 2D, TeaLeaf 3D are run and analyzed on both systems. A CUDA-enabled version of HPL is was used to measure performance and energy efficiency of our centralized and distributed systems. HPL is a freely available, portable implementation of the High-Performance Computing Linpack benchmark written in C [25]. This benchmark solves a random, dense linear system (Ax = B) of order n using 64-bit double precision arithmetic. The Linpack benchmarks is designed to measure a system's floating point computational power and are used to rank the TOP500 list of the world's most powerful supercomputers. NVIDIA has produced a release of HPL that includes CUDA support, allowing for the application to make use of a system's CPU and GPU. We also used a CUDA+MPI implementation of a Jacobi solver to explore potential differences between centralized and distributed GPUs when used for the purpose of accelerating computationally-intensive workloads [21]. This benchmark solves the Poisson equation with a Jacobi solver, applying a 2D domain decomposition to reduce communication between processes so that multiple GPUs across multiple nodes can be more effectively utilized. For our experiment, we ran 1000 iterations on both systems using a total domain size of 16800 x 16800 in each case. In the case of multiple TX1 nodes, the topology was always kept as close to square as possible. CloverLeaf iteratively solves the compressible Euler equations on a Cartesian grid and TeaLeaf solves the linear heat conduction equation in either two (TeaLeaf 2D) or three dimensions (TeaLeaf3D) [16]

[15]. For our experiments, we used a CUDA+MPI implementation of both Cloverleaf, TeaLeaf2D and TeaLeaf3D. On both systems, we ran CloverLeaf for 100 steps, TeaLeaf2D for 10 steps and TeaLeaf3D for 5 steps.

D. Performance and Power Measurements

We equipped all nodes across the various systems with the capability to record a wide range of measurements, including performance counters, network traffic information, and power consumption. We measure each system's power by sensing the external current at the 120V AC outlet with a 10Hz sampling rate. We then take an average of the power consumption data collected while an application is running and report this data point. When evaluating energy data, we consider two key metrics: total energy consumption and overall efficiency, measured in terms of throughput per watt. The units for throughput vary with the application.

4. Impact of Network on Mobile Clusters: 1Gb vs. 10Gb

Previous attempts to build high-performance computing clusters using mobile SoCs have utilized 1Gb network connectivity for communication between nodes for two main reasons: First, early mobile boards were not capable of expansion via a PCIe port, which meant that it was not possible to add an additional network adapter to upgrade the node's network connectivity. Furthermore, the CPU cores of older boards were not potent enough to take advantage of the additional network bandwidth offered by these 10Gb network cards—these early ARM CPUs were simply not capable of driving enough network traffic. Unlike these previous efforts, our cluster made of NVIDIA Jetson TX1 boards feature 10Gb network controllers, making them much more competitive with modern clusters. Several experiments were conducted to demonstrate the benefits of pairing these mobile-class SoCs with 10Gb network adapters.

In the first experiment, we used the iPerf3 and ping utilities to evaluate the raw throughput and latency of both the 1Gb and 10Gb network adapters on the cluster. iPerf3 is a tool used to obtain active measurements of the maximum achievable bandwidth for an IP network. The utility runs on two nodes, where one node acts as the server and the other node acts as a client. For each test, iPerf3 reports the bandwidth, loss, and a host of other parameters. Ping is a network monitoring utility that is typically to check whether there a network connection exists between two computers. Ping measures the round-trip time for messages sent from a host computer to a destination computer that are then relayed back to the source. This feature of Ping makes it a very useful tool for measuring network latency between two nodes. Table 3 shows the average throughput and latency measured using the iPerf3 and ping tools, respectively, for both the 1Gb and 10Gb networks between two TX1 nodes. These results show that the addition of 10Gb network adapters

between two TX1 nodes offers a 6x improvement in throughput and 60% reduction in compared to the standard 1Gb network.

	10Gb	**1Gb**
Throughput (Gb/s)	3.13	0.53
Latency (ms)	0.52	1.30

Table 3: Measured Average Throughput and Latency of 10Gb and 1Gb Interfaces Between Two TX1 Boards

While these raw throughput and latency results are useful in that they serve as an upper bound, it is important to evaluate the effects of the different network connections in the context of applications. To do this, we ran the NPB benchmarks as well as the five GPU benchmarks mentioned in chapters 3.C.i and 3.C.iii on the TX1 nodes, using the 1Gb network and then the 10Gb network for communication between nodes. The speedups provided by using the 10Gb network for the eight NPB benchmarks for two, four, and eight TX1 nodes are shown in Figure 2, and the speedups for the GPU benchmarks using the same configurations of nodes is show in Figure 3. Note that the results for the FT benchmark for two nodes are not included because the combined memory of two TX1 nodes is not large enough to hold the matrix. The general trends shown in the figure indicate that speedups are dictated by two factors: the size of the cluster and the network intensity of a given benchmark. As more nodes are added to the cluster, more inter-node communication occurs as the benchmarks are run. It follows that increased network activity leads to greater speedups when faster network technology is used. Furthermore, benchmarks that are network-intensive, such as CG, FT, and IS on the CPU side and HPL on the CPU + GPU side, show large speedups of 2.7x, 2.6x, 3.3x, and 1.54x respectively, when the 10Gb network is used between eight nodes. On the contrary, the EP benchmark on the CPU side and the Jacobi

benchmark on the GPU side, which both involve very little network communication, do not receive any speedup from using the 10Gb network cards.

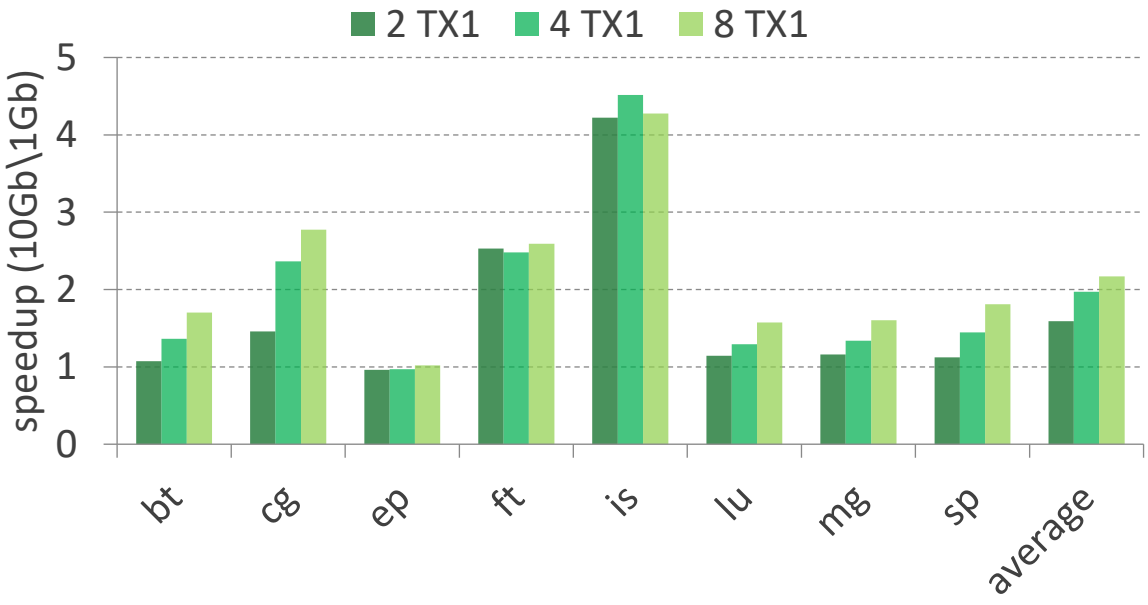

Figure 2: Speedup gained by using 10Gb adapters compared to 1Gb for the NPB Benchmarks

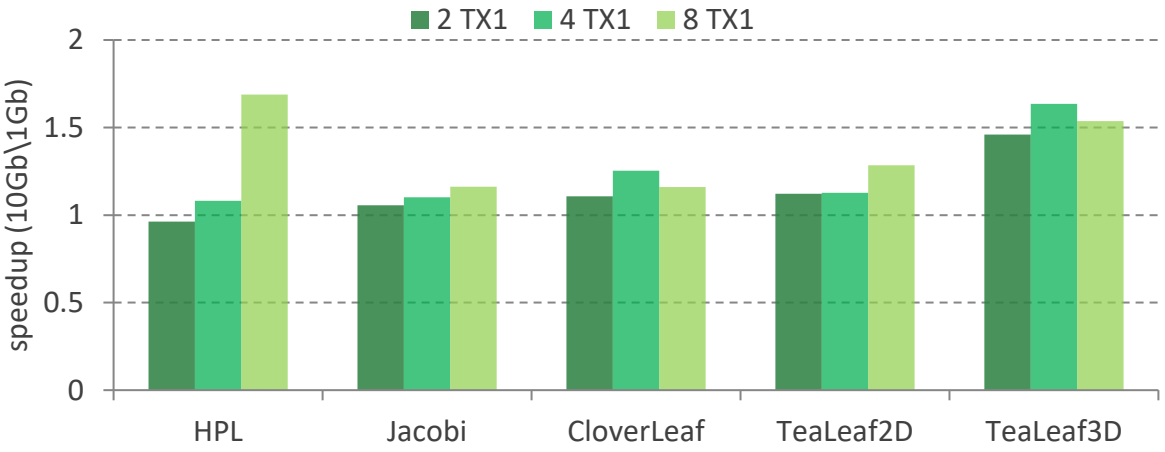

Figure 3: Speedup gained by using 10Gb adapters compared to 1Gb for the GPU benchmarks.

While the 10Gb controllers have shown to offer significant speedups in many cases, they themselves consume additional power. The TX1 board on its own consumes a maximum of

approximately 20W; however, when a 10Gb network adapter is connected, the same board can consume up to 30W. This increase in per-node power consumption when the 10Gb network cards are used translates to an average increase of 56% in power consumption for the cluster of eight nodes. We compared the energy efficiency ratio of the same benchmark sets used to show speedup above. This energy efficiency ratio of using the 10Gb network adapters compared to the standard 1Gb network adapters are shown in Figure 4. For this comparison, power measurements were recorded (in watts) for each of the eight individual NPB benchmarks using the 10Gb and 1Gb networks.

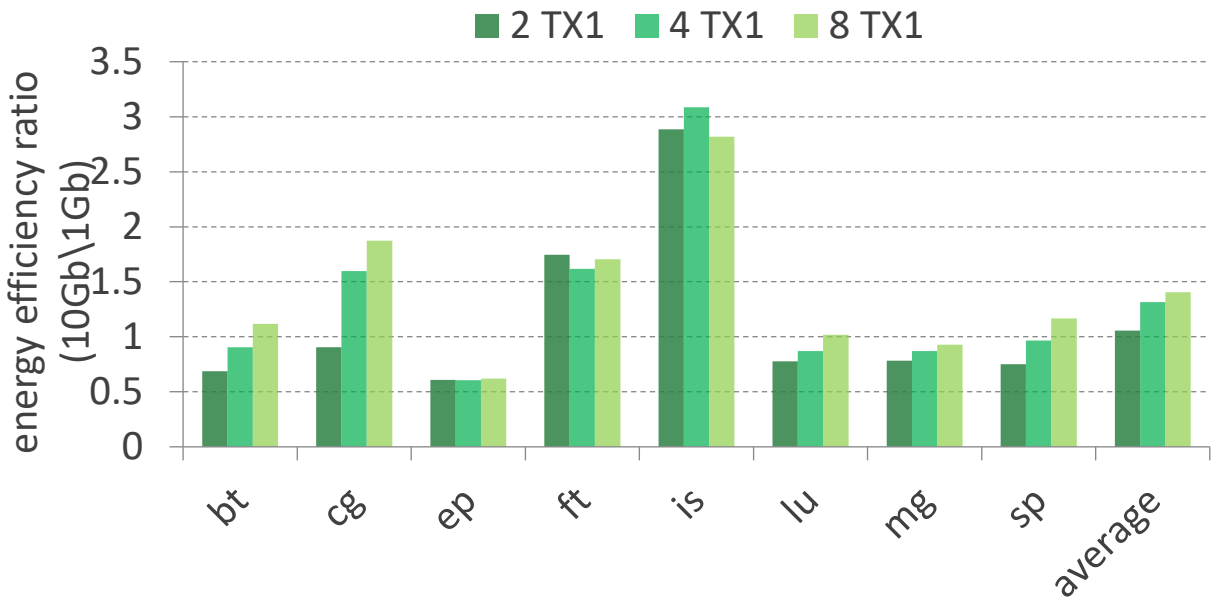

Figure 4: Energy efficiency ratio of using 10 Gb adapters compared to 1Gb for the NPB benchmarks.

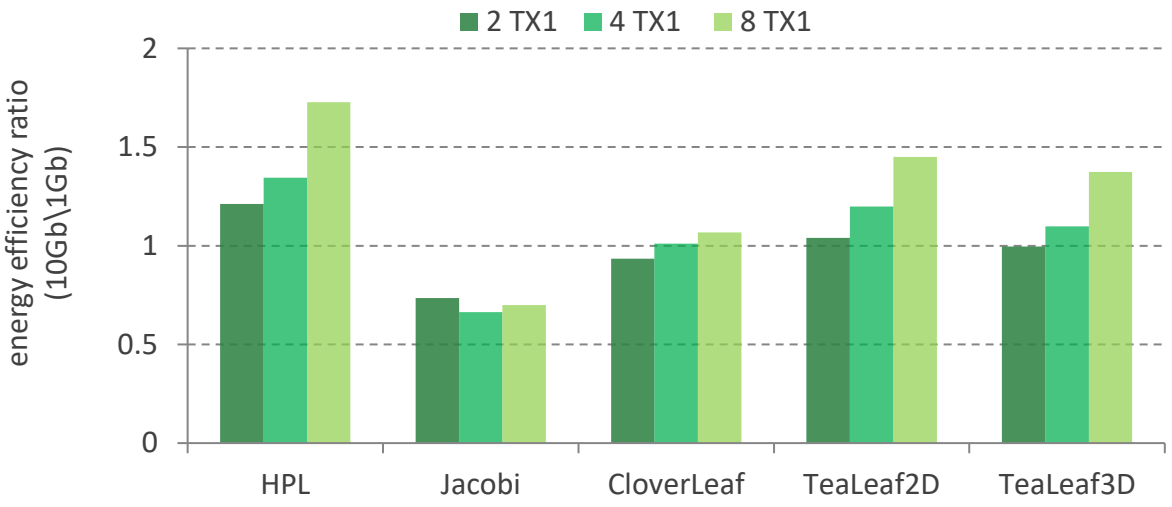

Figure 5: Energy efficiency ratio of using 10 Gb adapters compared to 1Gb for the NPB benchmarks.

Energy efficiency was then calculated in terms of operations per watt. This energy-efficiency ratio exhibits the same trends as those found with the speedups: more network activity, either resulting from an increase in the number of nodes or higher network activity for a given benchmark, results in a higher energy-efficiency ratio. On average, the 10Gb network cards were responsible for a 30% increase in energy-efficiency for the cluster of eight TX1 nodes. While the communication-intensive benchmarks benefited from the presence of the 10Gb adapters, the benchmarks that did not heavily rely on network communication, such as EP, LU, and MG, saw reductions in energy-efficiency when using the faster network adapters. This is easily explained by the fact that the 10Gb adapters' increased power draw translated to little if any increase in performance for the benchmarks that did not heavily utilize the network; in other words, the increase in power consumption was not offset by an increase in performance, leading to a decrease in overall energy-efficiency.

Unlike earlier HPC clusters created using mobile-class SOCs, the 64-bit Jetson TX1 nodes in our cluster are powerful enough to make use of the 10Gb network adapters' additional bandwidth.

The results and analysis above validate the benefits of using 10Gb network controllers instead of standard 1Gb Ethernet connectivity in modern clusters made from mobile SoCs, especially for applications that heavily rely on communication between nodes.

5. Memory Hierarchy Analysis: Methodology and Results

A. Mobile ARM vs. Server ARM

We used LMbench3 memory read latency benchmarks to evaluate the memory hierarchy of the X-Gene and TX1 nodes [18]. LMbench is an open-source suite of portable benchmarks used for comparing the performance of different UNIX systems. The benchmarks are all written in C, and can be compiled with GCC. The memory latency test was used to show the latency of all memory caches present in each system. Figure 6 shows how increasing the array size, and thus testing the different levels of the memory hierarchy, affects the load latency of the TX1 and X-Gene. The latencies for each cache level as well as sequential and random reads are reported in Table 4. X-Gene shows a lower latency than the TX1 for all cache levels. However, the X-Gene has a longer latency for main memory sequential read, which makes sense, as it was intended for server-class workloads with larger caches.

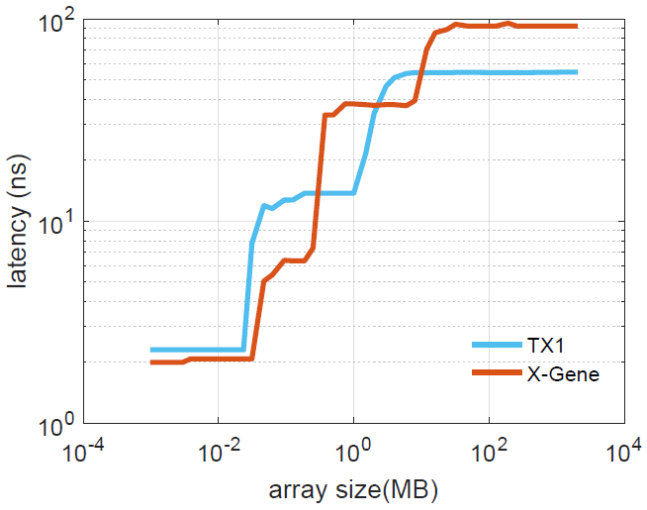

Figure 6: Load latency as a function of array size.

	TX1 (ns)	X-Gene (ns)

L1 Latency	2.31	2.08
L2 Latency	13.71	6.39
L3 Latency	-	37.33
Main memory sequential latency	54.23	92.07
Main memory random latency	254.87	143.51

Table 4: Memory Hierarchy Latency of TX1 and X-Gene

The Sustainable Memory Bandwidth in High Performance Computers (STREAM) and RandomAccess benchmarks from the High-Performance Computing Challenge (HPCC) benchmark suite were used to evaluate memory throughput by measuring CPU and GPU bandwidth. The HPCC Benchmark suite is a combination of several different benchmarks that aim to test several independent attributes of HPC systems. The DARPA High Productivity Computing Systems program, the United States Department of Energy, and the National Science Foundation have all co-sponsored the HPCC project. The STREAM benchmark measures sustainable memory bandwidth in MB/s, and is designed to work with datasets that greatly exceed the available cache on the system in order to separate the measurement of the memory subsystem from the machine's theoretical peak performance. The RandomAccess benchmark measures the peak capability of a machine's memory subsystem by measuring the rate of integer random updates of memory in mega updates per second (MUPS).

Table 5 shows the results from the memory throughput tests for a single node. While the two systems' DRAM both have a frequency of 1600MHz, the TX1 uses newer LPDDR4 RAM, whereas the X-Gene uses older LDDR3 memory. The TX1's GPU improves its stream throughput by 2x on average compared to its four CPU cores; however, random access does not benefit from the GPU. Table 6 shows results from the STREAM and RandomAccess tests for four TX1 and

two X-Gene nodes. These results demonstrate one of the key benefits of cluster computing with a large number of nodes; as the number of nodes in a cluster increases, so does the aggregate bandwidth of the nodes' cores to their main memory. This is clearly shown in Table 6, as the four TX1 nodes outperform the X-Gene nodes in both the STREAM and RandomAccess tests, despite having the same number of cores.

	TX1		**X-Gene**
	4 CPU cores	256 GPGPU Cores	8 CPU Cores
STREAM copy (GB/s)	11.72	20.99	17.7
STREAM scale (GB/s)	11.89	20.91	14.25
STREAM add (GB/s)	9.47	21.19	26.66
STREAM triad (GB/s)	9.32	21.08	19.41
RandomAccess (MUPS)	17.30	17.28	64.41

Table 5: Memory Throughput Using STREAM and RandomAccess Benchmarks from HPCC

	TX1 Cluster (4x4 Cores)	**X-Gene (2x8 Cores)**
STREAM copy (GB/s)	46.09	29.28
STREAM scale (GB/s)	46.56	25.60
STREAM add (GB/s)	37.47	29.12
STREAM triad (GB/s)	57.44	27.04
RandomAccess (M Updates/s)	68.17	110.45

Table 6: Memory Throughput Using Stream and Random Access Benchmark From HPCC

6. CPU Only | Methodology and Results

This chapter aims to examine the CPU configurations of the TX1, X-Gene, and Xeon systems. Our methodology for evaluating these different systems is as follows. We first compare our mobile-class cluster to a cluster made of dedicated ARM servers, making use of the eight benchmarks from the NPB suite discussed in Chapter 3.C.i to help give insights on the best way build a system that most efficiently makes use of the ARM architecture. We then introduce a reference x86 Xeon server and use the same NPB benchmarks to understand how the CPUs on mobile-class ARM clusters compare to traditional servers in both performance and energy efficiency

A. Mobile ARM vs. Server ARM

The NPB benchmark suite was also used to help evaluate the CPUs of the TX1 and X-Gene. To make the comparison as meaningful as possible, we used four TX1 nodes and two X-Gene nodes, as both of these setups offer sixteen ARM cores each. The runtime and power consumption of both clusters for the eight NPB benchmarks are shown in Table 7. On average, the four TX1 nodes exhibit 30% better runtime while consuming 31% less power compared to the two X-Gene nodes. This translates to the cluster of four TX1 nodes being 2.1x more energy-efficient than the X-Gene cluster. The higher power consumption of the X-Gene SoC was likely due in part to the fact that it was manufactured using 40nm fabrication technology compared to the 20nm fabrication technology used for the TX1.

Benchmark	**TX1**		**X-Gene**	
	16 threads (4 nodes x 4 cores)		**16 threads (2 nodes x 8 cores)**	
	runtime (s)	power (W)	runtime (s)	power (W)

bt	241.5	98.2	432.9	129.6
cg	122.6	91.5	281.4	136.2
ep	40.1	88.1	47.0	128.8
ft	120.6	94.8	121.0	149.9
is	14.1	88.9	17.1	134.5
lu	299.8	94.5	354.7	133.5
mg	25.7	99.1	42.5	134.1
sp	341.7	94.1	624.4	129.0

Table 7:Runtime and Power Consumption for NPB on TX1 and X-Gene Clusters

However, when looking past differences in power consumption and analyzing raw performance by looking at the runtimes of the two different clusters, the fact that the 16 TX1 cores outperform the 16 more powerful, higher-frequency, X-Gene cores may seem surprising at first glance, especially given the X-Gene's additional 8MB L3 cache that is nonexistent on the TX1. In an attempt to explain this unexpected performance discrepancy, we collected and recorded values from many performance counters while running the various NPB benchmarks.

We were able to examine cache misses as a potential source performances differences by collecting and examining the counter for the number of DRAM accesses. When a system attempts to retrieve data at a particular address, it first checks its top-level L1 cache for the data. If there is a cache miss, it checks the next cache in the hierarchy for the data, continuing down the chain until it either finds the data or has looked in all available caches (L1 and L2 in the case of the TX1 and L1, L2, and L3 in the case of the X-Gene) without success. In the latter scenario of cache misses at all levels, the system then attempts to retrieve the data from its main memory (DRAM).

Repeated cache misses can prove to cause significant decreases in performance, as accessing the main memory is much slower than accessing the processor caches.

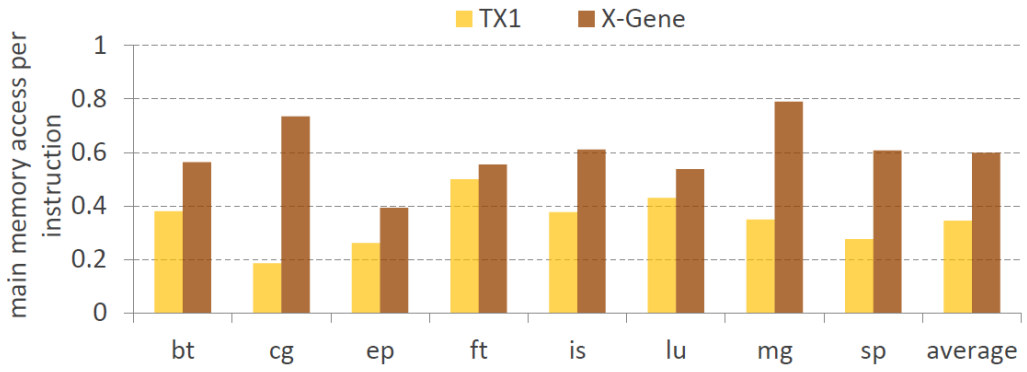

Figure 7:Main memory access per instructions for TX1 and X-Gene

We obtained number of main memory accesses per instruction for both the TX1 and X-Gene by dividing the main memory accesses counter by the retired instructions counter. The number of main memory accesses per instruction for the TX1 and X-Gene when running the NPB benchmarks is shown in Figure 7 We found that on average, the TX1 had about 70% less main memory accesses per instruction than the X-Gene despite the X-Gene's larger cache hierarchy. To ensure that compilation differences were not playing a part in the large difference in the number of main memory accesses, we executed the binaries compiled on the TX1 on the X-Gene and vice versa. Nevertheless, the results remained the same; the TX1 still accessed the main memory about 70% less frequently. Because cross-executing the binaries did not present us with any measurable difference in results, results from the natively-compiled binaries are included in Table 7.

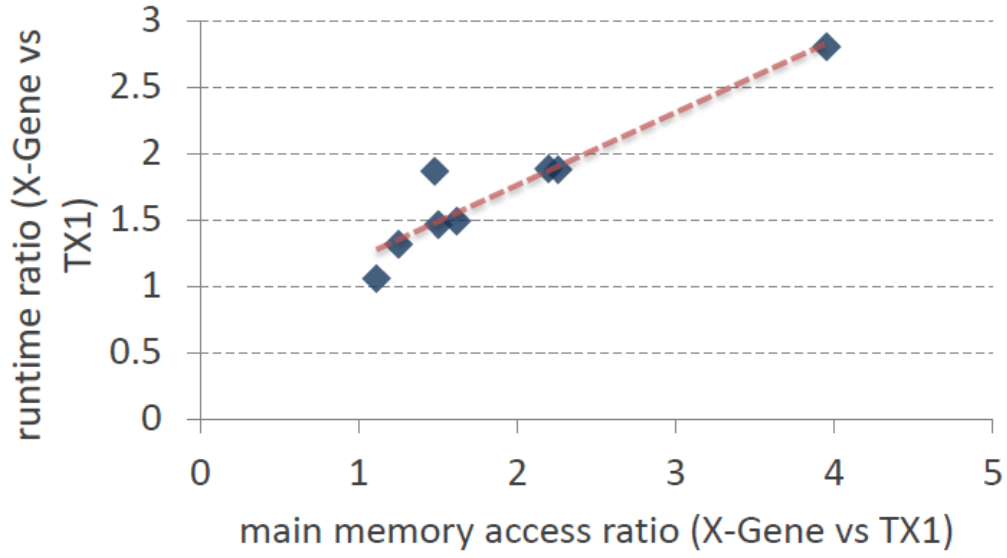

Figure 8: Comparing the runtime and main memory accesses of X-Gene and TX1

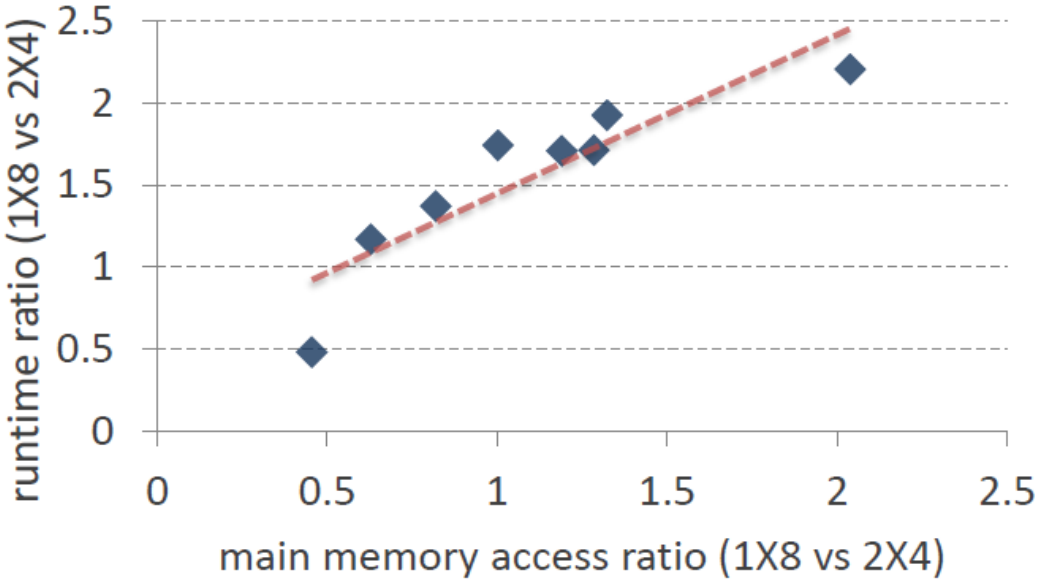

Figure 9: Comparing the run-time and main memory accesses of X-Gene configuration, using 8 cores of one X-gene vs. using 4 cores on two X-Gene.

The normalized runtime ratio of X-Gene to TX1 as a function of main memory access ratio for the NPB benchmarks is shown in Figure 8. The trend in the figure confirms that the primary culprit of the X-Gene's poor NPB performance is the design of its memory hierarchy. In the X-Gene, as

shown in Figure 10, each of the eight cores have their own 32KB L1 data cache and each pair of cores share a 256KB L2 cache; however, there is a single 8MB L3 cache shared between all eight cores. The L3 cache is connected to the eight cores by a 'Central Switch (CSW) Coherence Network.' This central switch provides a path between the CPU cores and I/O that does not necessarily pass through main memory. The coherence network offers cache coherency for the shared system cache to ensure that data is not lost before it is transferred from one of the caches to the target memory. Since multiple X-Gene CPU cores with separate L1 and L2 caches all share the same L3 cache and main memory, it is important to maintain a state of coherence between the caches by ensuring that any change of data in one cache is reflected throughout the entire system. To accomplish this, the X-Gene uses a snooping system, in which all caches 'snoop' the bus to determine whether they already possess a copy of the requested data. This system is largely responsible for the excessive number of main memory accesses, and therefore the poor performance, exhibited in the NPB benchmarks. The communication bus is prone to bottlenecks as more cores attempt to simultaneously access the single L3 cache.

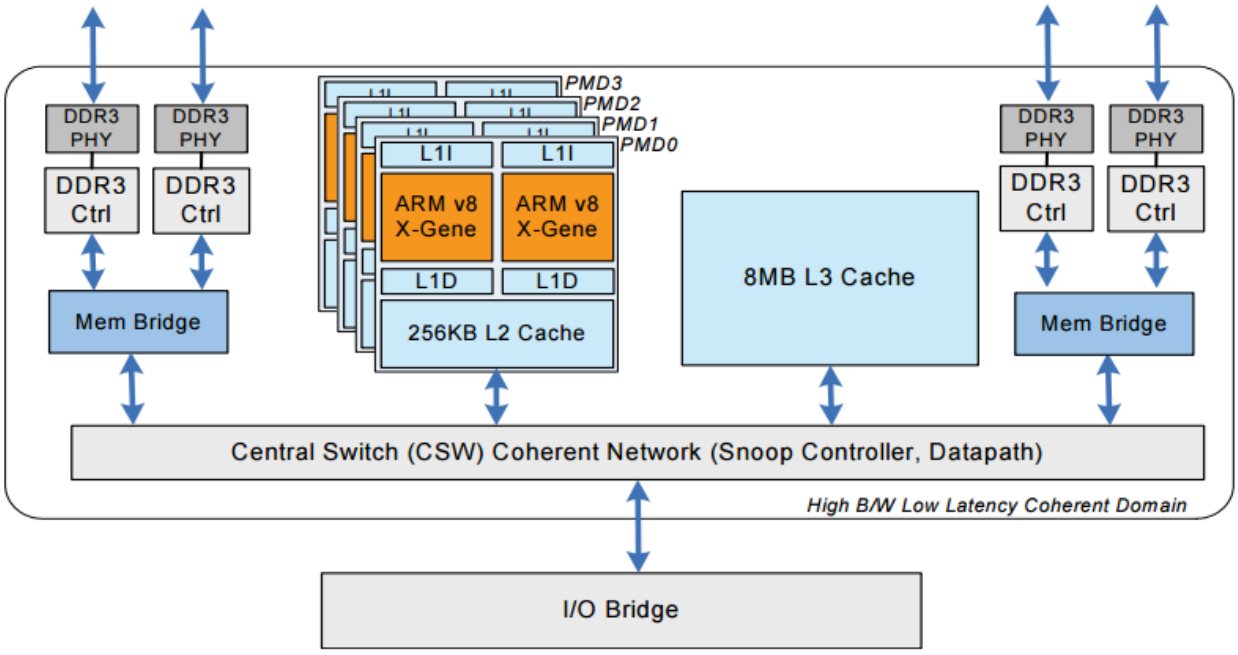

Figure 10: Partial block diagram of the X-Gene SoC showing the connection between the CPU cores and L3 Cache

Further testing was performed in order to confirm that the design of the connection between the CPU cores and the shared L3 cache is the primary cause of the X-Gene's poor performance in the NPB benchmark suite. The NPB benchmark tests were repeated using two different X-Gene configurations: one using all eight cores of a single X-Gene machine, and another using four cores each on two X-Gene machines connected with 10Gb network connectivity. The results of this experiment are plotted in Figure 9 as the runtime ratio of the single X-Gene to the two X-Genes as a function of the number of main memory accesses. Even though the two setups made use of an identical number of cores, the second configuration involving two X-Gene machines consistently outperformed the single X-Gene, despite the fact that the two X-Genes had the added overhead of inter-node communication over the network. The exception was the IS benchmark; here, the single X-Gene still performed better due to the fact that IS exhibits very high network activity, causing performance to significantly deteriorate when communication between two nodes was introduced.

Overall, the server-class X-Gene offers worse performance than the cluster we built using mobile-class NVIDIA TX1 SoCs. Despite the fact that the X-Gene possesses more capable CPU cores and a larger memory hierarchy, the poor design of its central switch connecting the CPU cores and their L1 and L2 caches to the shared L3 cache significantly reduces the speed of memory accesses when multiple cores are active. Furthermore, our mobile-class cluster that contains more nodes with less CPU cores per node boasts higher aggregate memory bandwidth, making it better suited for many HPC workloads.

B. Mobile ARM vs. x86

Despite the fact that the case for ARM servers is becoming stronger with every passing year as the ARM SoC market continues to advance and mature, there is no question that x86-based systems are still by far the most commonly used machines today for server-class workloads. While we demonstrated that heterogeneous clusters made of mobile-class SoCs possess many advantages over dedicated ARM servers, it is also important to understand how this type of systems compare to traditional x86 systems.

We first wanted to gain an understanding of the relative CPU performance of the TX1 cluster compared to the reference Xeon machine, before factoring in the TX1's GPGPU acceleration. To accomplish this, we used the same set of 8 NPB benchmarks, BT, CG, EP, FT, IS, LU, MG, and SP, as we did in chapter 6 when comparing the two different classes of ARM clusters. Our reference Xeon server nominally consumes the same amount of power as about 6 TX1 nodes. However, since many benchmarks, such as NPB, need the number of threads (and hence, number of cores), to be a multiple of two, we collected results using a cluster of 4 TX1 nodes and a cluster of 8 TX1 nodes for our experiments.

The normalized speedup of the TX1 clusters of 4 and 8 nodes compared to the Xeon server is shown in Figure 11. The results show that the performance across the 8 NPB benchmarks varies. The TX1 performance for a given benchmark heavily depends on its communication intensity. For benchmarks that are compute-bound, such as EP, BT, and SP, the TX1 clusters of 4 TX1 nodes and 8 TX1 nodes both outperform the reference Xeon server. However, in the IS, FT, and CG benchmarks, which are communication-bound, performed better on the Xeon server, as the advantage gained from having multiple TX1 nodes was offset by the increase in network activity, and therefore communication time.

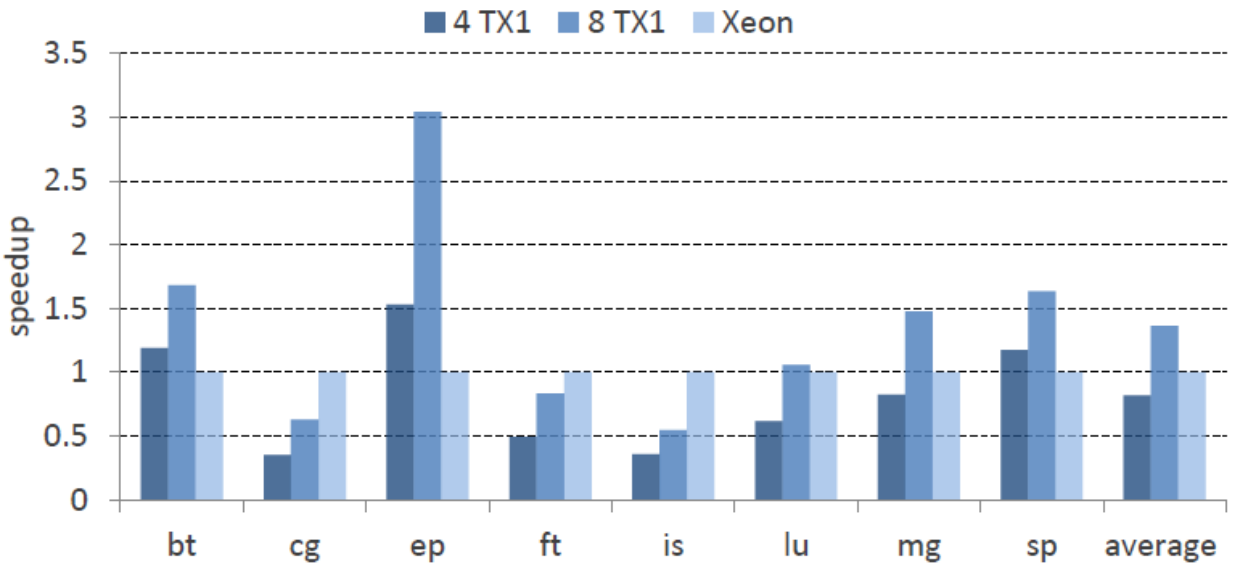

Figure 11: Normalized speedup of the TX1 clusters of 4 and 8 nodes compared to the Xeon server for the NPB Benchmarks

Energy efficiency between the Xeon server and TX1 clusters of 4 and 8 nodes was also compared. These results, measured in normalized throughput per watt, are given in Figure 12. The trends in energy-efficiency largely followed those of the speedup comparison. Similarly to the results seen with throughput, the TX1 clusters proved to be more energy efficient in the compute-bound benchmarks. However, for benchmarks that required high communication time,

the advantages of distributing the workload between multiple nodes was lost, causing the single Xeon node to be the most efficient of the three tested configurations.

The speedup and energy-efficiency results from the NPB benchmarks show that mobile-class ARM CPUs can provide performance and efficiency advantages for high-performance computing workloads that have low memory and network demands. Workloads that are characterized by intensive memory or network requirements, however, are better suited for x86 servers. The greater number of cores per node and superior memory hierarchy of x86 nodes can effectively reduce the amount of necessary network traffic and improve memory performance compared to clusters mobile-class systems.

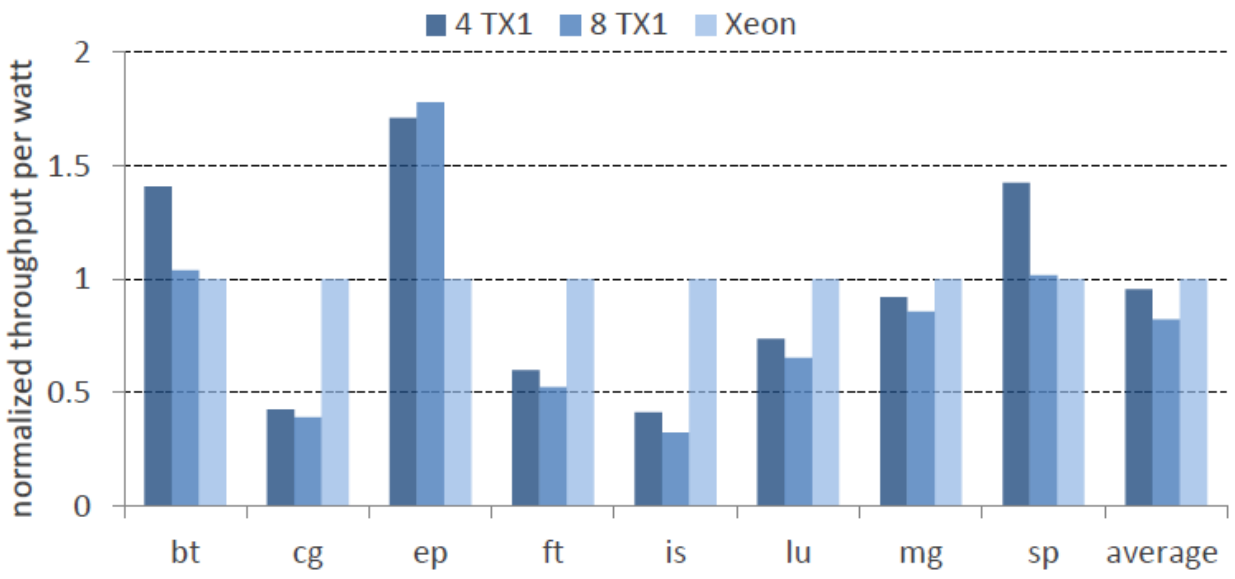

Figure 12: Energy efficiency of the Xeon server and TX1 clusters of 4 and 8 nodes for the NPB Benchmarks.

7. CPU + GPGPU: Impact of Heterogeneity | Methodology and Results

Each TX1 has 256 Maxwell CUDA GPU cores in addition to its quad-core ARM CPU. These GPU cores can be deployed as GPGPU cores, meaning that they can be used to perform many mathematical operations instead of just being used for graphics. Using GPGPU cores as accelerators carries the potential to dramatically increase performance in certain types of applications that have parallelizable operations. To test whether adding GPGPU cores to a mobile system is a viable alternative to server-class ARM machines, we performed two different types of experiments. The first evaluated performance using Rodinia benchmarks, which evaluate single-node performance, and can run on the CPU or GPU. As mentioned in chapter 3.C.ii, we the Leukocyte, Heart Wall, LU Decomposition, Needleman-Wunsch, SRAD, Streamcluster, and LavaMD Rodinia benchmarks for our experiment; as they have the longest runtimes, they help provide the most meaningful comparisons. The second type of experiment involved using the Caffe and TensorFlow deep learning frameworks to measure evaluation performance on the two types of clusters. The AlexNet and GoogleNet convolutional neural network models were used for evaluation on both the Caffe and TensorFlow frameworks for both clusters. A subset of 1,000 images from the ImageNet database were used to evaluate the image classification performance of the server-class cluster and the mobile, heterogeneous cluster.

A. Rodinia Single-Node Experiment

i. Mobile ARM vs. Server ARM

For the first, single-node experiment we compiled and ran the selected Rodinia benchmarks on three different configurations: first using the TX1 CPUs only, then using the TX1 GPUs, and finally using the X-Gene CPUs. The runtime of the Rodinia benchmarks on the three different

configurations is shown in Table 8. Relatively unsurprisingly, the X-Gene CPU outperforms the TX1 CPU in every Rodinia benchmark we tested, as the TX1's mobile-class ARM cores were no match for the more powerful, higher frequency CPU cores found on the X-Gene. However, it is clear that the TX1 sees a significant performance increase when incorporating its GPU cores. Running the Rodinia benchmarks on the TX1's GPU yields a 4.3x reduction in runtime over its CPU, and a 2.7x reduction in runtime compared to the X-Gene.

	Runtime (s)		
Benchmark	X-Gene (8 Cores)	TX1 CPU (4 Cores)	TX1 CPU + GPU
Heartwall	32.49	64.88	10.14
Leukocyte	71.03	105.24	5.95
LUD	46.01	63.96	3.16
Needle	4.83	6.36	2.96
SRAD v1	44.42	64.37	14.17
SRAD v2	36.45	54.63	19.28
Streamcluster	18.82	32.70	18.58
LavaMD	59.38	205.59	11.73
Normalized Average	1.00x	1.79x	0.39x

Table 8:Runtime of Rodinia on single nodes

Power consumption figures for the three Rodinia trials are shown in Table 9. As expected, the X-Gene consumed the most power across the benchmarks due to its heavier-duty CPU cores. The TX1's CPU consumed 6% more power than its GPU on average, despite the fact that the CPU performed much worse in the benchmarks. This further demonstrates the significance of the simultaneous performance gains and power consumption reduction. Additionally, the TX1 GPU

consumed 80% less power on average than the X-Gene even though the TX1 GPU outperformed the X-Gene in terms of runtime for every tested Rodinia Benchmark.

Benchmark	Power (W)		
	X-Gene (8 Cores)	TX1 CPU (4 Cores)	TX1 CPU + GPU
Heartwall	67.11	16.06	18.97
Leukocyte	65.10	12.70	13.33
LUD	59.50	13.97	8.44
Needle	58.49	13.49	7.74
SRAD v1	63.90	12.64	16.19
SRAD v2	67.35	13.63	14.84
Streamcluster	76.36	14.20	12.39
LavaMD	68.16	16.27	14.76
Normalized Average	1.00x	0.22x	0.20x

Table 9: Power consumption of Rodinia on single nodes

Finally, Table 10 shows the energy consumption for the X-Gene, TX1 CPU, and TX1 GPU. The trends seen in these energy consumption results largely mirror those of the power consumption results. The TX1 GPU proved to be 4.3x more energy efficient than its CPU and a notable 13.4x more energy efficient than the X-Gene. The summary of these results clearly demonstrates that workloads such as those seen in the Rodinia benchmarks stand to benefit from GPGPU acceleration. For these types of workloads, combining a less powerful mobile-class ARM CPU with a GPGPU can provide performance benefits as well as impressive improvements in energy efficiency.

Benchmark	Energy (kJ)		
	X-Gene (8 Cores)	TX1 CPU (4 Cores)	TX1 CPU + GPU
Heartwall	2.18	1.04	0.19
Leukocyte	4.62	1.34	0.08
LUD	2.74	0.89	0.03
Needle	0.28	0.09	0.02
SRAD v1	2.84	0.81	0.23
SRAD v2	2.45	0.74	0.29
Streamcluster	1.44	0.46	0.23
LavaMD	4.05	3.34	0.17
Normalized Average	1.00x	0.39x	0.07x

Table 10: Energy consumption of Rodinia on single nodes

ii. Mobile ARM vs. x86

While the ARM CPU found in a single TX1 SoC obviously cannot hold a candle to the x86 server's Xeon E5-2630 v3 CPU in terms of raw performance, we wanted to see how the landscape changed when adding the TX1's GPGPU to the equation. To do this, we used the same set of eight Rodinia benchmarks as in the previous chapter: Heartwall, Leukocyte, LUD, Needle, SRAD v1, SRAD v2, Streamcluster, and LavaMD. The runtimes for each of the Rodinia benchmarks when run on the Xeon server and the TX1 in CPU+ GPU mode are shown in Table 11. The results varied, with the TX1 performing better in some benchmarks, such as Heartwall and Leukocyte, but performing worse in others, such as SRAD v2 and Streamcluster. Averaging the results of all eight benchmarks, the Xeon server showed 16% better runtime than a single TX1 node in CPU + GPU mode.

Benchmark	Runtime (s)	
	Xeon E5-2630 v3	TX1 CPU + GPU
Heartwall	21.43	10.14
Leukocyte	102.86	5.95
LUD	4.26	3.16
Needle	1.51	2.96
SRAD v1	12.56	14.17
SRAD v2	9.91	19.28
Streamcluster	8.30	18.58
LavaMD	15.47	11.73
Normalized Average	1.00x	1.16x

Table 11: Runtime of Rodinia on a single TX1 Node (GPU) and Xeon server.

The variation in performance for the Rodinia benchmarks running on the heterogeneous TX1 and Xeon Server can be explained by the fact that the benchmarks utilized both the CPU and GPU cores of the TX1, but to different extents. We measured the average CPU and GPU utilization for each of the eight Rodinia benchmarks when running on the TX1 in an effort to better understand the breakdown of resources the benchmarks were using. These results are reported in Figure 13. In benchmarks that were found to have high GPU utilization and comparatively low CPU utilization, such as Heartwall, Leukocyte, LUD, and LavaMD, the TX1 node was able to outperform the Xeon server. On the other hand, for benchmarks that showed lower GPU utilization and higher CPU utilization, such as Needle, SRAD v2, and Streamcluster, the Xeon server outperformed the TX1 node. These results make sense. It is known that the Xeon CPU is superior to that of the TX1. The TX1's key to closing the performance gap is using

its GPGPU for acceleration. It follows, then, that benchmarks that are able to take better advantage of the GPU should perform better on the TX1, while benchmarks that do make as much use of the GPU should not see nearly as much of a performance boost. These insights are further confirmed by the fact that the gap in performance between the TX1 and Xeon server for the SRAD v1 benchmark, which exhibited both high CPU an GPU utilization, was small, with the TX1 and Xeon completing the benchmark in 14.17 seconds and 12.56 seconds, respectively.

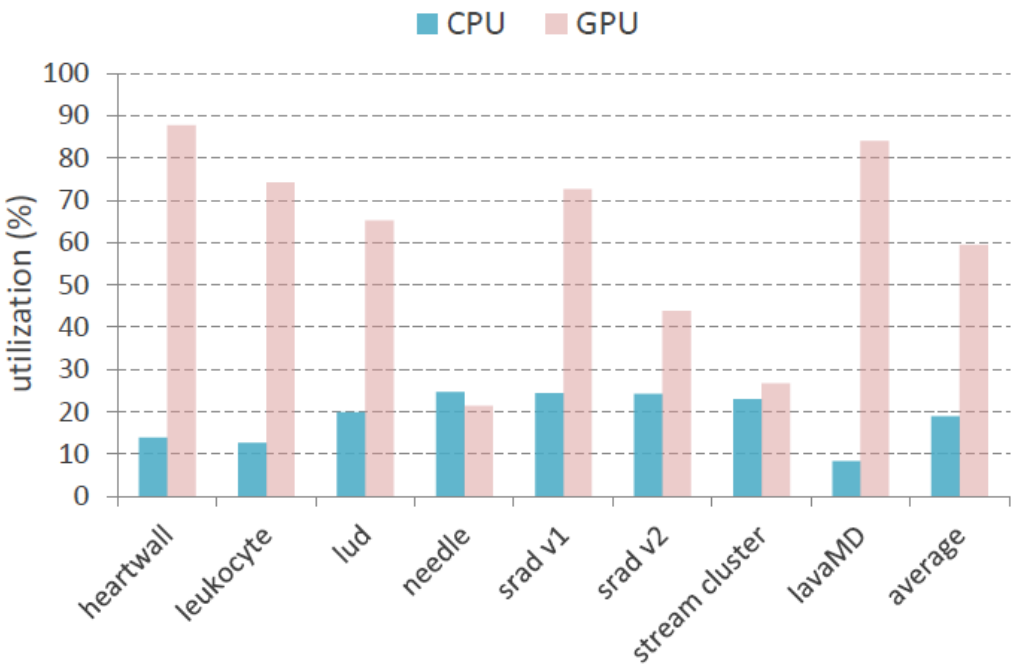

Figure 13: CPU and GPU utilization of each Rodinia benchmark running on a TX1 node

However, this 16% advantage in performance came at the cost of a 7.1x increase in power consumption over a TX1 node, as shown in Table 12. This means that while the Xeon was able to post overall higher performance, it was still trounced by the TX1 in energy efficiency. Table 13 shows the energy consumption in kJ for each of the benchmarks running on the TX1 and Xeon server. On average, the Rodinia benchmarks running on the TX1's CPU+GPU consumed 85% less energy than the Xeon server. These drastic energy savings demonstrate that adding

heterogeneity to a mobile-class CPU in the form of a GPGPU can go a long way toward closing the performance gap between it and a traditional server, all while still consuming minimal energy in comparison.

Benchmark	Power (W)	
	Xeon E5-2630 v3	TX1 CPU + GPU
Heartwall	83.12	18.97
Leukocyte	89.77	13.33
LUD	93.92	8.44
Needle	90.28	7.74
SRAD v1	95.70	16.19
SRAD v2	104.92	14.84
Streamcluster	107.32	12.39
LavaMD	98.57	14.76
Normalized Average	1.00x	0.14x

Table 12: Power Consumption of Rodinia on a Single TX1 Node (GPU) and Xeon Server

Benchmark	Energy (kJ)	
	X-Gene (8 Cores)	TX1 CPU + GPU
Heartwall	1.78	0.19
Leukocyte	9.23	0.08
LUD	0.40	0.03
Needle	0.14	0.02
SRAD v1	1.20	0.23

SRAD v2	1.04	0.29
Streamcluster	0.89	0.23
LavaMD	1.52	0.17
Normalized Average	1.00x	0.15

Table 13: Energy Usage of Rodinia on a Single TX1 Node (GPU) and Xeon Server

B. Image Classification Multi-Node Experiment

i. Mobile ARM vs. Server ARM

A multi-node experiment was also performed to evaluate the effects of heterogeneity in the context of the cluster as a whole. The experiment was performed using two of the leading deep learning frameworks, Caffe and Tensorflow, and measuring both the throughput and efficiency of the two different setups. Experiments in this chapter compare the two X-Gene systems with 8 TX1 nodes because both of these clusters consume around 150 watts for this workload. Attempting to normalize power in this manner helps to better understand the differences in the different setups' efficiencies. The performance of 2 X-Gene nodes, and 8 TX1 nodes are compared using both of the aforementioned deep learning frameworks and two different neural network models: AlexNet and GoogleNet. In each case, a set of 1000 images obtained from the ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) dataset [30] is evenly distributed between the eight TX1 nodes and the two X-Gene nodes. This allows the workload to scale in a near-ideal fashion as more nodes are added. This type of experiment is useful in that it provides an upper bound of efficiency for this type of cluster configuration and can serve as an example application that is well-suited for deployment on a large cluster. Caffe and Tensorflow are both run in CPU+GPU mode on the TX1 cluster, while they run parallelized on all 16 CPU cores of the X-Gene cluster. In the first experiment, the set of 1000 images is classified using the AlexNet and GoogleNet

model and Caffe framework. The second experiment uses the same image set and model, but is evaluated using the Tensorflow framework instead of Caffe. Note that the TX1 cluster used the 1Gb network for the TensorFlow experiment, as Linux For Tegra 24.2.1, which was needed for the TensorFlow experiment, did not work well with the 10Gb network adapters. The 1Gb network should not be a major factor in this experiment, as image classification can be considered close to an embarrassingly parallel workload.

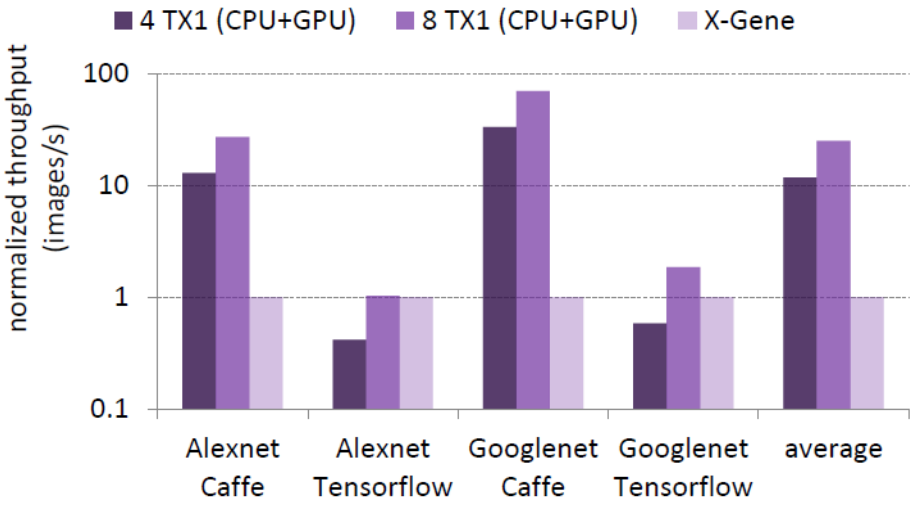

Figure 14: Normalized throughput, measured in classified images per second, for the TX1 and X-Gene clusters.

The results are given in Figure 14, where the y-axis uses a logarithmic scale. Averaging the AlexNet and GoogleNet results obtained using the Caffe framework, the 8 TX1 nodes outperformed the 2 X-Gene nodes by a wide margin, classifying images approximately 48x faster than the X-Gene. This significant performance gap is explained by the embarrassingly parallel nature of this type of workload, which makes it very suitable for computation on GPUs and their many cores compared to the more general architecture of a CPU. The results also show near-ideal scalability for comparing the results of 4 and 8 TX1 nodes. In the Tensorflow experiment, however, the results were much closer between TX1 and X-gene. When the AlexNet model was used, the 2 X-Gene systems nearly matched the throughput of the 8 TX1 nodes; the 8 TX1 nodes

only classified images about 2.4% faster than the X-Gene cluster. The TX1 cluster performed slightly better on the GoogleNet model using TensorFlow, classifying images at 1.85x the speed of the X-Gene cluster. However, this still pales in comparison to the performance it achieved using the Caffe framework. Taking both frameworks and both models into account, the 8-node TX1 cluster improved the throughput by 43% on average compared to 2 X-Gene nodes.

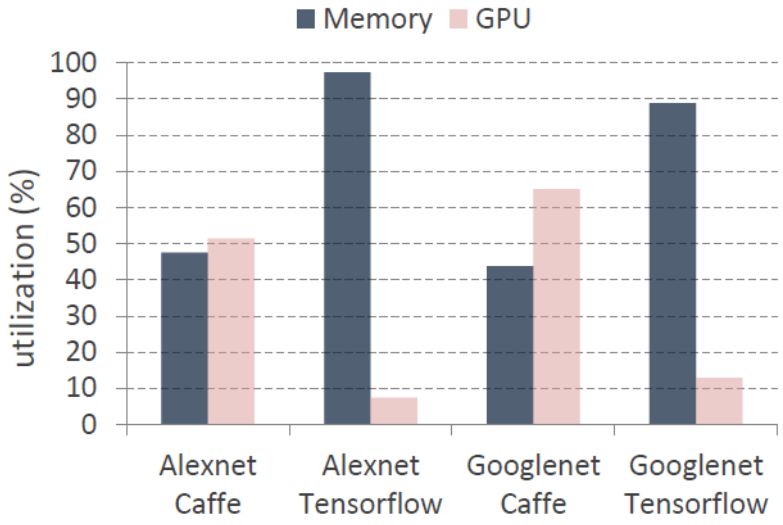

Figure 15: Memory and GPU utilization of the TX1 for both network models on both frameworks.

Given that the same network model and set of images were used in both the Caffe and Tensorflow experiments, the discrepancy in performance between the two frameworks when used for image classification on the TX1 might initially seem odd. However, we identified the limited total memory size (4 GB) on the TX1 board as the main culprit. The TX1 does not have dedicated GPU memory like most desktop or server-class GPUs. Rather, the TX1's 4GB of DRAM is shared between the CPU and GPU. While there are certainly benefits to having shared memory, such as increased flexibility, there are also certain drawbacks that became apparent in this experiment. Tensorflow has a much larger memory footprint than Caffe when being used to classify images, and therefore causes problems for the TX1 and its meager 4 GB of RAM that is shared between

the CPU and GPU. Because the TensorFlow application requires more memory to run, very little memory (sometimes under 1 GB) is left to allocate to the GPU. As a result, the GPU is starved of data to process and never comes close to reaching full utilization. As shown in Figure 15, AlexNet and GoogleNet running on the TX1 with Caffe use an average of 47.5% and 43.8% of their total RAM, respectively. This translates to 51.4% GPU utilization for AlexNet and 65.1% GPU utilization for GoogleNet when running the classification experiments using Caffe. However, when switching to Tensorflow, the average memory utilization of AlexNet increases to 97% and average memory utilization of GoogleNet increases to 88% of the total memory. This causes GPU utilization to drop to 7.4% and 12.9% on average, respectively.

It is worth noting that GoogleNet's slightly lower memory utilization, higher GPU utilization, and better performance is almost certainly due to the smaller size of the GoogleNet model compared to the AlexNet model. GoogleNet's main contribution was reducing the number of weight parameters to 2 million (compared to AlexNet's 60 million). This is also reflected in the size of the model file itself (23MB for GoogleNet compared to 60MB for AlexNet). Since GoogleNet has fewer parameters to store in memory, more memory remained available to allocate to the GPU. This means the GPU is able to store more data to process concurrently, leading to increased GPU utilization. It is also worth noting that the X-Gene, with its ample 16GB of RAM per node, did not appear to be adversely affected by the increase in memory utilization, as each X-Gene CPU core remained close to 100% utilization throughout the duration of the experiments.

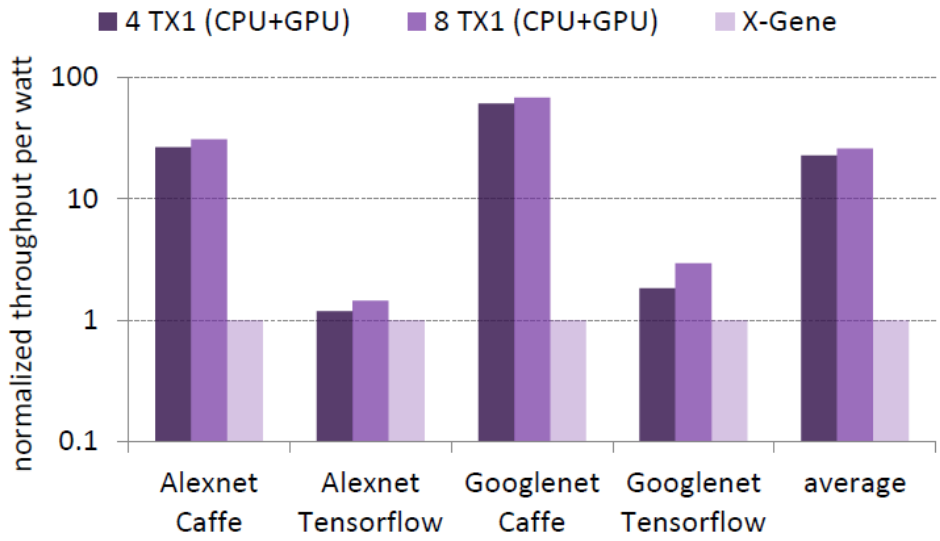

Figure 16: Normalized throughput per watt for TX1 clusters of 4 and 8 nodes compared to the X-Gene cluster.

Despite the memory issues experienced by the TX1 running Tensorflow, it still outperformed the X-Gene in both performance and energy efficiency, measured in (images/s)/W, in both experiments. The results given in Figure 16 show that on average, energy efficiency is improved by 25.7x. Caffe showed a higher energy efficiency of 49.3x compared to the 2.1x of Tensorflow because of the previously discussed issue with TensorFlow's memory utilization.

Our results show that incorporating general-purpose GPU cores into ARM SoCs can bring dramatic improvements to runtime, power consumption, and energy efficiency. As the landscape of relevant workloads continues to shift toward artificial intelligence and deep learning, it follows that modern hardware setups should be positioned to perform optimally for these types of applications. Our results indicate that planned efforts to integrate more cores into future server-based ARM SoCs may not be the best use of silicon area; rather, heterogeneous integration can lead to better results, especially for HPC applications that can take advantage of GPGPUs.

ii. Mobile ARM vs. x86

Incorporating the GPU of a single TX1 node brought a significant increase in performance and efficiency; however, a single TX1 node was never intended to compete with the Xeon server. The bigger picture, of course, is evaluating the performance and efficiency of the TX1 cluster compared to a traditional x86 server. To do this, we performed the same image classification experiment that was used to evaluate the different types of ARM clusters. Again, a set of 1,000 images taken from the ImageNet dataset were classified using four different configurations. Each of the two neural network models used, AlexNet and GoogleNet, were first run using the Caffe framework, then the TensorFlow framework. These four tests were run on a TX1 cluster of 4 nodes, a TX1 cluster of 8 nodes, and the reference Xeon server. Again, classifying a set of images is inherently an embarrassingly parallel task—the total set of images can easily be divided between multiple threads or nodes. This makes this type of workload ideal for clusters based on mobile SoCs, which will naturally consist of more nodes than their x86 counterparts. This held true in practice, as the throughput, measured in images per second, almost doubled when increasing the number of TX1 nodes in the cluster from 4 to 8.

In the AlexNet Caffe experiment, the cluster of 4 TX1 nodes consumed 54.2% less power than the Xeon server, yet classified images 12x faster. This translates to the 4 TX1 nodes providing a 20x increase in energy efficiency over the Xeon server. When increasing the size of the TX1 cluster from 4 nodes to 8 for the AlexNet Caffe, the TX1 cluster was able to classify images 25x as fast as the Xeon server. While the 8 TX1 nodes consumed 3.8% more power than the Xeon server during this experiment, it still was 24.5x more energy efficient. The results for the throughput and efficiency of the GoogleNet model on Caffe are very consistent with the

aforementioned results from AlexNet, showing that these comparisons hold true across multiple neural network models.

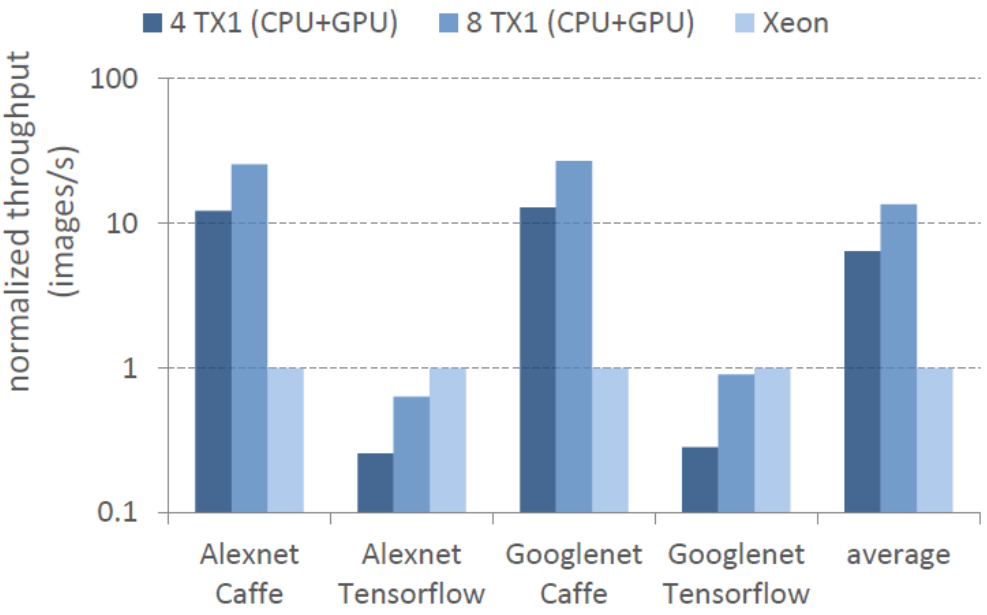

Figure 17:Normalized throughput measured in classified images per second for the Xeon server and TX1 clusters of 4 and 8 nodes.

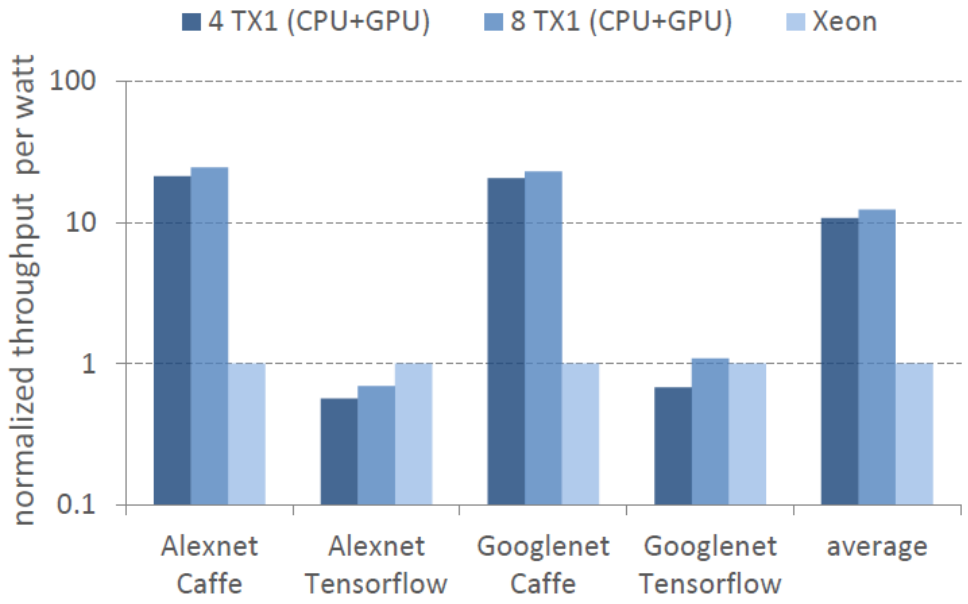

Figure 18: Normalized throughput per watt for the Xeon server and TX1 clusters of 4 and 8 nodes

Conducting these same experiments with AlexNet and GoogleNet on Tensorflow shows that the throughput of the Xeon improves greatly on both deep neural network models and is greater than that of the 8 TX1 nodes. As mentioned previously when discussing the deep learning experiment performed on the TX1 and X-Gene clusters, the TX1 performs very poorly in image classification using TensorFlow due to its memory limitations preventing its GPU from being properly utilized. This handicap allows the Xeon to report better efficiency than the TX1 cluster for AlexNet by 1.4x. However, the TX1 is still slightly more energy efficient than the Xeon for GoogleNet on Tensorflow. This is not unexpected, as the GoogleNet model's smaller size allows the TX1 to allocate more of its shared memory to its GPU compared to AlexNet.

C. Distributed vs. Centralized GPU

i. CUDA + MPI Benchmarks

While our experiments thus far have shown that taking advantage of the GPU cores found on mobile-class SoCs can significantly improve their performance and efficiency for many types of workloads, it is important to remember that GPUs are obviously not exclusive to mobile SoCs. The advent of many modern workloads which are more computationally intense and inherently parallelizable, along with the explosive popularity of deep learning, has helped make the case for equipping workstations and even servers with GPGPUs in recent years. While the typical approach to a GPGPUs for the purpose of accelerating mathematical operations has been to simply connect a discrete GPU 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, as discussed throughout this paper. Each TX1 SoC contains a CUDA GPU with 256 Maxwell cores, meaning that the total amount of GPU cores is evenly distributed throughout the

cluster. With these differences in mind, our goal was to determine whether a centralized or distributed GPU was best suited for a variety of applications.

We created a test setup for these series of experiments by adding an MSI NVIDIA GeForce GTX 960 Gaming 4G GPU to the Xeon server used for previous experiments. The GTX 960 was an ideal choice for GPU, as it uses the same core architecture as the TX1 GPU, but has four times the number of CUDA cores. While both GPUs technically have 4GB of memory, the TX1's available memory is shared between its GPU and CPU, meaning the full 4GB will never be available exclusively for the GPU. Specifications for the TX1 GPU and GTX 960 are given in Table 14.

	TX1 GPU	**MSI GTX 960**
Cores	256 Maxwell CUDA Cores	1024 Maxwell CUDA Cores
GPU Frequency	998.4 MHz	1315 MHz
Memory	4GB LPDDR4 shared between CPU and GPU	4GB GDDR5
L2 Cache	256KB	1MB
Nominal Power Consumption (W)	20W (For entire Jetson TX1)	120W

Table 14: Specifications of the TX1 GPU and the MSI version of the GTX 960

We used the five benchmarks, HPL, Jacobi, CloverLeaf, TeaLeaf2D and TeaLeaf3D, as described in chapter 3.C.iii, to analyze the centralized and distributed setups. Each of the benchmarks were run on the Xeon server with the GTX 960 GPU as well as 4, 8, and 16 TX1 nodes. Average power consumption was collected for each run as described in chapter 3.D. The dimensions of the matrices in all benchmarks were kept as close to square as possible for the

various TX1 cluster configurations. Figure 19 shows the speedup results from the five GPU benchmarks normalized to the performance of the Xeon + GTX 960 system.

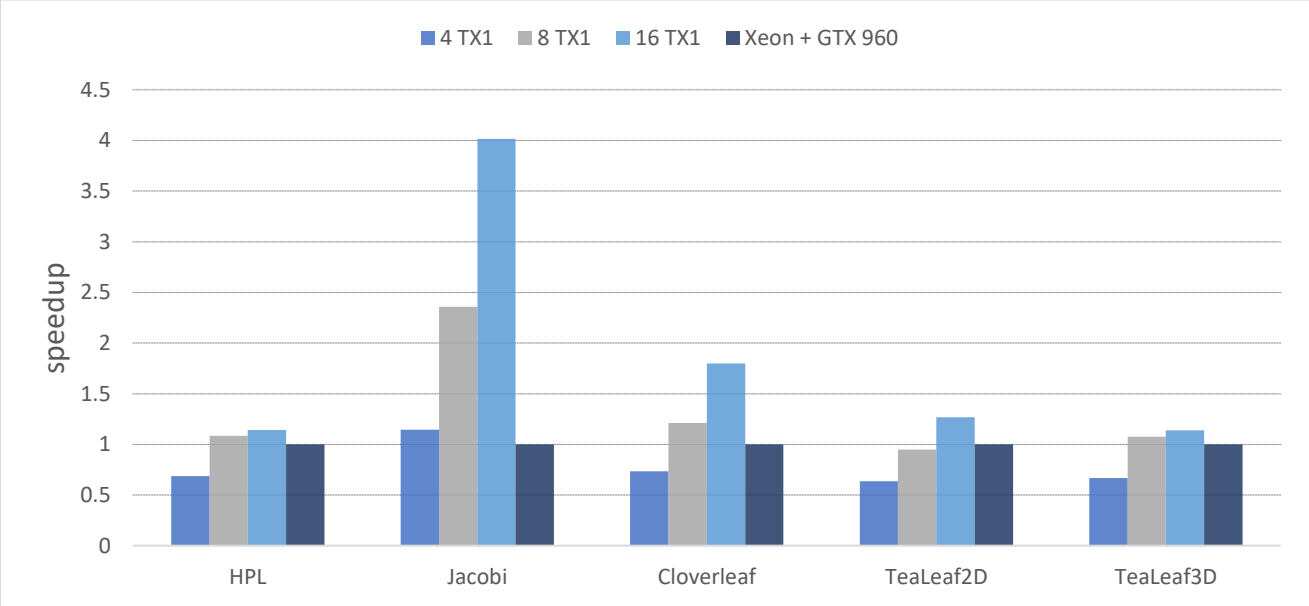

Figure 19: Speedup results from the GPU benchmarks normalized to the Xeon + GTX 960 system.

In terms of raw throughput, one interesting data point to observe is the comparison between 4 TX1 nodes and the GTX 960, as these two setups both make use of 1024 Maxwell CUDA cores, allowing for a more direct performance comparison between the two systems. While the centralized GPU would be expected to perform better in this scenario, given the communication between nodes introduced in the TX1 setup, the Jacobi benchmark notably did not follow this trend. The Jacobi benchmark exhibited good parallelizability and scalability, maintaining low network traffic throughout the benchmark which indicated little communication between the nodes was required. This is confirmed in Figure 20, which shows the per-node ratio of communication throughput for each of the benchmarks for 4, 8, and 16 nodes. The Jacobi benchmark shows the lowest communication requirements, which partially explains its superior performance in the distributed setting. This lower communication time allows the TX1 nodes to spend more time in

computation as shown in Figure 21. On average, the TX1 nodes showed the higher GPU utilization for the Jacobi compared to the other benchmarks by a wide margin. The trends seen in GPU utilization largely coincide with the speedup results presented in Figure 19.

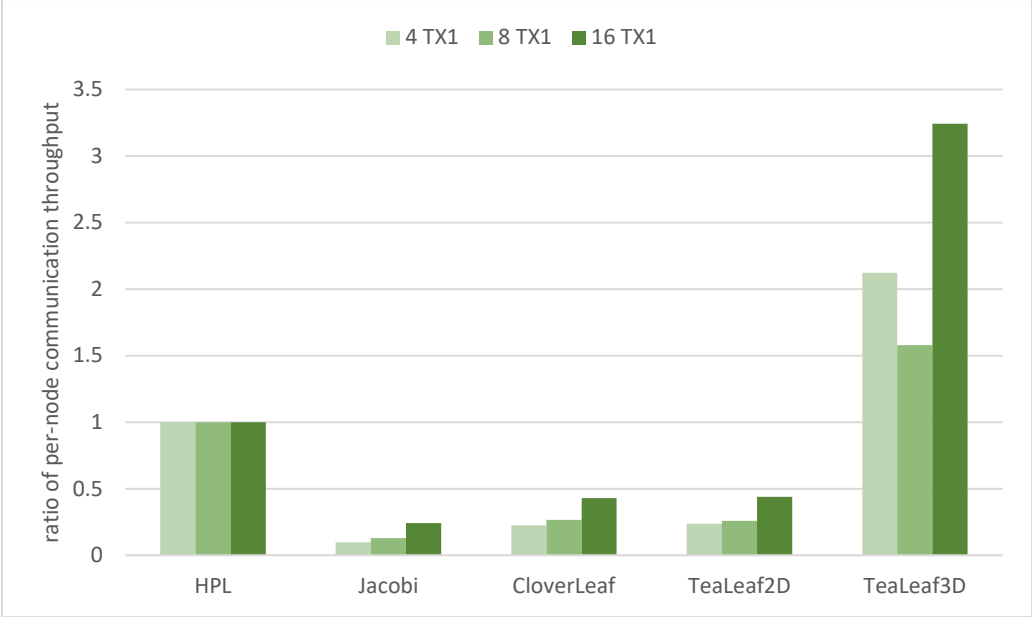

Figure 20: Ratio of per-node throughput of communication between nodes, normalized to HPL for 4, 8, and 16 TX1 nodes.

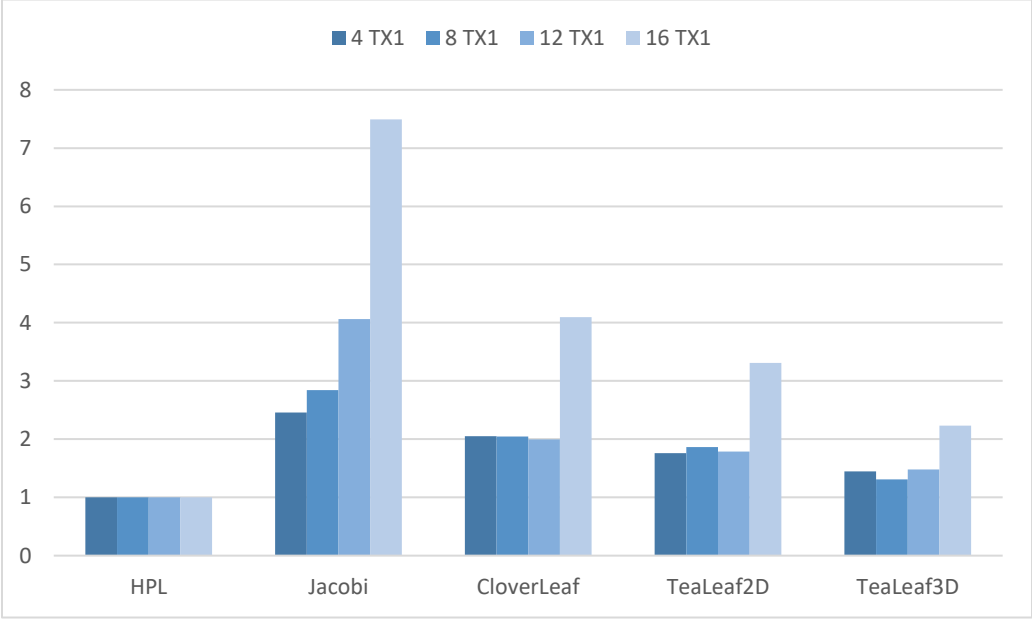

Figure 21: Ratio of GPU utilization for each benchmark for 8 TX1 nodes normalized to HPL.

Arguably the most interesting information gathered from Figure 20, however, is that the Jacobi benchmark has the highest memory demands of any of the benchmarks by a wide margin. To better understand the implications of this on overall performance, we examined the memory bandwidth of the two systems. Unfortunately, the TX1 GPU does not have the capability to report the dram_read_throughput or dram_write_throughput profiling metrics when using NVIDIA's nvprof GPU profiling tool, so we were forced to rely on the benchmark's reported memory bandwidth, which was calculated by measuring the time it took to transfer data of known size. The Jacobi benchmark reported an aggregate memory bandwidth of 104.01 GB/s for 4 TX1 nodes compared to 81.22 GB/s on the GTX 960. This difference highlights a key advantage of distributed GPUs: splitting the computational power of the GPUs between multiple nodes allows for higher potential memory bandwidth, increasing the speed at which data can be transferred to the GPU for computation. Furthermore, because the total amount of work is split between multiple nodes, each node is responsible for only a fraction of the total data processing and transfer.

The TX1's shared memory and integrated GPU also present an interesting opportunity for applications to gain a performance advantage over traditional systems. Typical workstations or servers have a GPU connected via a PCIe slot. These GPUs have their own memory (4 GB in the case of our GTX 960). In order to for the GPU to operate on data, the application first has to transfer the data from the host (CPU) to the device (GPU) over the PCIe bus. After the GPU finishes its computations, the data has to then be transferred from the device back to the host. In this design, the PCIe bus transmission speed becomes the bottleneck, especially when transferring very large amounts of data between the host and the device. However, The TX1 GPU is integrated as part of the SoC, meaning there is no PCIe bus over which the data must pass before being operated on by the GPU. Furthermore, the TX1 GPU does not have its own dedicated memory—

the 4GB of DRAM is shared between its CPU and GPU. This makes the CUDA model of copying data between the host and device redundant in this case.

In an attempt to improve performance by eliminating these seemingly unnecessary copies, we modified the Jacobi benchmark to employ NVIDIA's zero copy technique. Zero copy allows the GPU to directly access the host memory. In the CUDA programming model, this is implemented by allocating memory on the host for the data using *cudaHostAlloc()*, then passing the host pointer to the memory into *cudaHostGetDevicePointer()* to get a device pointer that points to the same host memory. Note that this programming technique is not commonly used for discrete GPUs, since data accessed on the host would still have to pass over the PCIe bus to reach the GPU.

We ran the original Jacobi benchmark and the zero copy version on a single TX1 node with a domain size of 4096x4096 to observe how the zero copy memory model affected performance.

To our surprise, the modified Jacobi benchmark with zero copy memory implemented performed about 8x worse than the unmodified version that employed *cudaMemcpy*(). After some further investigation, we determined that the reason for this large performance discrepancy is that zero copy memory is uncached on the TX1. We first confirmed that the implementation of zero copy correctly eliminated the calls to *cudaMemcpy()* using profiling data collected with NVIDIA's GPU profiling tool, nvprof [22]. The time spent in the *cudaMemcpy()* function for both runs is also shown in Figure 22. However, we found that in using zero copy memory to avoid these copies, the GPU completely ignored the TX1 memory hierarchy, directly accessing the DRAM every time it needed to retrieve data from memory. While the unmodified version of the Jacobi benchmark spent almost twelve seconds copying data between the host and the device, subsequent accesses to this data could be found in the L2 cache. Unfortunately, nvprof on the TX1 does not support the dram_read_transactions metric; however, we were able to use nvprof to profile the L2

cache utilization as well as the number of L2 cache misses for each run. As shown in Figure 22, the ratio of the average L2 cache utilization for the unmodified Jacobi benchmark to the zero copy-enabled version was 1:0.125, verifying that the zero copy version of the benchmark poorly utilized the TX1's memory hierarchy. Furthermore, we can assume that all L2 cache misses were then followed by DRAM read transactions, since the TX1 does not have an L3 cache. The zero copy version of the Jacobi benchmark showed twice as many L2 cache misses, and therefore twice as many DRAM accesses as the unmodified Jacobi benchmark. This further proves that the uncached nature of the zero copy memory on the TX1 was the reason it performed so poorly. Clearly, in our tests, the performance benefits obtained from utilizing the memory caches drastically outweigh the benefits of eliminating the copies between the host and device. It is evident that a memory system that allows memory shared between the CPU and GPU to utilize the caches would give the TX1 and other embedded SoCs with shared memory a large performance advantage over traditional systems.

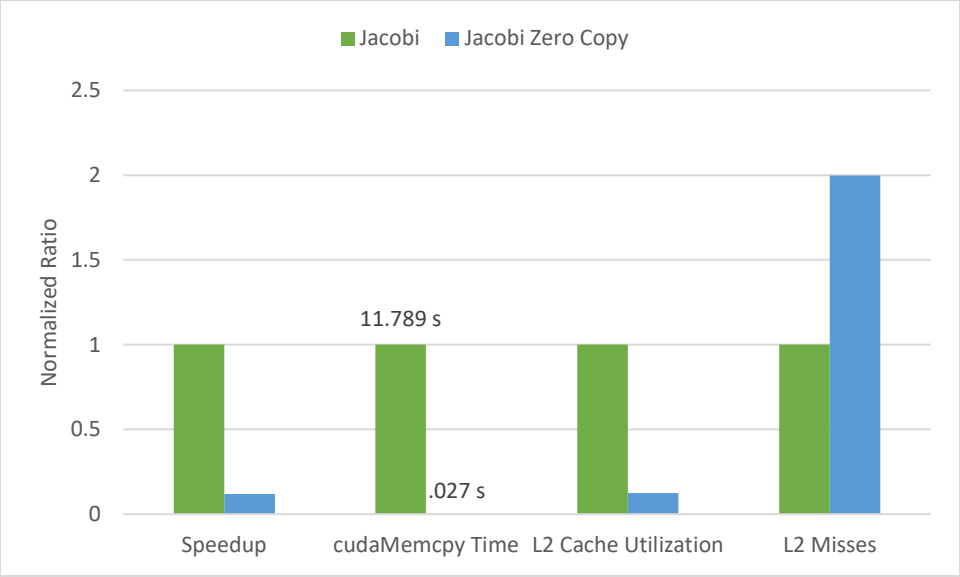

Figure 22: Ratios for speedup, time spent in cudaMemcpy, and average L2 cache utilization for Jacobi with zero copy, normalized to the unmodified Jacobi benchmark on a single TX1 node.

Figure 23 shows the energy efficiency ratio between 4, 8, and 16 TX1 nodes, normalized to the reference Xeon + GTX 960 system, for each of the five GPU benchmarks. Just as in the performance results, the TX1 cluster appeared to be best suited for the Jacobi benchmark, as it was most energy efficient when running this benchmark. The benchmark's low communication demands and high memory bandwidth requirements allowed the application to efficiently utilize each TX1 node in the cluster. Other benchmarks with very high communication needs, such as HPL, were not nearly as energy efficient. For four of the five benchmarks, the cluster became less energy efficient as more nodes were added. This shows that the additional inter-node communication more than counteracted the memory bandwidth benefits provided by these additional nodes in these network-bound applications. The extra network communication caused the 10Gb network cards on each node to draw more power and did not allow the GPUs to be as efficiently utilized, since the network-bound applications could not feed the GPUs with data fast enough.

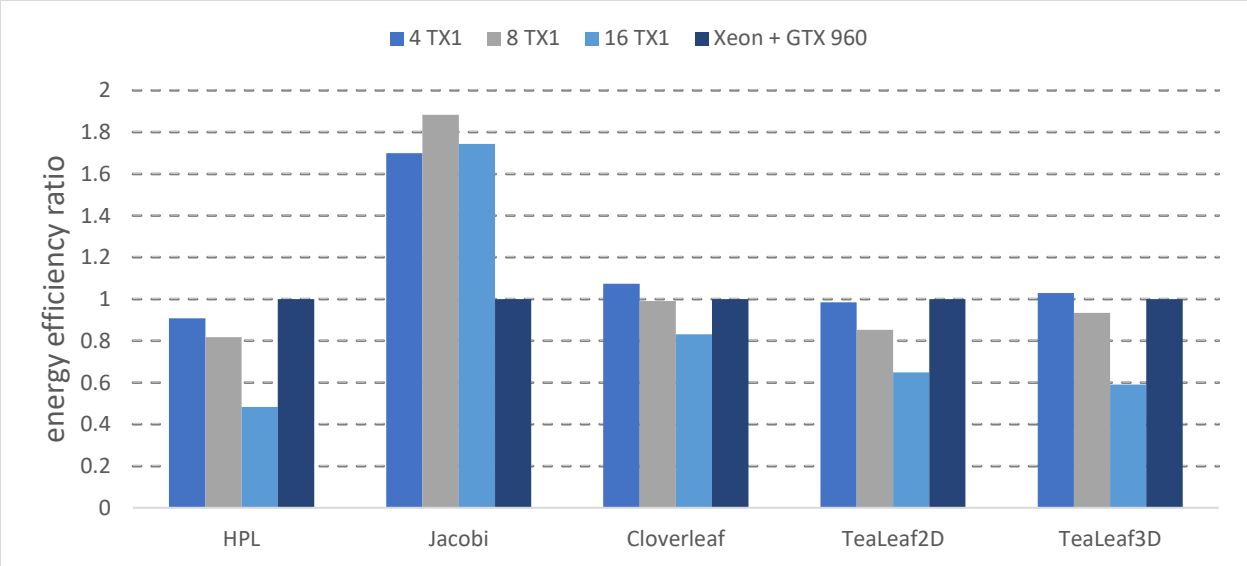

Figure 23: Energy efficiency ratio of 4, 8, and 16 TX1 nodes for the GPU benchmarks normalized to the Xeon + GTX 960 System.

The results from the HPL, Jacobi, CloverLeaf, TeaLeaf2D and TeaLeaf3D benchmarks demonstrate that a distributed GPU setup can be advantageous for certain types of workloads. Applications that are very parallel and are characterized by low communication time between threads or processes and high memory bandwidth demands are generally well-suited for distributed clusters. The higher memory bandwidth, and in the case of certain embedded boards, shared main memory, of clusters made from embedded SoCs can help data reach the GPU more quickly, increasing GPU utilization and leading to better performance. Allowing shared zero copy data accessible by both the CPU and GPU should be cached, as this would give these types of embedded SoCs with shared DRAM between the CPU and GPU to gain a large advantage over traditional systems, especially when working with copious amounts of data. Applications that are network-bound, however, will never be able to scale to the large numbers of nodes these types of ARM clusters made from embedded SoCs require. Network communication is inherently at least an order of magnitude slower than DRAM accesses; as such, the increases in memory bandwidth will never be able to offset an application constrained by inter-node communication.

ii. TensorFlow Distributed Training

TensorFlow offers a framework for training network models in a distributed environment. This can be done by sharing all model parameters across all worker nodes while parallelizing data and gradient updates. For distributed training, TensorFlow divides processes into two distinct types of jobs: parameter servers and workers. Parameter servers act as centralized repositories for parameters by maintaining a single, unified copy of model parameters and weights. The parameter server job can be run on the CPU; as it is just responsible for storing and distributing data, it does not benefit from GPU acceleration. Workers are stateless nodes that can

perform computationally-intensive tasks and are responsible for most of the heavy lifting during training.

We first experimented with different configurations of parameter servers and workers within the TX1 cluster. To do this, we trained a convolutional MNIST model to 20,000 training steps using a variety of configurations. Note that as in Chapter 7.B, the 1Gb network was used, as the 10Gb network adapters did not work well with Linux for Tegra 24.2.1, which was used for the TensorFlow experiments. A table of the training performance, measured in steps per second, for various configurations of 1, 2, 4, and 8 TX1 nodes is shown in Table 15.

Since the parameter server job runs on the CPU and the worker job runs on the GPU, it would intuitively seem as though the ideal configuration would be to run a parameter server on each the CPU and a worker on each GPU of all TX1 nodes in the cluster (i.e. where #Nodes = #Workers = # PS), as this would minimize required network activity per node. However, this was not the case for our setup. Running a parameter server on the CPU and worker on the GPU of the same node simultaneously requires two separate instances of TensorFlow in separate processes to run. Just as before, the 4GB of memory on each TX1 node was a limitation here. Because running the first parameter server instance of Tensorflow consumed a sizable portion of the shared memory, little memory was left to allocate to the GPU on the worker job. Using a node as a dedicated parameter server with no worker so that no 2 nodes were running multiple processes also was not ideal. Because this experiment was limited to 1Gb network connectivity, having one node running both a parameter server and worker to take advantage of being able to transfer parameters over localhost proved to be the best compromise and lead to the best performance.

# TX1 Nodes	# PS	# Workers	Normalized Steps/Second
1	1	1	1
2	1	1	0.53
2	1	2	1.32
2	2	2	1.09
3	1	3	1.51
3	1	2	0.79
3	2	3	1.45
3	3	3	1.306639
4	1	4	1.679622
4	2	4	1.56966
4	4	4	1.421694
8	1	8	1.896646973

Table 15: Training performance measured in steps/second on an MNIST network trained to 20,000 steps for various parameter server/worker configurations for 1, 2, 4, and 8 TX1 nodes. Note that 1Gb network connectivity was used for this experiment, as the 10Gb adapters did not work well with L4T 24.2.1 and CUDA 8.0

The best parameter server and worker configuration for TX1 cluster size was compared to training on the Xeon + GTX 960 system. Performance results for 1, 2, 4, and 8 TX1 nodes collected while training a convolutional MNIST model to 20,000 steps and a CIFAR-10 model to 500 steps, normalized to the Xeon + GTX 960 machine, are shown in Figure 24. The memory and network limitations of the TX1 cluster in this experiment proved to be too much of a bottleneck for the TX1 to be competitive with the Xeon and GTX 960. 8 TX1 nodes, despite having twice the number of Maxwell CUDA cores as in the GTX 960, was only able to achieve about 60% of the performance of the GTX 960 on average in these experiments. TX1 nodes with more memory per node as well as resolving the issues with the 10Gb network in this setup

would allow the TX1 to much more efficiently use its CPU and GPU cores, and likely change the landscape in this comparison.

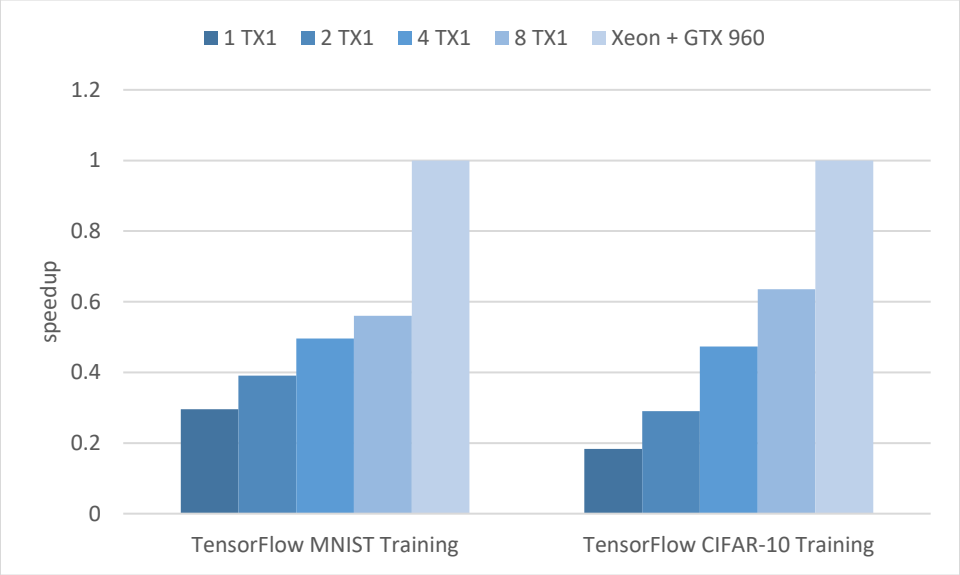

Figure 24: TensorFlow training performance for a convolutional MNIST model trained to 20,000 steps and a CIFAR-10 model trained to 500 steps for 1, 2, 4, and 8 TX1 nodes normalized to the Xeon + GTX 960 system.

8. Impact of 10Gb Network and GPGPU Acceleration in HPL

In summary, our HPC cluster made using TX1 SoCs differs from previous mobile-class clusters in two major ways: 1) It uses 10Gb network connectivity for communication between nodes and 2) it uses distributed, integrated GPGPUs on the TX1 SoCs to provide computational acceleration. To quantify the benefits obtained from each of these contributions, we performed one final experiment with HPL. We first ran HPL using only the CPU cores of the TX1 nodes connected using the 1Gb network to obtain a reference point for performance before adding any accelerators. We then ran HPL again, still using only CPU cores, but now adding 10Gb network connectivity for inter-node communication. Finally, we ran HPL using the CPUs, 10Gb network connectivity, as well as the integrated GPGPUs. Note that for this final experiment using the GPUs, only three CPU cores were used for HPL computation, as the fourth CPU core was reserved for feeding data to the GPGPU.

The performance and efficiency results for the experiments are shown in Figure 25. The addition of 10Gb network connectivity yielded a 75% increase in energy efficiency over TX1s using the 1Gb NIC. Adding the GPGPU provided a further 41% increase in energy efficiency over the configuration using only CPU cores and the 10Gb network. Overall, the addition of both the 10Gb network and GPGPU cores provided a 116% improvement in energy efficiency compared to the TX1 cluster without either of the accelerators enabled. The energy efficiency results from the previous Tibidabo and Snowball systems discussed in Chapter 2 are also included in the figure.

The TX1 cluster with both accelerators achieved a 255% and 106% increase in energy efficiency over Tibidabo and Snowball, respectively [24], [28].

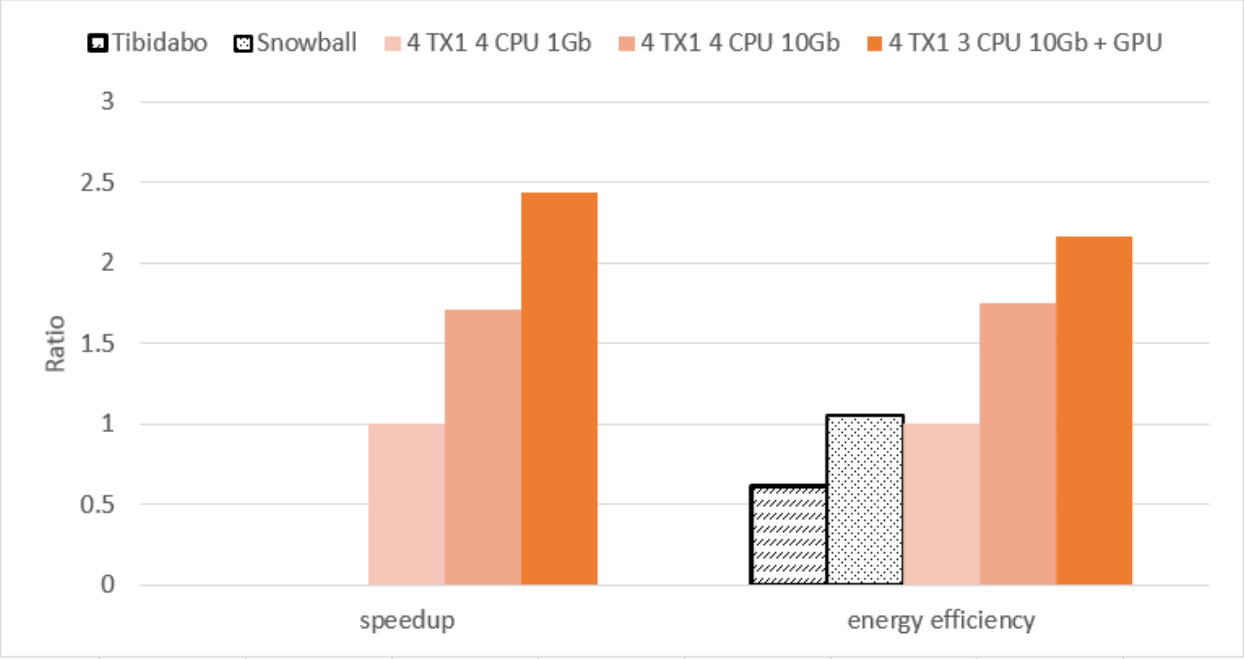

Figure 25: HPL speedup and efficiency for 4 TX1 nodes using only CPU cores with 1Gb network, CPU cores with 10Gb network, and CPU+GPU cores using 10Gb network compared to Tibidabo and Snowball

Benchmark	Tibidabo	SnowBall	4 TX1 4 CPU 1Gb	4 TX1 4 CPU 10Gb	4 TX1 3 CPU 10Gb + GPU
Energy Efficiency MFLOPS/W	120	206.6	196.7	344.3	426.6

Table 16: Energy efficiency results expressed in MFLOPS per Watt for the five configurations shown in Figure 25

9. Conclusion

In this paper, we explored the two different approaches to creating ARM clusters intended for high-performance computing applications, compared these systems to a traditional, reference x86 machine, and analyzed the differences between using distributed and centralized GPUs. We showed that mobile-class ARM SoCs now perform well enough to take advantage of the additional bandwidth and improved latency provided by 10Gb network controllers, which are needed to create a truly competitive cluster. We have also demonstrated that a large number of higher performing ARM CPU cores on a single chip does not necessarily guarantee better performance, especially when the network on a chip connecting the CPU cores to a shared L3 cache is poorly designed.

Furthermore, this paper showed that mobile-class ARM SoCs using integrated GPGPUs as accelerators are capable of outperforming both server-class ARM SoCs as well as traditional x86 servers in both performance and energy consumption for certain types of applications. Finally, we have shown that distributed processing using distributed GPUs across a cluster of mobile-class ARM SoCs connected by a 10Gb network can provide the nodes' individual cores with larger aggregate memory bandwidth, leading to improved performance compared to a centralized GPU for applications that are not bounded by network traffic.

Finally, we demonstrated the impact of our 10Gb network and GPGPU acceleration using the HPL benchmark. We showed that the addition of 10Gb network connectivity and GPGPU cores to the cluster resulted in a 116% energy efficiency over the same cluster with the accelerators disabled. Furthermore, our final cluster with 10Gb network connectivity and GPGPU cores demonstrated 255% and 106% increases in energy efficiency over the previous Tibidabo and Snowball systems.

There are multiple areas in which future work could be focused. While we created a cluster of 16 nodes, it could be beneficial to see how different applications scale to much larger clusters. To do this, scalability models could be created to simulate clusters of arbitrary sizes for a variety of applications. This would help understand which applications scale in such a way that would be well-suited for mobile-class ARM clusters and allow us to compare to much larger traditional, x86 clusters. Another future area of focus could involve looking how to better exploit the fact that the DRAM of embedded boards is shared between the CPU and GPU to improve the efficiency with which data is transferred to the GPU for processing. Finally, embedded boards would benefit from more memory per node since their memory is shared between the CPU and GPU. More memory per node would allow for intensive applications to run multiple processes to take advantage of the computational resources of both the CPU and GPU cores simultaneously.

10. Appendices

A. Appendix A

Patch for the tn40.h file in the Tehiti_TN4010 Linux driver:

```
diff --git Tehuti_TN4010/Linux/tn40.h Tehuti_TN4010_arm/Linux/tn40.h
@@ -933,7 +933,7 @@
 #endif

 #if LINUX_VERSION_CODE >= KERNEL_VERSION(2, 6, 31)
-#define USE_PAGED_BUFFERS             1
+//#define USE_PAGED_BUFFERS             1
 /*
  * Note: 32 bit  kernels use 16 bits for page_offset. Do not increase
  *       LUXOR__MAX_PAGE_SIZE beyind 64K!
@@ -944,7 +944,7 @@
 #define LUXOR__MAX_PAGE_SIZE    0x10000
 #endif
 #elif defined(RHEL_RELEASE_CODE) && (RHEL_RELEASE_CODE >= 1285) &&
defined(NETIF_F_GRO)
-#define USE_PAGED_BUFFERS             1
+//#define USE_PAGED_BUFFERS             1
 /*
  * Note: RHEL & CentOs use 16 bits for page_offset. Do not increas
  *       LUXOR__MAX_PAGE_SIZE beyind 64K!
```

Patch for the dma-mapping.c file in the TX1 kernel:

```
diff --git kernel/arch/arm64/mm/dma-mapping.c kernel_new/arch/arm64/mm/dma-
mapping.c
@@ -2112,7 +2112,8 @@
         * compound page then there's probably a bug somewhere.
         */
        if (page_offset > 0)
-               BUG_ON(page_count(page) == 1);
+               BUG_ON(page_offset > (1 <<
compound_order(compound_head(page))) - ((page - compound_head(page)) <<
PAGE_SHIFT));
+               //BUG_ON(page_count(page) == 1);

        dma_addr = __alloc_iova(mapping, len, attrs);
        if (dma_addr == DMA_ERROR_CODE)
```

11. References

[2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 17, 18, 19, 23, 24, 26, 27, 28, 29, 30, 31, 33, 35, 9, 1]

[32, 21, 20, 34, 25, 16, 15, 22]

[1] NLANR/DAST : Iperf - the TCP/UDP bandwidth measurement tool. http://dast.nlanr.net/Projects/Iperf/, Accessed 2007.

[2] Martin Abadei *et al.* Tensorflow: Large-scale machine learning on heterogeneous systems, 2015.

[3] 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 http://arxiv.org/abs/1410.3441*, 2014.

[4] D. Andersen, J. Franklin, and M. Kaminsky. FAWN: A Fast Array of Wimpy Nodes. *Proceedings of the Symposium on Operating Systems Principles*, pages 1–14, 2009.

[5] R. V. Aroca and M. G. Gonçalves. Towards green data centers: A comparison of x86 and ARM architectures power efficiency. *Journal of Parallel and Distributed Computing*, 72(12):1770–1780, 2012.

[6] E. Blem, l. Menon, and K. Sankaralingarn. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In *International Symposium on High Performance Computer Architecture*, pages 1–12, 2013.

[7] Ben Cumming. cuda-stream. https://github.com/bcumming/cuda-stream, 2016.

[8] Saumitro Dasgupta. caffe-tensorflow. https://github.com/ethereon/caffe-tensorflow, 2016.

[9] Message P Forum. Mpi: A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.

[10] M. Jarus, S. Varrette, A. Oleksiak, and Pascal Bouvry. Performance Evaluation and Energy Efficiency of High-Density HPC Platforms Based on Intel, AMD and ARM Processors. In *Lecture Notes in Computer Science*, volume 8046, pages 182–200, 2013.

[11] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[13] Z. Krpic, G. Horvat, D. Žagar, and G. Martinovic. Towards an energy efficient SoC computing cluster. Ín *International Convention on Information and Communication Technology, Electronics and Microelectronics*, pages 184–188, 2014.

[14] W. Lang, J. Patel, and S. Shankar. Wimpy Node Clusters: What About Non-Wimpy Workloads? In *International Conference on Management of Data*, pages 47–55, 2010.

[15] A. C. Mallinson, David A. Beckingsale, W. P. Gaudin, J. A. Herdman, J. M. Levesque, and Stephen A. Jarvis. Cloverleaf : preparing hydrodynamics codes for exascale. In *A New Vintage of Computing : CUG2013*. Cray User Group, Inc., 2013.

[16] Andy Mallinson, David Beckingsale, and Oliver Perks. Uk mini-app consortium (ukmac) github page. http://uk-mac.github.io/.

[17] J. Maqbool, S. Oh, and G. C. Fox. Evaluating arm hpc clusters for scientific workloads. *ncurrency Computat.: Pract. Exper*, 27:5390–5410, 2015.

[18] Larry W McVoy, Carl Staelin, *et al.* lmbench: Portable tools for performance analysis. In *USENIX annual technical conference*, pages 279–294. San Diego, CA, USA, 1996.

[19] nattoheaven. cuda_randomaccess. https://github.com/nattoheaven/cuda_randomaccess, 2016.

[20] NVIDIA. Jetson tx1 developer kit. https://developer.nvidia.com/embedded/buy/jetson-tx1-devkit.

[21] NVIDIA. Gpu-based jacobi solver. https://github.com/parallel-forall/code-samples/tree/-master/posts/cuda-aware-mpi-example/src, 2015.

[22] NVIDIA. nvprof. http://docs.nvidia.com/cuda/profiler-users-guide/#nvprof-overview, 03 2017.

[23] Z. Ou, B. Pang, Y. Deng, J. K. Nurminen, A. Yla-Jaaski, and P. Hui. Energy- and Cost-Efficiency Analysis of ARM-Based Clusters. In *International Symposium on Cluster, Cloud and Grid Computing*, pages 115–123, 2012.

[24] E. L. Padoin, D. A. G. de Oliveira, P. Velho, P. O. A. Navaux, B. Videau, A. Degomme, and J.-F. Mehaut. Scalability and Energy Efficiency of HPC cluster with ARM MPSoC. In *Workshop on Parallel and Distributed Processing*, pages 1–4, 2013.

[25] Antoine Petitet, Clint Whaley, Jack Dongarra, and Andy Cleary. Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers. http://www.netlib.org/benchmark/hpl/, 02 2016. Version 2.2.

[26] X. Zhan R. Azimi and S. Reda. How good are low-power 64-bit socs for server-class workloads? In *IEEE International Symposium on Workload Characterization*, pages 116–117, 2015.

[27] 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*, number 40, pages 1–12.

[28] N. Rajovic, N. Puzovic, and A. Ramirez. Tibidabo: Making the Case for an ARM Based HPC System. *ElSevier Future Generation Computing Systems*, 36:322–334, 2014.

[29] N. Rajovica, L. Vilanovaa, C. Villaviejaa, N. Puzovica, and A. Ramirez. The low power architecture approach towards exascale computing. *ElSevier Journal of Computational Science*, 4(6):439–443, 2013.

[30] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[31] J. Meng D. Tarjan J. W. Sheaffer S.-H. Lee S. Che, M. Boyer and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing? In *IEEE International Symposium on Workload Characterization*, pages 44–54, 2009.

[32] Startech. Drivers & downloads. https://www.startech.com/Networking-IO/Adapter-Cards/10gb-pcie-nic~ST10000SPEX.

[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[34] Princeton University. Wordnet: A lexical database for english. http://-wordnet.princeton.edu/, 2005.

[35] A. Yeung, H. Partovi, Q. Harvard, L. Ravezzi, J. Ngai, R. Homer, M. Ashcraft, and G. Favor. A 3GHz 64b ARM v8 processor in 40nm bulk CMOS technology. In *International Conference Solid-State Circuits*, pages 110–112, 2014.