CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3602

Evaluating ARM HPC clusters for scientific workloads

Jahanzeb Maqbool[1], Sangyoon Oh[1,2,*,†] and Geoffrey C. Fox[2]

[1]*Department of Computer Engineering, Ajou University, 446-749, Korea*
[2]*Pervasive Technology Institute, Indiana University, Bloomington, IN, USA*

SUMMARY

The power consumption of modern high-performance computing (HPC) systems that are built using power hungry commodity servers is one of the major hurdles for achieving Exascale computation. Several efforts have been made by the HPC community to encourage the use of low-powered system-on-chip (SoC) embedded processors in large-scale HPC systems. These initiatives have successfully demonstrated the use of ARM SoCs in HPC systems, but there is still a need to analyze the viability of these systems for HPC platforms before a case can be made for Exascale computation. The major shortcomings of current ARM-HPC evaluations include a lack of detailed insights about performance levels on distributed multicore systems and performance levels for benchmarking in large-scale applications running on HPC. In this paper, we present a comprehensive evaluation of results that covers major aspects of server and HPC benchmarking for ARM-based SoCs. For the experiments, we built an unconventional cluster of ARM Cortex-A9s that is referred to as Weiser and ran single-node benchmarks (STREAM, Sysbench, and PARSEC) and multi-node scientific benchmarks (High-performance Linpack (HPL), NASA Advanced Supercomputing (NAS) Parallel Benchmark, and Gadget-2) in order to provide a baseline for performance limitations of the system. Based on the experimental results, we claim that the performance of ARM SoCs depends heavily on the memory bandwidth, network latency, application class, workload type, and support for compiler optimizations. During server-based benchmarking, we observed that when performing memory intensive benchmarks for database transactions, x86 performed 12% better for multithreaded query processing. However, ARM performed four times better for performance to power ratios for a single core and 2.6 times better on four cores. We noticed that emulated double precision floating point in Java resulted in three to four times slower performance as compared with the performance in C for CPU-bound benchmarks. Even though Intel x86 performed slightly better in computation-oriented applications, ARM showed better scalability in I/O bound applications for shared memory benchmarks. We incorporated the support for ARM in the MPJ-Express runtime and performed comparative analysis of two widely used message passing libraries. We obtained similar results for network bandwidth, large-scale application scaling, floating-point performance, and energy-efficiency for clusters in message passing evaluations (NBP and Gadget 2 with MPJ-Express and MPICH). Our findings can be used to evaluate the energy efficiency of ARM-based clusters for server workloads and scientific workloads and to provide a guideline for building energy-efficient HPC clusters. Copyright © 2015 John Wiley & Sons, Ltd.

Received 16 May 2014; Revised 20 April 2015; Accepted 15 June 2015

KEY WORDS: energy-efficiency; arm evaluation; multicore cluster; large-scale scientific application

1. INTRODUCTION

Since the appearance of the IBM Roadrunner in June 2008 (the first Petaflop/s machine), the fervor to improve the peak performance of parallel supercomputers has been maintained [1]. The Tianhe-II is a supercomputer in November 2013's TOP 500 list that boasts of 34 Petaflop/s. However, this seems far behind the goal of achieving Exaflop/s by 2018. Breaking the Exascale barrier requires us to overcome

*Correspondence to: Sangyoon Oh, Department of Computer Engineering, Ajou University, 446-749 Korea.
†E-mail: syoh@ajou.ac.kr

several challenges related to energy costs, memory costs, communications costs, and others. The approach of increasing processor clock speeds to achieve higher peak performances seems to have reached a saturation point. Therefore, it is clear that Exascale system designs should employ alternate approaches. An energy efficiency of approximately 50 GFLOPS/W is required in order to meet the community goal of creating a 20 MW Exascale system [2].

Modern parallel supercomputers are dominated by power hungry commodity machines that are capable of performing billions of floating point operations per second. The primary design goal of these systems, thus far, has been high floating-point performance. Energy efficiency, on the other hand, has been a secondary objective. However, with the increasing cost of cooling these supercomputers, there is a growing concern about the power consumption of future Exascale systems. Considerations related to energy usage and related expenses have led to the development of energy efficient infrastructures. Therefore, it is unanimously agreed that energy efficiency is a major design challenge on the road to Exascale computing.

Energy-efficient techniques have been employed in both high performance computing (HPC) and general-purpose computing. Modern x86 processors employ dynamic voltage and frequency scaling (DVFS) techniques that alter the processor clock at runtime based on processor usage [3]. Low-powered embedded processors, mainly from ARM Holdings [4], have been on the market and are designed to meet the growing demands for handheld devices in mobile industries. The primary design goal for embedded processors has been low power consumption because of their use in battery-powered devices. Owing to the advantages of ARM processors in terms of performance and energy, some researchers have argued that HPC systems must borrow concepts from embedded systems in order to achieve Exascale performance [5]. The increasing multicore density, memory bandwidth, and newly arrived 64-bit Cortex-A53 processor have made ARM processors comparable with x86 processors. Thus, it is important to address questions about whether these ARM-based processors can be used to replace commodity x86 processors in the same way that Reduced Instruction Set Computer (RISC) processors replaced Vector processors more than a decade ago [6].

In order to address this question, some researchers proposed low-powered ARM SoC-based cluster designs that are referred to as Tibidabo [7] and performed initial evaluations and comparisons with x86 processors using microbenchmark kernels [8, 9]. Another group of researchers evaluated ARM SoCs using server benchmarks for in-memory databases, video transcoding, and Web server throughput [10]. The initial success of their efforts provided strong motivation for further research. However, their evaluation methodologies did not cover vital aspects of benchmarking HPC applications as suggested by Bhatele *et al.* [11]. Because modern supercomputers use distributed memory clusters consisting of shared memory multiprocessors (SMPs) (also known as multicore processors), a systematic evaluation methodology should cover large-scale application classes and provide insights about the performance of these applications in multicore and cluster-based programming models. The *de facto* programming models for these two systems are message passing interface (MPI) for distributed memory clusters and multithreaded SMP programming for multicore systems. We argue that ARM SoC-based HPC systems must exhibit high levels of performance for representative benchmarks in terms of floating point and energy-efficiency on shared memory and distributed memory.

There have been several evaluation studies related to the feasibility of ARM SoCs for HPC, and these studies are discussed briefly in Section 2. Efforts so far have focused mainly on single-node performance using microbenchmarks. There have been a few exceptions that included multi-node cluster performance. However, these efforts have not used large-scale application classes for their evaluations even though these classes are a vital aspect of future Exascale computing [11]. In this paper, we bridge this gap by providing a systematic evaluation of multicore ARM SoCs based cluster that covers major aspects of HPC benchmarking. Our evaluation methodology includes benchmarks that are widely accepted and represent large-scale applications running on parallel supercomputers. We provide insights about the comparative performances of these benchmarks under C-based and Java-based parallel programming models (i.e., MPICH (Chameleon MPI developed by William Gropp) [12] and MPJ-Express [13] for message passing C and Java, respectively). Several performance metrics are evaluated, and optimizations techniques are discussed in order to achieve better performances. We used a single quadcore ARM Cortex-A9-based SoC for multicore benchmarking and our 16-node Weiser cluster for benchmarking clusters of multicore

DOI: 10.1002/cpe

SoCs. The single node performance was also compared with an Intel x86 server in terms of performance and energy efficiency. Our evaluation methodology helps readers to understand the advantages in terms of power consumption and performance trade-offs for running scientific applications on ARM-based systems in multicore and cluster configurations.

The main contributions of this paper are summarized as follows:

- We design a systematic and reproducible evaluation methodology for covering vital aspects of single-node and multi-node HPC benchmarking for ARM SoCs. We also discuss floating-point performance, scalability, trade-offs between communications and computations, and energy efficiency.
- For single-node benchmarking, we provide insights about the database server performance and shared memory performance. We also discuss the memory bandwidth bottleneck. During shared memory tests, we analyze the reason why ARMs suffer during CPU-bound workloads, but show better speedup during I/O bound tests.
- We provide insights about the various optimization techniques for achieving maximum floating-point performance on ARM Cortex-A9 by analyzing the multi-node cluster benchmarking results from High-performance Linpack (HPL) benchmarks. We utilized a NEON single instruction multiple data (SIMD) floating point unit on an ARM processor along with compiler tuned optimization and achieved 2.5 times increased floating point performance for HPL as compared with an unoptimized run.
- We incorporated the support for ARM in MPJ-Express (a Java binding for MPI) runtime and performed comparative analysis with MPICH using NAS Parallel Benchmark (NPB) and Gadget-2 cluster formation simulation. We analyzed the scalability of large-scale scientific simulations on an ARM-based cluster.

The rest of the paper is organized as follows. Section 2 discusses related studies. Section 3 provides an overview of the benchmark applications and discusses the motivations behind their use. Section 4 describes our experimental design, hardware, and software setup. Section 5 discusses the motivations behind the use of Java HPC evaluations for ARM. Section 6 discusses the experiments that we performed and the results. It presents the analysis and the insights that we gained from the evaluation of the results. Section 7 concludes our study and sheds light on possibilities for future studies.

2. RELATED STUDIES

Modern computers have undergone drastic improvements in performance and power efficiency. DVFS has been one of the most widely accepted techniques for reducing power dissipation. It uses hardware characteristics to lower the supply voltage and the operating frequency of the processor. DVFS algorithms are able to save considerable amounts of energy in general-purpose computers [3].

The growing concerns about the energy efficiency of supercomputers on the TOP500 list [14] gave rise to the Green500 list [15], which is designed to raise awareness about performance metrics other than speed. The Green500 list focuses on performance-per-watt and is considered a major milestone for energy-efficient supercomputing. The IBM Blue Gene/P topped the first Green500 list in November 2007 with a total power consumption of 31.10 KW and a peak performance of 357.23 MFLOPS/W. The Green HPC efforts continued, and Balaji *et al.* proposed the Green Index (TGI) [16], which is a metric for evaluating the system-wide energy efficiency of HPC systems. They proposed a methodology for computing TGI and evaluated the system-wide energy efficiency using TGI. The power consumption, so far, has become a central question in research about HPC systems.

Bhatele *et al.* [11] presented a feasibility study for three application classes in order to formulate the constraints for achieving a sustained performance of 1 Exaflop/s. Their work is important in terms of providing the possible application classes (molecular dynamics, cosmological N-body simulations, etc.) for future Exascale systems. This paper strengthens the motivations for the applications and benchmarks that we used to evaluate ARM for HPC. He *et al.* [17] also provided an evaluation methodology for running HPC applications in the Cloud. In order to evaluate the HPC-in-cloud, they used existing benchmarks and a large-scale NASA climate application. The existing

DOI: 10.1002/cpe

benchmarks they used included HPL and NAS. This supports the argument for using large-scale applications for HPC benchmarking. We have also used benchmarks that have been widely accepted in the HPC community.

Fast Array of Wimpy Nodes (FAWN) [18] is a cluster architecture consisting of embedded CPUs and local flash storage for balancing computations and I/O for low-power data-intensive applications. FAWN-KV is a highly available, consistent, reliable, and high-performance storage system that is capable of handling 350 key-value queries per joule. This reference justifies the motivation for evaluating embedded processors for large-scale HPCs. Vasudevan *et al.* [19] presented the architecture and motivation for a cluster-based, many-core computing architecture for energy-efficient, data-intensive computing.

Rajovic *et al.* presented Tibidabo [7], which was a first attempt to build a large-scale HPC cluster using ARM-based NVidia Tegra SoCs. They evaluated the performance and energy efficiency of a single node with a commodity Intel® Core™ i7 processor. In their study [7–9], they tried to encourage the use of ARM-based processors for large-scale HPC systems. This paper provided an early evaluation of ARM clusters and used Linpack to measure the scalability of ARM clusters. Furthermore, it simulated the behavior and performance levels of future Cortex-15 ARM processors and compared their energy efficiency with BlueGene/Q. We not only performed a wide variety of experiments in order to cover different aspects of HPC but also extended the scope by including Java-based HPC and server benchmarking.

Bodgan *et al.* provided a trace-driven analytical model to understand the performance of server workloads on ARM Cortex-A9 based systems [20]. They modeled the degree of CPU core, memory, and I/O and estimated the clock frequency to achieve energy-efficient performance without compromising execution time. Emily *et al.* tried to address the question whether ISA plays an intrinsic role in performance or energy efficiency [21]. They analyzed measurements on ARM Cortex-A8, Cortex-A9, and Intel Atom and SandyBridge using mobile, desktop, and server workloads. They concluded that there is nothing fundamentally more energy efficient, and ISA being RISC or CISC is irrelevant. Edson *et al.* analyzed the time-to-solution and energy-to-solution comparison of ARM and Intel Xeon and found that Xeon has still a better tradeoff from user's point-of-view [22]. Rajovic *et al.* performed initial evaluation of NVIDIA Tegra 2 and Tegra 3 mobile SoCs and NVIDIA Quadro 1000 M GPU under HPC microbenchmarks to evaluate their potential for energy-efficiency [23].

Furlinger *et al.* [24] analyzed the energy efficiency of parallel and distributed computing commodity devices. They compared the performance and energy consumption of an AppleTV cluster using an ARM Cortex A8 processor. They also evaluated the performance of an AppleTV and a BeagleBoard ARM SoC development board. Stanley *et al.* [25] analyzed the thermal constraints on low power consumption processors. They established a connection between energy consumption and the processor architecture (e.g., ARM, Power, and Intel Atom). They observed that the ARM platform consumed less energy and was more efficient with light-weight workloads. However, the Intel platform consumed more energy and had the best energy efficiency for heavy workloads.

Ou *et al.* [10] studied the feasibility of ARM processors for general-purpose computing and ARM clusters for data centers. They performed an analysis of ARM development boards in Intel x86 workstations for computationally lightweight applications (i.e., in-memory database and network-bound web applications). Their cost model puts ARM at an advantage over x86 servers for lightweight applications. However, this advantage diminishes for heavy applications. They employed DVFS techniques on x86 platforms in order to reduce performance levels for comparisons with ARM. However, we argue that in a production environment, a fair comparison would require maximum speeds on both platforms. They provided a good comparison for server-based benchmarking. However, they did not focus on HPC and cluster computing applications. HPC (shared-memory and cluster benchmarking) and the optimization techniques related to floating point performance were not included in their paper. They focused on general-purpose server benchmarking, but omitted the details for cluster computing applications.

Edson *et al.* compared the execution times, power consumption, and maximum instantaneous power between two clusters based on a Cortex-A9 PandaBoard and a Cortex-A8 BeagleBoard [26]. Keville *et al.* [27] also performed early attempts for ARM benchmarking. They evaluated ARM-based

DOI: 10.1002/cpe

clusters for the NAS benchmark and used emulation techniques to deploy and test VM in the cloud. Although, they attempted to evaluate ARM emulations of VM in the cloud, an evaluation of real-time VM support for ARM was not provided. Without this type of evaluation, it is not possible to provide a real time analysis of application performance.

Jarus *et al.* [28] provided a comparison of performance and energy for two types of processors from different vendors using (1) processors that were energy-efficient, but limited for performance and (2) high-performance processors that were not as energy efficient. This study is unique because it is based on current research about energy efficiency, and it not only covers embedded RISC processors but also includes CISC x86-based processors, such as Intel Atom, that incorporate low-power techniques. This comparison provides insights about different processors that are available from different vendors. This paper is helpful, but it covers different aspects (i.e., comparing different vendors for energy usage and performance). However, our goal is to focus on a few vendors and provide a comprehensive evaluation of HPC benchmarking using application kernels that are widely accepted in the HPC community [11,9].

Up to this point, ARM SoC evaluations have focused mainly on single-node evaluations and server evaluations using microbenchmarks like Coremark, FHourstones, Whetstone, web-server tests, and OSU Microbenchmarks. Some efforts have also been made to perform multi-node cluster evaluations using Linpack. However, so far, these studies have fallen short of covering the impact of different application classes on ARM-based clusters that are employing shared-memory and distributed-memory programming models.

3. EVALUATION METHODOLOGY

We devised our evaluation methodology, which covers the vital aspects of shared-memory and distributed-memory benchmarking, in order to provide a systematic view of the analysis of ARM SoC-based systems for single-nodes and multi-node configurations. The benchmarks represent application classes that are representative of large-scale computing platforms (e.g., molecular dynamics, *n*-body simulations, and others). In this section, we discuss benchmark applications and their purposes in detail.

Memory bandwidth is the key benchmark that provides a basis for understanding the performance limitations of the systems under test. We used the STREAM benchmark to measure the memory bandwidth for ARM Cortex-A9 and Intel x86 in the single node tests. We also compared the memory bandwidth performances for C and Java-based versions of STREAM on ARM Cortex-A9. STREAM is a widely used benchmark that uses simple vector kernels to measure the memory bandwidth (in MB/s) and the corresponding computational performance levels of the systems under test. Three out of the four kernels (i.e., *Triad*, *Sum*, and *Scale*) perform arithmetic operations and the other kernel (i.e., *Copy*) counts the read and written bytes. Our interest was mainly focused on *Triad* because *Multiply* and *Accumulate* are the most widely used computations in scientific computing. The *Triad* kernel scales a vector, adds it to another vector, and stores the result in a third vector.

The database server performance levels for ARM Cortex-A9 and Intel x86 were evaluated using the Sysbench MySQL OLTP (OnLine Transaction Processing) test [29]. Using this test, we provided apple-to-apple comparisons of Intel x86 and ARM Cortex-A9 in terms of performance for query processing and energy efficiency. We created a large table with a size of one million rows in MySQL and used INSERT and SELECT for test queries. The performance metrics used include transactions per second and transactions per second per Watt.

We used the PARSEC shared memory benchmark to evaluate the multithreaded performance levels of ARM Cortex-A9 and Intel x86 servers. The PARSEC benchmark is composed of multithreaded programs and focuses on emerging workloads. It is designed to be the representative of the next generation of shared memory programs for SMPs. The workloads are diverse in nature and are chosen from different areas such as computational fluid dynamics and computational finance. Two applications of PARSEC benchmarking, namely Black-Scholes and Fluidanimate, were evaluated for strong scaling tests. Among many applications in PARSEC, we choose Black-Scholes representing EP (data-parallelism) application class and Fluidanimate representing dense linear algebra (matrix-vector) computation application class. Because a majority of the scientific application kernels involve either of these parallel designs, providing a detailed analysis of these

DOI: 10.1002/cpe

representative applications, we can assume a similar performance on other scientific application involving these parallel design approaches.

Because we are using message passing libraries to evaluate distributed memory benchmarks, the network performance is a critical factor that needs to be considered here. We performed bandwidth and latency tests on our Weiser cluster using C-based and Java-based message passing libraries (MPICH and MPJ-Express) and provided a baseline for the performance trade-offs and limitations. The distributed memory cluster benchmark contains three major HPC benchmarks: HPL, Gadget-2, and the NPB [30]. HPL is currently a *de facto* standard for benchmarking large-scale compute clusters. The TOP 500 list of supercomputers uses the HPL score to rank the world's fastest supercomputers. The benchmark kernels solve problems in random dense linear systems at double precision using 64-bit arithmetic on distributed-memory systems. They use generic implementations of MPI [12] for message passing and BLAS [31] libraries for linear algebra operations. We evaluated the performance of our ARM-based cluster for C-based and FORTRAN-based executions of HPL. The main purpose of this benchmark is to show the performance levels for ARM SoC-based clusters under BLAS libraries for 64-bit arithmetic.

Gadget-2 [32] is a massively parallel structure formation code for N-body hydrodynamic simulations. It simulates the evolution of very large cosmological systems under the influence of gravitational and hydrodynamic forces. It models the universe using a sufficiently large number of test particles that represent ordinary matter or dark matter. Gadget-2 is a perfect application for I/O and communication tests because of the finer granularity of communications that is involved. (Table I)

NASA advanced supercomputing (NAS) parallel benchmark is a benchmark suite that consists of scientific kernels derived from computational fluid dynamics (CFD) applications [30]. These kernels measure the performance for computational aspects like integer or floating-point arithmetic and complex matrix operations, and for communication aspects like unstructured adaptive meshes, parallel I/O, and irregular latencies between processors. We used four different kernels, namely, CG, EP, Fourier transform (FT), and integer sort (IS), for evaluation purposes. Each of these kernels represents a distinctive application class and was used in a wide variety of large-scale scientific applications like oil reservoir simulations and particle-based simulations.

4. EXPERIMENTAL DESIGN AND CONFIGURATION

This section presents the summary of our experimental design and testbed configurations. This mainly covers the software, hardware, and energy measurement setup. It is recommended to follow these configurations in order to reproduce the results.

4.1. Software configuration

Each node of the Weiser cluster ran on Ubuntu Linaro 12.04 with a kernel version of 3.6.0. For all of our experiments and configurations, we turned off CPU throttling and used *performance* CPUfreq governor. In this setting, we were able to utilize highest possible CPU clock on both ARM as well

Table I. Summary of evaluation benchmarks used for each platform.

			Platform	
Configuration	Benchmark	Application class	C	Java
Single node	STREAM	System bandwidth	✓	✓
	PARSEC	Fluid-dynamics	✓	✗
	Sysbench	OLTP transactions	✓	✗
Cluster	HPL	Linear Algebra	✓	✗
	NPB	HPC Kernels	✓	✓
	Gadget-2	*n*-body cosmological simulation	✓	✓

HPL, High-performance Linpack; NPB, NAS Parallel Benchmark; HPC, high-performance computing.

DOI: 10.1002/cpe

as Intel testbeds. The benchmarks and their supported platform are illustrated in Table II. We used MPICH 3.0.2 and MPJ-Express 0.38 as our message passing libraries. In order to evaluate the memory bandwidth of a single node, we used STREAM benchmark v5.9. PARSEC 3.0 [33, 34] with all sets of workloads ranging from *small* to *native*. The Sysbench benchmark [29] was also installed with MySQL server 5.1 in order to execute the database transaction test. For distributed memory benchmarking, we used HPL [35], Gadget-2 [32], and two versions of NPB, one with MPICH (NPB-MPI) and the other with MPJ-Express (NPB-MPJ) [36].

4.2. Hardware configuration

4.2.1. Single node configuration. We used Hardkernel's ODROID-X SoC development board with an ARM Cortex A-9 processor. The Cortex-A9 application processor is based on ARM v7 architecture, which has efficient power management for superscalar instructions and features NEON technology. NEON executes SIMD instructions in parallel using advanced branch predictions [26]. The CPU contains 1 MB of L2 cache and operates at a clock frequency of 1.4 GHz. The x86 machine that was chosen for the multicore benchmarking was an HP server with a quadcore 32 nm Intel® Xeon® Processor X3430 (1 MB L2 Cache, 8 MB L3 Cache, 2.40 GHz). The server also contains 16 GB of DDR3 1333 MHz RAM and 1 Gbps network adapter. The specification details for the ODROID-X development board are given in Table II.

4.2.2. Cluster configuration. The Weiser cluster consists of 16 quadcore ODROID-X boards connected through an ethernet switch. An x86 machine, used as a monitoring station, is connected to the switch. A power meter is attached to the monitoring station in order to measure the energy consumption during the benchmark execution. An external storage of 1 TB is mounted as an Network File System (NFS) drive for the shared storage.

4.3. Energy measurement

The objective metric for energy efficiency is *performance per watt*. Green500 describes the general method for measuring the *performance per watt* of a computer cluster [37]. Based on their approach, the Linpack benchmark is executed on a supercomputer with $N-nodes$, and the power of a single computer is measured by attaching a power meter to one of the nodes. The value that is obtained is then multiplied by the total number of nodes N in order to obtain the total power consumption. The primary assumption is that the workload during the execution remains well balanced between nodes. The formula that is used by Green500 is shown in Equation 1.

$$\overline{P}\ (R_{\max}) \ = \ N \ * \ P_{unit} \ * \ (R_{\max}) \tag{1}$$

R max is the maximum performance measured in GFLOPS when running the Linpack benchmark. $\overline{P}(R_{\max})$ is the average system power consumption (in watts) during the execution of Linpack with a problem size that delivers $R_{\max}$.

Table II. Configurations of single ARM SoC board and Intel x86 server.

	ARM SoC	Intel server
Processor	Samsung Exynos 4412	Intel Xeon x3430
Lithography	32 nm	32 nm
L1d/L1i/L2/L3	32 K/32 K/1 M/None	32 K/32 K/1 M/8 M
No. of cores	4	4
Clock speed	1.4 GHz	2.40 GHz
Instruction set	32-bit	64-bit
Main memory	1GB DDR2 @ 800 MHz	16 GB DDR3 @ 1333 MHz
Kernel version	3.6.1	3.6.1
Compiler	GCC 4.6.3	GCC 4.6.3

SoC, system-on-chip.

DOI: 10.1002/cpe

In order to measure the power drawn out of the Weiser cluster, we followed a Green500 approach that is shown in Equation 1. A power meter was installed between the power supply unit and a single node of the cluster, and the power meter was connected to a monitoring station via serial port. The monitoring station was an x86 Linux computer. We used ADPower's Wattman HPM-100 smart power analyzer to measure the power consumption. The accuracy of our power meter is ±0.4%. Measuring current range is AC 100 μA ~ 10 A (±0.4%) and measuring voltage range is AC 90 V ~ 250 V (±0.3%). Figure 1 shows our power measurement setup.

5. MOTIVATION FOR MESSAGE PASSING JAVA ON ARM

Several studies have been performed in order to evaluate the performance of ARM using MPI, but HPC Java on ARM has long been neglected. We took this as an opportunity and included MPJ-Express in our evaluation. This section discusses the MPJ-Express library and our efforts to add support for ARM architecture in MPJ-Express software. We also discuss the initial tests that included latency and effective bandwidth comparisons between MPJ-Express and MPICH. It is important to discuss these comparisons before jumping into the actual evaluation of applications and the discussion of results.

MPJ-Express is a Java message passing system and binding for the MPI standard that provides thread-safe communications [13]. We included message passing Java-based benchmarks in our cluster evaluation because of the increasing popularity of Java as a mainstream programming language for HPC [38]. The advantages of the Java language include platform portability, object-oriented higher-level language constructs, enhanced compile time and runtime checking, faster debugging and error checking, and automatic garbage collection [39]. Because of these highly attractive features and the availability of scientific software, Java is gaining in prominence in the HPC arena. To the best of our knowledge, at the time of writing this paper, no one has evaluated ARM SoC clusters for Java-based HPC, particularly MPJ-Express. We saw this as an opportunity and evaluated the performance of MPJ-Express using the large-scale, scientific Gadget-2 application, and NPB kernels on our ARM based SoC cluster. We not only measured the performance and scalability of MPJ-Express on ARM but also performed a comparison with MPICH, a widely used C implementation of MPI. The detailed results are explained in Section 6.

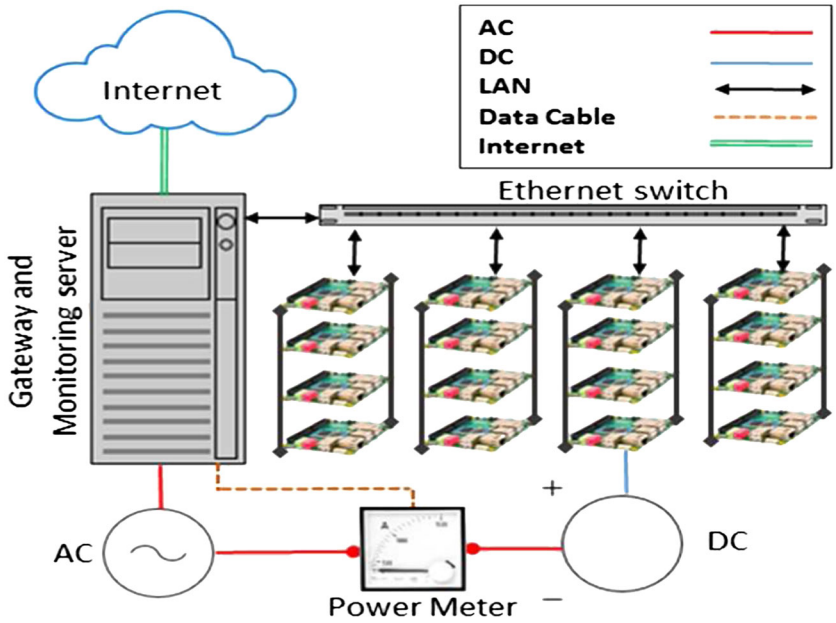


Figure 1. Weiser cluster setup.

DOI: 10.1002/cpe

The current version of MPJ-Express does not support execution on ARM architecture for cluster configurations. To proceed with our evaluation, we first incorporated the support for ARM during the MPJ-Express runtime. The MPJ-Express runtime provides the MPJdaemon and MPJrun modules for starting Java processes on computing nodes. The daemon module is a Java application that executes on computing nodes and listens to an IP port for requests for MPJ-Express processes. After receiving a request from an MPJrun module that acts as a client to the daemon module, the daemon launches a new JVM and starts the MPJ-Express processes. MPJ-Express uses the Java Service Wrapper Project [40] to install daemons as a native OS service [39]. The software distribution for MPJ-Express contains MPJboot and MPJhalt scripts that can be used to start and stop daemon services on computing nodes. In order to add support for ARM, we modified the MPJboot and MPJhalt scripts in the original MPJ-Express code and added new ARM specific floating-point binaries and Java Service Wrapper scripts to the MPJ-Express runtime.

6. RESULTS AND DISCUSSION

This section presents our findings and analysis based on the experiments that we conducted during the evaluation. The experiments did not include disk I/O during the test, and the datasets that we used were stored in the main memory. We started with the single node evaluation and later extended it to a cluster evaluation. The measurement methods that were followed during the evaluation are described in Section 4. Each experiment was repeated three times, and the average measurement was recorded.

6.1. Single node multicore evaluation

These sections present our findings from comparisons of an ARM SoC board and an Intel x86 server based on server and scientific benchmarks.

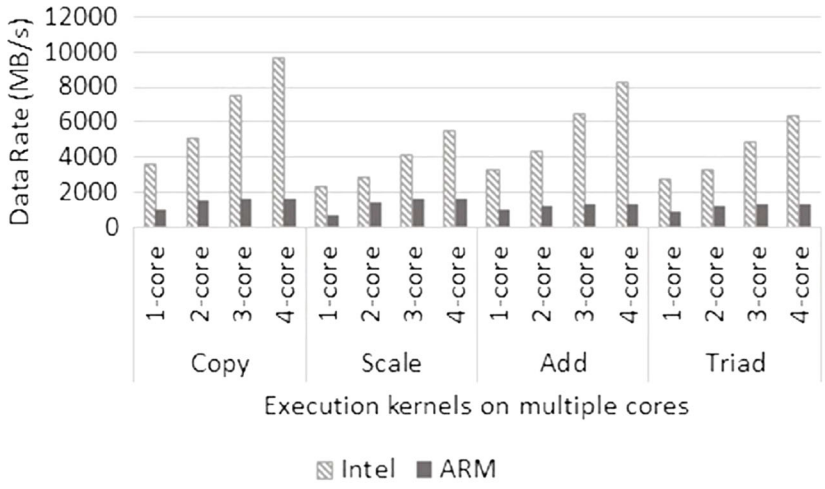
6.1.1. Memory bandwidth. Memory bandwidth plays a crucial role in evaluations of memory intensive applications. It provides a baseline for other scientific benchmarks that involve shared memory communications and synchronizations primitives. The memory management mechanisms in different mainstream languages for HPC (i.e., C and Java) have a significant impact on the performance of scientific applications that are written in these languages. Hence, it is important to provide a baseline for other benchmarks by evaluating the memory latency performance of C and Java on the target platforms (e.g., ARM and x86).

We used two implementations of the STREAM benchmark (STREAM-C and STREAM-J) and measured memory bandwidth for Intel x86 and ARM Cortex A-9. We kept the size of the input arrays at 512 K with 8-byte double elements for optimal cache utilization in the 1 MB cache for the ARM SoC board. It is also important to note is that we used OpenMP compile with one thread for single-threaded execution of the STREAM-C. The reason for this alteration is that many compilers traditionally generate much less aggressive code when compiling for OpenMP threading than when compiling for a single thread. In order to observe the multicore scalability, we always compiled STREAM-C with OpenMP and used #OMP NUM THREAD = 1 for the serial run.

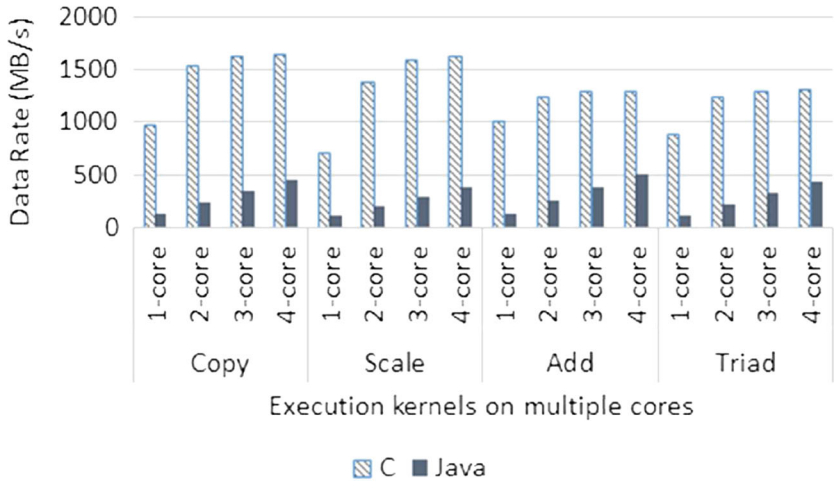
In the first part of this experiment, we compared the STREAM-C performance of ARM Cortex-A9 with the STREAM-C performance of Intel x86 commodity server. Figure 2(a) shows the results from running four STREAM kernels (Copy, Scale, Add, Triad) on ARM Cortex-A9 and Intel x86 server. The graph shows that for single threaded and multithreaded bandwidth comparisons, Intel outperformed ARM in all of the kernel executions by a factor of 3 to 4, and it also scaled well for multiple cores. The reason for the poor performances of ARM in comparison with Intel x86 is the limited bus speed (800 MHz) and the memory controller. An increase in STREAM performance is possible if STREAM is modified to run for single precision.

In the second phase of the bandwidth comparison, we compared the performance of STREAM-C and STREAM-J on ARM Cortex-A9. This comparison helped us to understand the impact of language specific memory management on the performance of shared memory applications. These results will be used as a reference when we proceed with our C-based and Java-based HPC evaluations in later sections. Figure 2(b) shows that STREAM-C performs four to five times better

DOI: 10.1002/cpe



(a)



(b)

Figure 2. STREAM benchmark evaluations on an ARM Cortex-A9 and an Intel Xeon x86 server. The first figure shows a comparison between the two architectures. The second figure shows the performance comparisons for the C and Java-based versions of STREAM on ARM Cortex-A9. (a) STREAM kernels comparison of Intel and ARM and (b) STREAM kernels C and Java on ARM.

on one core and two to three times better on four cores. One of the reasons for this huge performance difference in memory bandwidth is that the current version of JVM does not include a double precision floating point optimization for ARM Cortex-A9. The soft float application binary interface (ABIs) were used to emulate the double precision floating point performance, and this caused performance drops in performance during double precision benchmark executions. The performance differences between STREAM-C and STRAM-J on ARM should be kept in mind when analyzing the performances for shared memory benchmarks in later sections.

6.1.2. Database operations. In order to evaluate server workload performance, we used MySQL database to create a test table with one million rows of data in a single user environment. We measured the raw performance in terms of *transactions per second*, and energy-efficiency in terms of *transactions per second per watt*. The first graph in Figure 3(a) represents the raw performance comparison in terms of total OLTP transactions performed in 1 s. It can be observed that the Cortex-A9 manages to achieve approximately 400 transactions per second. We also observed that the Cortex-A9 showed better performances when transitioning from serial to multithreaded execution

DOI: 10.1002/cpe

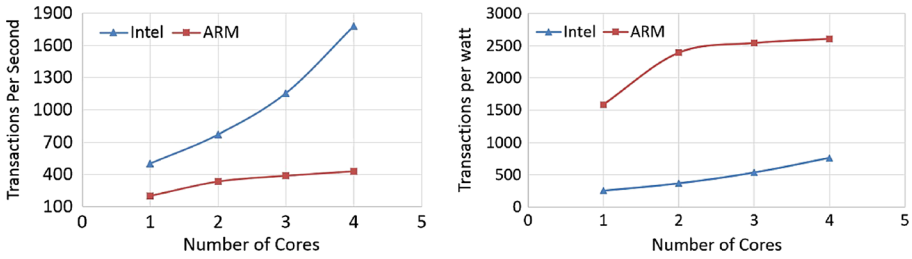


Figure 3. Comparisons of transactions/second and transactions/Watt for Intel x86 and ARM Cortex-A9 on multiple cores using the MySQL server OLTP benchmark. (a) Trans. per second test as raw performance and (b) performance per watt using average power.

(40% increase), but it did not scale well as the number of threads increased (14% on tri-cores and 10% on quad-cores). We found that the increasing number of cores also increased the cache miss rate during multithreaded executions. The small size of the cache affected the data locality because the block fetches from main memory occurred frequently and degraded the performance as the bus speed, as shown earlier, was a huge bottleneck. Clearly, Intel x86 has a significant performance edge (60% for serial and 230% on quad cores) over ARM Cortex-A9 because of its increased cache size, which accommodates larger blocks, exploits spatial data locality, and limits bus access.

The second graph that appears in Figure 3(b), displays the raw performance from an energy efficiency perspective. The *y*-axis in graph represents the total number of transactions when divided by average power consumed during the simulation on multiple threads. The *y*-axis in graph represents the total number of transactions when divided by average power consumed during the simulation on multiple threads. On four threads, Intel x86 consumed an average of 139.69 W with a standard deviation of 13.8, while ARM Cortex-A9 consumed 5.94 W with a standard deviation of 0.46. The figure shows that the serial execution of the benchmark is around seven times more energy efficient for Cortex-A9 than for Intel x86. However, as the number of cores increases the ARM energy efficiency advantage seems to diminish slowly. There is a burst increase in energy efficiency during the transition from single to dual core (about 50%), but the increase in efficiency slows down as the number of cores continues to increase. Although, the Cortex-A9 remained ahead of x86 for all executions, we had expected a linear growth in energy efficiency for ARM as we had found with Intel. We found out that bus speed impacts the energy efficiency as well. Slower bus speeds can waste CPU cycles during block fetches from memory, prolong the execution time for simulations, and increase the power consumption. Nonetheless, ARM Cortex-A9 showed significant advantages over Intel x86 during *performance per watt* comparisons.

6.1.3. PARSEC Shared memory benchmark. We used the PARSEC shared memory benchmark in order to evaluate the multithreaded performances of ARM Cortex-A9. Due to the emergence of multicore processors, modern software programs rely heavily on multithreaded libraries to exploit multiple cores [41]. As discussed in Section 3, the PARSEC benchmark is composed of multithreaded programs and focuses on emerging workloads that are diverse in nature. In this paper, we chose two EP applications that are used for physics animations for the PARSEC benchmark, namely, Black-Scholes and Fluidanimate. We used strong scaling experiments to measure the performance levels for Intel x86 and ARM Cortex-A9. Equation 2 is a derivation of Amdahl's law [42] that can be used to calculate strong scaling.

$$E_{strong} = \frac{t_1}{p*t_p} \tag{2}$$

The strong scaling test is used to find out how parallel overhead behaves as we increase the number of processors. In strong scaling experiments, the size of the problem was kept constant while the number of threads was increased during multiple runs. We used the largest *native* datasets for

DOI: 10.1002/cpe

Fluidanimate and Black-Scholes that were provided by the PARSEC benchmark. The datasets consisted of 10 million options for Black-Scholes and 500,000 particles for Fluidanimate. The strong scaling efficiency used Amdahl's law of speedup as described in Equation 2. The efficiency graph shows us how close the speedup is to the ideal speedup for n cores on a given architecture. If the speedup is ideal then the efficiency is 1, no matter how many cores are being used. The timing graphs show the execution time performances for the benchmarks for both systems.

The graph in Figure 4(a) shows the efficiency comparison between ARM Cortex-A9 and Intel x86 servers running the *native* dataset from the Black-Scholes application. As discussed earlier, Black-Scholes computes the prices of options using Black-Scholes Option Pricing Formula. The application itself is EP in nature, which means that communications occur only at the start and end of the simulation. The reduced communication also lessens the burden of synchronization between multiple threads. We observes that in an EP application like Black-Scholes, the efficiency of Intel x86 on two threads even surpasses the efficiency for one thread, which leads to superlinear speedup. The efficiency of ARM Cortex-A9 with four threads was 0.78. The efficiency of Intel x86, on the other hand, was 0.86. Although, the parallel efficiency of ARM remained 10% lower than that of Intel x86, it showed minimal diminishment in efficiency because of parallel overhead (around 9% for each additional core/thread). The ARM Cortex-A9 managed to scale well for large dataset in an EP application even though the ARM Cortex-A9 had a lower clock speed than the Intel x86. The execution times for Black-Scholes are shown in Figure 4(b). The execution times for Intel x86 were approximately 20% better for serial runs and 34% better on quad cores. This was expected because of the higher processor clock speeds and the higher level of memory bandwidth. However, the ARM Cortex-A9 also showed significant reductions in execution time (2.5 times reduction from 1 core to dual cores and 0.7 from two to four cores) as the number of cores increased. According to Aroca *et al.* [43], it is necessary to underclock the Intel x86 to 1 GHz in order to perform a fair comparison between ARM Cortex-A9 and Intel x86 during multicore testing, but we do not encourage underclocking because commodity servers operate at maximum performance in production environments (even when they are overclocked). We argue that if we are considering ARM for replacements for commodity servers, then a fair comparison must show the results from testbeds that are running at their maximum configuration.

The execution time graph for Fluidanimate is shown in Figure 4(d). We observed that the Intel x86 has three times better performance than the ARM Cortex-A9 for single and multithreaded executions.

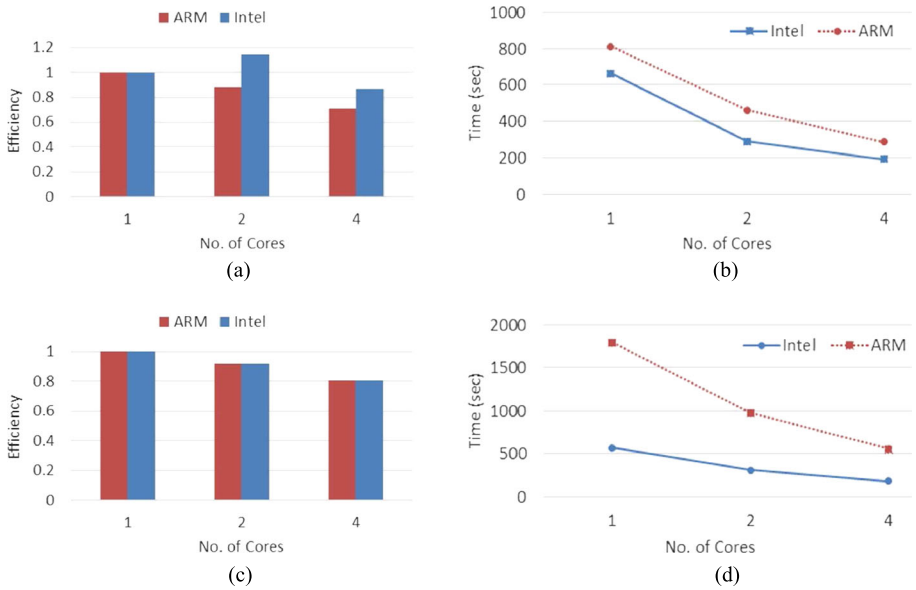


Figure 4. Black–Scholes and Fluidanimate strong scaling comparison of Intel x86 and ARM Cortex A-9. (a) Black–Scholes strong scaling efficiency graph; (b) Black–Scholes execution time graph; (c) Fluidanimate strong scaling efficiency graph; and (d) Fluidanimate execution time graph.

DOI: 10.1002/cpe

We expected this behavior because the SPH algorithm that is used in Fluidanimate contains interactions between particles in the 3D grids that are represented as dense matrix data structures. The particles are arranged in a cell data structure containing 16 or fewer particles per cell. The properties of a particle, such as force, density, and acceleration, are calculated based on the effects of other particles, and this causes an increase in overhead due to communications between threads. The simulation was run 500 frames, and in each frame, five kernels were computed [34]. The communications phases occur after each kernel execution (i.e., when boundary cells exchange information) and at the end of each frame (i.e., when rebuilding the grid for next frame). The concurrent threads working on different sections of the grid use synchronization primitives between the communication phases in order to produce correct output.

The scaling efficiency graph in Figure 4(c) shows that there was very little difference (almost negligible) between the scaling efficiencies for ARM Cortex-A9 and Intel x86. We observed that the efficiency values for x86 and Cortex-A9 were both at 0.9 for two cores and 0.80 for four cores (12.5% increase). This means that, even in highly I/O bound applications where intensity of the communications and the synchronization between threads is higher than usual, the ARM Cortex-A9 shows comparable scaling efficiency. We observed that despite being slower in absolute floating point performance because of slower processor and memory bandwidth, Cortex-A9 was able to achieve comparable performances with x86 for communication-oriented application classes. In these applications, the clock cycles of the faster x86 processor are wasted because of I/O waits which results in performance levels that are comparable with those of the Cortex-A9. Additionally, the low power consumption of the ARM Cortex-A9 gives it a substantial edge over Intel x86 because the power utilization of x86 processors increases much faster than that of Cortex-A9 as the cores are scaled.

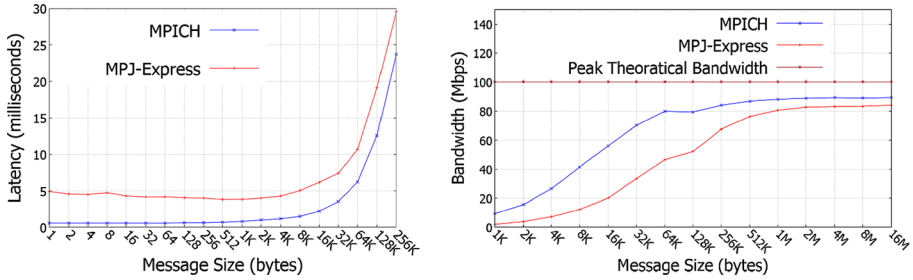
6.2. Multi-node cluster evaluation

This section presents our results for distributed memory performance for an ARM Cortex-A9-based multi-node cluster. We do not compare the ARM cluster with x86-based cluster in terms of performance or energy-efficiency. The reason is that clearly, x86-based cluster will outperform ARM, and we gain no significant insight. However, restricting our ARM vs. x86 comparison with a single node, we gain insights on multi-threaded performance and energy efficiency as previously described in Section 6.1. For the cluster part, we are interested in evaluating scientific libraries and applications on our ARM-based cluster under different performance metrics.

We started by measuring the bandwidth and latency of C-based and Java-based message passing libraries (i.e., MPICH and MPJ-Express) on our Weiser cluster. Later, we performed evaluations using HPL, NPB, and Gadget-2 cluster formation simulation.

6.2.1. Latency and memory bandwidth. The message passing libraries (MPICH and MPJ-Express) are the core for each of the distributed memory benchmarks. Thus, it is necessary to evaluate and compare the performance levels of these libraries on the Weiser cluster before starting our discussion about distributed memory benchmarks. We performed bandwidth and latency tests in order to establish baselines. We have already evaluated the intra-node bandwidth (i.e., memory bandwidth of a single SoC) in Section 6.1.1 that provides the basis for performance comparisons between Intel x86 and ARM Cortex-A9. In this section, we evaluate the network bandwidth for cluster of nodes (i.e., inter-node network bandwidth) in order to provide a basis for performance comparisons between message passing libraries (e.g., MPICH and MPJ-Express) that are running on Weiser. Figure 5(a) shows the latency comparison between MPICH and MPJ-Express on the Weiser cluster. The test began with an exchange of a 1-byte message, and at each step, the message size was doubled. We observed that there was a gradual increase in latency until the message size reached 64 KB. After that, there was a sudden increase in latency. On the other hand, the bandwidth graph in Figure 5(b) shows that the bandwidth increased gradually until the message size reached 128 KB. At 128 KB, we see that the bandwidth dropped slightly before increasing again. The reason for this behavior in both MPICH and MPJ-Express is that the messaging protocol changes when the message size reaches 128 KB. MPICH and MPJ-Express both use Eager and Rendezvous message delivery protocols. In Eager protocol, no acknowledgement is required by the receiving process, and this means that no synchronization is needed. This protocol is useful for smaller messages up to a

DOI: 10.1002/cpe



(a) (b)

Figure 5. Latency and bandwidth comparison for MPICH and MPJ-Express on a 16-node ARM Cortex-A9 cluster with fast ethernet. (a) Latency comparison MPJ-E and MPICH and (b) bandwidth comparison MPJ-E and MPICH.

certain message size. For larger messages, Rendezvous protocol is used. This protocol requires acknowledgements because no assumptions can be made regarding the buffer space that is available for the receiving process [44]. In both MPJ-Express and MPICH, the message limit for Eager protocol is set at 128 KB. When the message size reaches 128 KB, the MPI/MPJ-Express runtime switches the protocol to Rendezvous [45].

MPICH performs 60% to 80% better than MPJ-Express for smaller messages. However, as the message size increases and the protocol switch limit is surpassed, the performance gap shrinks. In Figure 5(b), we see at larger message-sized (e.g., 8 M or 16 M), the MPICH advantage has dwindled to only 8% to 10%. We found that the MPJ-Express architecture suffers in terms of performance for message passing communications. There are multiple buffering layers that the user message passes through after leaving the user buffer in order to be delivered to the destination. A newer version of MPJ-Express (i.e., v_0.41) overcomes this overhead by providing native devices that uses native MPI libraries directly for communications across the network. We conclude that MPJ-Express is expected to suffer in terms of performance for message passing benchmarks in cases where communications occur frequently and the message size is small. However, in EP applications where communications does not occur frequently or the message sizes are typically larger, the performance of MPJ-Express was comparable with MPICH on our ARM cluster. These performance trade-offs between MPICH and MPJ-Express libraries on clusters of nodes should be kept in mind when studying the results for other benchmarks.

6.2.2. High-performance Linpack. High-performance Linpack is a widely accepted benchmark that is used to evaluate to the performance levels of the world's top supercomputers that are listed in TOP500 list in terms of GFLOPS. It is also used to measure the *performance per watt* (energy-efficiency) of green supercomputers on the Green500 list. We used HPL benchmark to measure the raw performance levels and energy-efficiencies of our test platforms. The methodology for these measurements was described and explained in Section 3.

High-performance Linpack performance depends heavily on the optimization of the BLAS library. We used ATLAS (a highly optimized BLAS library) to build the HPL benchmark. In order to further enhance the performance of ATLAS, we hand tuned the compilation by adding compiler optimization flags. The GCC flags we used are -O3 –mcpu = cortex-a9 –mtune = cortex-a9 –march = arv7a -mfloat-abi = hard -mfpu = neon -funsafe-math-optimizations -funroll-loops, while further details can be found in [43,46]. In this way, we specifically instructed the compiler to use the NEON SIMD instruction set in the Cortex-A9, rather than several simpler standard RISC instructions. In order to observe the impact of the library optimizations and compiler optimizations on the floating-point performance of the ARM, we categorized our HPL experiment into three different runs based on the optimization levels of the BLAS and HPL compilation. Table III shows a comparison of the performances that were achieved for each of the optimized executions and the unoptimized one.

High-performance Linpack is a CPU bound application, and the floating-point performances depend on the CPU speed and the memory interface. In order to achieve maximum performance, we used optimal input sizes based on the memory sizes in the processors. The rule of thumb

DOI: 10.1002/cpe

Table III. Three different executions of HPL benchmarks based on BLAS and HPL software optimizations.

Execution	Optimized BLAS	Optimized HPL	Performance (comparison to Ex. 1).
1	No	No	1.0×
2	Yes	No	~1.8×
3	Yes	Yes	~2.5×

HPL, high-performance Linpack.

that is suggested by HPL developers is to keep the problem size at a level that approximates 80% of the total amount of memory system [47].

Figure 6(a) shows the results for all three executions of the HPL benchmark on our Weiser cluster. We observed that the compiler that was tuned with an ATLAS-generated BLAS library and tuned with NEON optimization (Execution 3) resulted in the best performances as compared with other HPL executions. We observed that the GFLOPS performance was 10% better for Cortex-A9 in Execution 3 when using a single core on a square matrix of size 9600 as compared with Execution 2. Execution 1, on the other hand, showed the worst GFLOPS performance. The results were similar with a higher number of processors. Execution 3 running on a Weiser cluster with 64 cores resulted in 18.5% more GFLOPS than Execution 2 and 56.3% more GFLOPS than Execution 1 for a square matrix of size 40 K. Figure 6(b) shows the performance of HPL on C and FORTRAN. The optimization that was used for HPL-FORTRAN was similar to Execution 2 for HPL-C. The HPL-C and HPL-FORTRAN executions performed equally well in terms of GFLOPS performance and showed good scalability.

We observed that Intel x86 was able to achieve 26.91 GFLOPS as compared with 24.86 GFLOPS for the Weiser. These results were expected. However, the differences in power consumption were more observable than difference in GFLOPS. Intel x86 consumed 138.7 W on average with a standard deviation of 8.02, which was almost double the 79.13-W power consumption with a standard deviation of 3.78 of the Weiser. It is important to note that the CPU utilization levels for the Intel x86 and the ARM Cortex-A9 were approximately 96% to 100%, respectively, during the HPL run. Table IV shows the *performance per watt* figures for x86 and Cortex-A9. It can be observed that the Weiser cluster achieved 321.70 MFLOPS/W as compared with 198.64 for the Intel x86. Although Intel x86 had better floating point performance because of higher processor clock speed, cache size, and faster memory I/O, it lagged behind by a substantial 38% in the MFLOPS/Watt comparison. Floating point operations are mainly CPU bound tasks. As a result, CPU speed, memory latency, and cache size are the key factors for determining the resulting performance. Even though, the ARM Cortex-A9 cluster remained about 9% below Intel x86 in terms of raw GFLOPS performance, it outperformed in terms of *performance per watt* comparison by showing a 61.9% increase in GFLOPS/Watt over that of Intel x86. Furthermore, through our evaluation of three different executions of HPL on the ARM Cortex-A9 cluster, we found that the

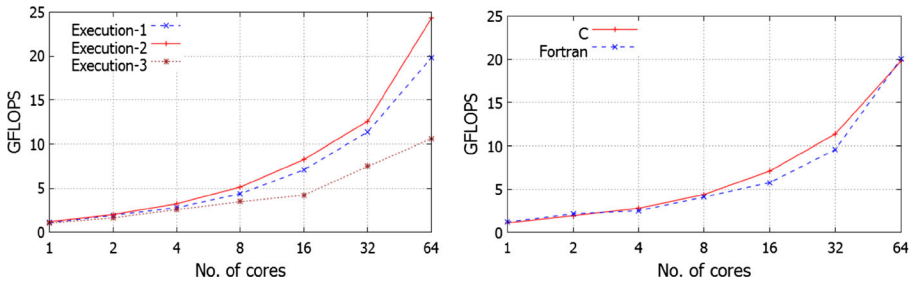


Figure 6. High-performance Linpack (HPL) performances on a 64-core ARM Cortex-A9 cluster. (a) Shows the effect of the optimization flags on the BLAS performances and (b) shows a comparison between C and FORTRAN executions for HPL. (a) HPL C using three different optimizations on Weiser and (b) HPL FORTRAN and C on the Weiser cluster.

DOI: 10.1002/cpe

Table IV. Energy Efficiency of Intel x86 server and Weiser cluster running HPL benchmark.

Testbed	Build	R_{max} (GFLOPS)	$\overline{P}(R_{max})$	PPW (MFLOPS/W)
Weiser	ARM Cortex – A9	24.86	79.13	321.70
Intel x86	Xeon x3430	26.91	138.72	198.64

HPL, high-performance Linpack.

compiler-based optimizations and the NEON SIMD floating-point unit (FPU) caused the level of performance to increase 2.5 times.

We conclude that the software optimizations for ARM had a significant role in achieving the best possible floating-point performances on ARM. The performance differences between ARM Cortex-A9 and Intel x86 seem to be a high barrier, but it should be kept in mind that there is a long history of community efforts related to software optimizations for x86 systems and that ARM is still developing.

6.2.3. Gadget-2. As discussed in Section 3, Gadget-2 is a massively parallel software program for cosmological simulations. It uses the *Barnes–Hut tree* algorithm or the *Particle-Mesh* algorithm to calculate gravitational forces. The domain decomposition of Gadget-2 is challenging because it is not practical to evenly divide the space. The particles change their positions during each timestep because of the effect of forces. Thus, dividing the space evenly would lead to poor load balancing. In order to solve this issue, Gadget-2 uses Peano–Hilbert space-filing as suggested by Warren and Salmon [48]. Gadget-2 is used to address a wide range of interesting astrophysical problems related to colliding galaxies, merging galaxies, and cluster formation in the universe. Considering the massively parallel, computationally intensive, and communications-oriented nature of Gadget-2, we consider it a perfect benchmark for the evaluation of our Cortex-A9 cluster. We used Gadget-2 to test scalability while running large-scale scientific software programs on our Weiser cluster. We measured the performance of the Gadget-2 simulation and used execution time and parallel scalability as the key metrics. Scalability was defined as the parallel speedup that was achieved as the number of cores increased, as suggested by Amdahl's law [42].

We used *cluster formation* simulations for Gadget-2 with 276,498 particles. We measured the execution times and levels of speedup while running the simulations on Weiser. Figure 7(a) shows the execution times for the simulations on Weiser. We observed that the execution times for ARM were twice as high as those for Intel when using one core. This behavior was expected as we have already seen in other benchmarks that ARM was no match for an x86 commodity server based on raw CPU performance because of its low clock speed, lower memory bandwidth, and smaller cache size. However, we were interested in evaluating the performance of ARM as the number of cores increased. The results of these tests give an indication of how well a large-scale scientific application like Gadget-2 will scale on ARM-based clusters. The speedup graph in Figure 7(b) shows the scalability of Weiser as the number of cores increased. This increase in speedup continued up to 32

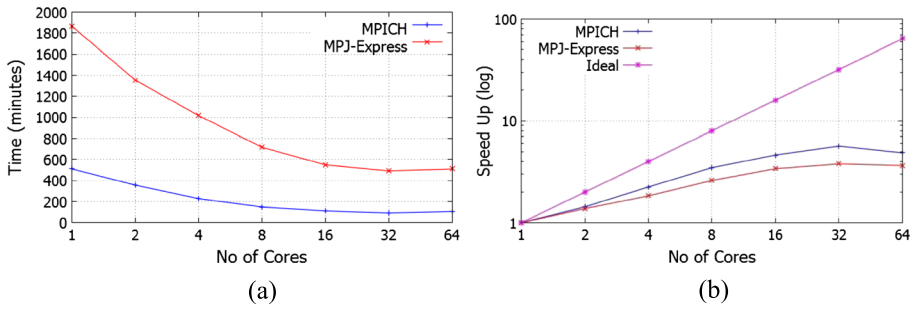


Figure 7. Execution time and speedup comparison for the Gadget-2 cluster formation simulations using MPJ-Express and MPICH. Speedup graph is log-scale on *y*-axis. (a) Execution Time graph, (b) Speedup Graph.

DOI: 10.1002/cpe

processors. After that, we observed a gradual decrease in the speedup. As discussed earlier, Gadget-2 uses communication buffers to facilitate particle exchange between different processors during the domain decomposition phase. As the number of processors increases, the communication to computation ratio also starts to increase. The problem size that we used was too small because of the memory limitations in the cluster nodes. As a result, the communications started to overcome the computations quite early (i.e., after 32 nodes). Gadget-2 should scale well on ARM in real-world scenarios, where memory levels are usually higher and networks are usually faster.

We observed that the ARM Cortex-A9 cluster displayed scalable performance levels and increases in parallel speedup while running large-scale scientific application code as the number of cores increased to 32. Therefore, due to its low power consumption, it seems that Weiser is an excellent option for scientific applications that are not time-critical.

6.2.4. NAS parallel benchmark. We performed a NPB experiment on Weiser cluster using two implementations (i.e., NPB-MPI and NPB-MPJ). The details for the NAS benchmark were discussed in Section 3. We evaluated two types of kernels: communication-intensive kernels (e.g., conjugate gradient (CG) and IS) and computationally intensive kernels (e.g., FT, EP). The workloads that we used were typically Class B workloads. However, for the memory intensive kernels like FT, we used a Class A workload because of the memory limitations of the ARM Cortex-A9 in our cluster. Each of the kernels was tested three times, and the average value was included in the final results. The performance metric that was used for all of the tests was millions of operations per second (MOPS). This metric refers to the number of kernel operations, rather than the number of CPU operations [5].

First, we evaluated the communication-intensive and memory-intensive kernels (e.g., CG and IS). Figure 8(a) shows the CG kernel results using Class B. We observed scalable performance for both implementations. However, NPB-MPI remained ahead of NPB-MPJ for each of the cores. NPB-MPJ managed to achieve only 44.16 MOPS on a Weiser cluster with 64 cores as compared with 140.02 MOPS for NPB-MPI. As a result, the execution time for NPB-MPJ was three times higher than the execution time than NPB-MPI. IS was the second benchmark in the same category. This benchmark

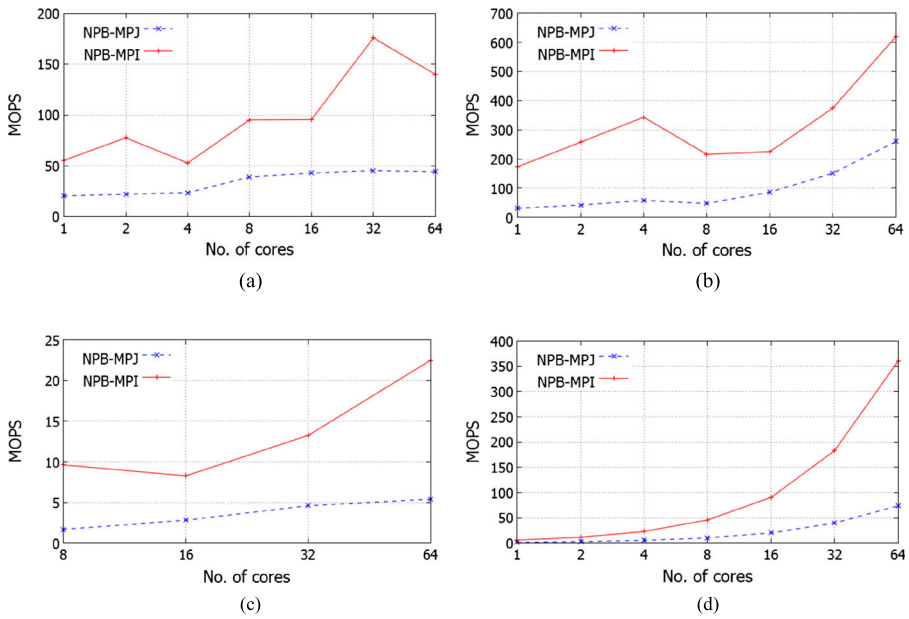


Figure 8. NAS parallel benchmark kernels performance evaluation on ARM-based cluster using two different benchmark implementations. (a) NPB-CG Kernel performance comparison of MPJ-Express and MPICH using Class B; (b) NPB-FT Kernel performance comparison of MPJ-Express and MPICH using Class A; (c) NPB-IS Kernel performance comparison of MPJ-Express and MPICH using Class B; and (d) NPB-EP Kernel performance comparison of MPJ-Express and MPICH using Class B.

DOI: 10.1002/cpe

performs sorting operations that are dominated by the random memory accesses and communication primitives, which are key processes in programs that handle particle methods. Figure 8(c) shows the test results for the Class B execution for IS. We observed that NPB-MPI achieved a maximum of 22.49 MOPS on a 64 core cluster, while NPB-MPJ achieved only 5.39 MOPS. However, the execution times for NPB-MPI dropped in a much more scalable manner than those of NPB-MPJ. It should be noted that we were not able to run this benchmark for one, two, and four cores because of memory constraints on the Weiser cluster. The communication-oriented NAS kernels, such as CG (unstructured grid computation) and IS (random memory access), are iterative problem solvers that put significant loads on the memory and the network during each iteration. Thus, the memory bandwidth and network latency are crucial to their performance levels. Because CG and IS are communication-intensive kernels, their levels of scalability depend heavily on the communication primitives in the message passing libraries. In the earlier bandwidth and latency tests, we observed that the performance of MPJ-Express was lower than that of MPICH. Slower connections (100 Mbps) were one of the reasons for slower performance of communication bound applications on multi-node clusters. However, another important reason for the differences in performance between MPJ-based and MPI-based implementations was the internal memory management of MPJ-Express. MPJ-Express explicitly manages the memory for each Send() and Recv() call by creating internal buffers [13]. The constant process of creating these buffers during each communication call for sending and receiving application data between cluster nodes results in slower performance. This overhead can be reduced by using native devices in MPJ-Express runtime and by calling native communication primitives in MPI in order to bypass MPJ-Express buffering. However, our version of MPJ-Express for ARM did not include these capabilities as of yet.

Our second evaluation included computation intensive kernels for NAS such as FT and EP. Both of these kernels displayed the upper limits of floating point arithmetic performance. The FT kernel results are shown in Figure 8(b). NPB-MPJ showed lower performances than NPB-MPI, but it showed good scalability as the number of cores increased. The scalability of NPB-MPI improved at first as the number of cores increased. However, we observed a sudden drop in scalability for NPB-MPI for 8 to 16 cores due to the congestion caused by the increasing number of cores on the network. The greater communication to computation ratio caused by smaller data sets (Class A) also caused poor performances. In this test, NPB-MPJ managed to achieve 259.92 MOPS on 64 cores as compared with 619.41 MOPS for NPB-MPI. We were only able to run FT for Class B on 16 cores or higher. Because of the memory constraints for ARM Cortex-A9 SoC, we were not able to fit larger problems in the main memory. Similarly, EP kernel due to its EP nature, scaled very well for both NPB-MPI and NPB-MPJ. Figure 8(d) shows that the performance of NPB-MPI was five times better than the performance of NPB-MPI on the Weiser cluster (360.88 MOPS as compared with 73.78). However, the performances of the MPI and MPJ versions of the CPU-intensive kernels (e.g., FT and IS) showed scalable performances because most of the computations were performed in way that was local to each node in the cluster. The absence of communication primitives resulted in scalable performances in terms of Amdahl's law of parallel speedup on Weiser [42]. We also observed that the reduced inter-node communications for EP kernels prevented MPJ-Express buffering overhead. As a result, the MPI and MPJ-based implementations both displayed better scalability with a higher number of cores for Weiser. Another reason for the comparatively better performances is that the integer and floating point arithmetic in CPU bound kernels take advantage of the NEON SIMD FPU unit in the ARM Cortex-A9, and this causes a boost in performance. The slower performances of MPJ, as compared with MPI, in FT and IS is because current JVM implementations still lack support for the NEON SIMD functionality for floating point. They rely, instead, on emulation of double precision in the FPU for ARM.

7. CONCLUSION

In this paper, we presented a comprehensive evaluation of ARM-based SoCs and covered major aspects of server and HPC benchmarking. In order to provide analysis, we conducted a wide set of measurements and covered diversified applications and benchmark tests, including single-node

DOI: 10.1002/cpe

benchmarks (e.g., memory bandwidth (STREAM), shared-memory benchmarking (PARSEC), and database transactions (Sysbench)), and multi-node cluster benchmarks (e.g., HPL, Gadget-2, and NAS).

Based on the measurement results, we found that the performance of single-node ARM SoC depends on the memory bandwidth, processor clock, application type, and that multi-node performance relies heavily on network latency, workload class, and the compiler optimizations and library optimizations that are used. During the server-based benchmarking, we observed that the multithreaded query processing performance levels for Intel x86 were 12% better than those for ARM for the memory intensive benchmarks like OLTP transactions. However, ARM performed four times better in tests of the performance to power ratio on a single core and 2.6 times better for tests on multiple cores. We also found that the emulations of double precision floating point in JVM/JRE resulted in performances that were three to four times slower (as compared with C) for Java-based benchmarks in CPU-bound applications. During the shared memory evaluations, Intel x86 showed a significant performance edge over ARM in EP applications (e.g., Black–Scholes). However, ARM displayed better efficiency for Amdahl's law scaling for I/O bound applications (e.g., Fluidanimate). Our evaluation of two widely used message passing libraries (e.g., MPICH and MPJ-Express) used NPB and Gadget-2 and revealed the impact of network bandwidth, workload type, and messaging overhead on scalability, floating point performance, the performance of large-scale applications, and energy-efficiency of the cluster. We found that, despite the slower network bandwidth as compared with commodity HPC clusters, our ARM-based cluster achieved 321 MFLOPS/W, which is just above the 222nd place supercomputer on the Green500 list for November 2013. Finally, we found that when a NEON SIMD floating point unit on the ARM processor was coupled with hand tuned compiler optimizations, the floating point performance for HPL was 2.5 times better than it was for straight forward (unoptimized) executions.

ARM processors have a niche in the mobile and embedded systems markets and are typically used in handheld devices. However, this is likely to change in the near future. From multicore evaluations of ARM Cortex-A9 SoC, we concluded that ARM processors have potential to be used as lightweight servers and they show reasonable performance levels for I/O bound shared memory benchmarks. Based on distributed memory cluster evaluations of ARM SoCs, we were able to confirm the scalability for large-scale scientific simulations and different optimization techniques for achieving the best possible performance levels. ARM-based SoCs are a reasonable answer to the growing need for energy efficiency in datacenters and in the HPC industry. However, there are still challenges related to poor software and hardware support that need to be addressed before ARM becomes mainstream.

ACKNOWLEDGEMENTS

This research is jointly supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the Global IT Talent support program (IITP-2015-R0618151003000102) supervised by the IITP (Institute for Information and Communication Technology Promotion), the ICT/SW Creative Research program (Microsoft, NIPA-2014-H0510-14-1037), under the NIPA(National IT Industry Promotion Agency) and Microsoft Research, and Microsoft Azure for Research. The authors would like to thank Professor Guillermo Lopez Taboada of Computer Architecture Group, University of A Coruna for NPB-MPJ source code and Professor Aamir Shafi of HPC Laboratory SEECS-NUST for Gadget-2 MPJ-Express source code.

REFERENCES

1. http://www.top500.org/TOP500 list, (Last visited in Aug 2013). http://www.top500.org/
2. P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, *et al.*, Exascale computing study: technology challenges in achieving exascale systems.
3. Pillai P, Shin K. Real-time dynamic voltage scaling for low-power embedded operating systems. In ACM SIGOPS Operating Systems Review, Vol. 35, ACM, 2001, pp. 89–102.
4. http://www.arm.com/products/processors/index.phpArm processors, (Last visited in Aug 2013). http://www.arm.com/products/processors/index.php
5. Jensen D, Rodrigues A. Embedded systems and exascale computing. *Computing in Science & Engineering* 2010; **12**(6):20–29.
6. Barroso L, Hölzle U. The datacenter as a computer: an introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture* Morgan & Claypool Publishers: CA, USA, 2009; **4**(1):1–108.
7. Rajovic N, Puzovic N, Ramirez A, Center B. Tibidabo: Making the Case for an arm Based HPC System.

DOI: 10.1002/cpe

8. Rajovic N, Puzovic N, Vilanova L, Villavieja C, Ramirez A. The low-power architecture approach towards exascale computing, in: Proceedings of the second workshop on Scalable algorithms for large-scale systems, ACM, 2011, pp. 1–2.
9. Rajovic N, Carpenter PM, Gelado I, Puzovic N, Ramirez A, Valero M, Supercomputing with commodity cpus: are mobile SOCS ready for HPC? In Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2013, p. 40.
10. Ou Z, Pang B, Deng Y, Nurminen J, Yla-Jaaski A, Hui P. Energy- and cost-efficiency analysis of ARM-based clusters. In Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on, IEEE, 2012, pp. 115–123.
11. Bhatele A, Jetley P, Gahvari H, Wesolowski L, Gropp WD, Kale L. Architectural constraints to attain 1 exaflop/s for three scientific application classes. In Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International, IEEE, 2011, pp. 80–91.
12. http://www.mcs.anl.gov/research/projects/mpi/Mpi home page, (Last visited in Aug 2013). http://www.mcs.anl.gov/research/projects/mpi/
13. Baker M, Carpenter B, Shafi A. Mpj express: towards thread safe java hpc, in: Cluster Computing, 2006 IEEE International Conference on, IEEE, 2006, pp. 1–10.
14. Sharma S, Hsu C, Feng W. Making a case for a green500 list. In Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, IEEE, 2006, pp. 8–pp.
15. http://www.green500.org/Green500 list, (Last visited in Oct 2013). http://www.green500.org/
16. Subramaniam B, Feng W, The green index: a metric for evaluating system-wide energy efficiency in HPC systems. In Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, IEEE, 2012, pp. 1007–1013.
17. He Q, Zhou S, Kobler B, Duffy D, McGlynn T. Case study for running HPC applications in public clouds. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, ACM, 2010, pp. 395–401.
18. Andersen D, Franklin J, Kaminsky M, Phanishayee A, Tan L, Vasudevan V. FAWN: a fast array of wimpy nodes. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, ACM, 2009, pp. 1–14.
19. Vasudevan V, Andersen D, Kaminsky M, Tan L, Franklin J, Moraru I. Energy-efficient cluster computing with FAWN: workloads and implications. In Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking, ACM, 2010, pp. 195–204.
20. Tudor BM, Teo YM. On understanding the energy consumption of arm-based multicore servers. In ACM SIGMETRICS Performance Evaluation Review, volume 41, No. 1 pp. 267-278. ACM, 2013.
21. Blem E, Menon J, Sankaralingam K. Power struggles: revisiting the RISC vs. CISC debate on contemporary AWRM and x86 architectures. In High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on, pages 1-12. IEEE, 2013.
22. Padoin EL, de Oliveira DAG, Velho P, Navaux POA. Time-to-solution and energy-to-solution: a comparison between ARM and XEON. In Applications for Multi-Core Architectures (WAMCA), 2012 Third Workshop on, pages 48-53. IEEE, 2012.
23. Rajovic N, Rico A, Vipond J, Gelado I, Puzovic N, Ramirez A. Experiences with mobile processors for energy efficient hpc. In Proceedings of the Conference on Design, Automation and Test in Europe, pages 464-468. EDA Consortium, 2013.
24. Fürlinger K, Klausecker C, Kranzlmüller D. Towards energy efficient parallel computing on consumer electronic devices. *Information and Communication on Technology for the Fight against Global Warming* 2011:1–9.
25. Stanley-Marbell P, Cabezas VC. Performance, power, and thermal analysis of low-power processors for scale-out systems. In Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW), 2011 IEEE International Symposium on, IEEE, 2011, pp. 863–870.
26. Padoin EL, Oliveira DAd, Velho P, Navaux PO, Evaluating performance and energy on arm-based clusters for high performance computing. In Parallel Processing Workshops (ICPPW), 2012 41st International Conference on, IEEE, 2012, pp. 165–172.
27. Keville KL, Garg R, Yates DJ, Arya K, Cooperman G, Towards fault-tolerant energy-efficient high performance computing in the cloud. In Cluster Computing (CLUSTER), 2012 IEEE International Conference on, IEEE, 2012, pp. 622–626.
28. Jarus M, Varrette S, Oleksiak A, Bouvry P, Performance evaluation and energy efficiency of high-density HPC platforms based on INTEL, AMD and ARM processors, in: Energy Efficiency in Large Scale Distributed Systems, Springer, 2013, pp. 182–200.
29. http://sysbench.sourceforge.net/Sysbench benchmark, (Last visited in Aug 2013). http://sysbench.sourceforge.net/
30. https://www.nas.nasa.gov/publications/npb.htmlNas parallel benchmark, (Last visited in Mar 2014). https://www.nas.nasa.gov/publications/npb.html
31. http://www.netlib.org/blas/Blas home page, (Last visited in march 2013). http://www.netlib.org/blas/
32. Springel V. The cosmological simulation code gadget-2. *Monthly Notices of the Royal Astronomical Society* 2005; **364**(4):1105–1134.
33. Bienia C. Benchmarking modern multiprocessors, PhD thesis, Princeton University (January 2011).
34. Bienia C, Kumar S, Singh JP, Li K. The parsec benchmark suite: Characterization and architectural implications, Tech. Rep. TR-811-08, Princeton University (January 2008).

DOI: 10.1002/cpe

35. http://www.netlib.org/benchmark/hpl/High performance linpack, (Last visited in Aug 2013). http://www.netlib.org/benchmark/hpl/
36. Mallon DA, Taboada GL, Touriño J, Doallo R. NPB-MPJ: NAS Parallel Benchmarks Implementation for Message-Passing in Java. In Proc. 17th Euromicro Intl. Conf. on Parallel, Distributed, and Network-Based Processing (PDP'09), Weimar, Germany, 2009, pp. 181–190.
37. Ge R, Feng X, Pyla H, Cameron K, Feng W, Power measurement tutorial for the green500 list, The Green500 List: Environmentally Responsible Supercomputing.
38. Taboada GL, Touriño J, Doallo R. Java for high performance computing: assessment of current research and practice. In Proceedings of the 7th International Conference on Principles and Practice of Programming in Java, ACM, 2009, pp. 30–39.
39. Shafi A, Carpenter B, Baker M, Hussain A. A comparative study of Java and C performance in two large-scale parallel applications. *Concurrency and Computation: Practice and Experience* 2009; **21**(15):1882–1906.
40. http://wrapper.tanukisoftware.com/doc/english/download.jspJava service wrapper, (Last visited in October 2013). http://wrapper.tanukisoftware.com/doc/english/download.jsp
41. Sodan AC, *et al.* "Parallelism via multithreaded and multicore CPUs."*Computer* 2010; **43.3**:24–32.
42. Michalove A. "Amdahls Law." *Website*: http://home. wlu. edu/whaleyt/classes/parallel/topics/amdahl. html (2006).
43. Aroca RV, Garcia Gonçalves LM. Towards green data-centers: a comparison of x86 and ARM architectures power efficiency. *Journal of Parallel and Distributed Computing*.
44. https://computing.llnl.gov/tutorialsMpi performance topics, (Last visited in October 2013). https://computing.llnl.gov/tutorials
45. http://mpj-express.org/docs/guides/windowsguide.pdfMpj guide, (Last visited in October 2013). http://mpj-express.org/docs/guides/windowsguide.pdf
46. http://gcc.gnu.org/onlinedocs/gcc/ARM-Options.htmlArm gcc flags, (Last visited in Aug 2013). http://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html
47. http://www.netlib.org/benchmark/hpl/faqs.htmlHpl problem size, (Last visited in October 2013). http://www.netlib.org/benchmark/hpl/faqs.html
48. Salmon JK, Warren MS. Skeletons from the treecode closet. *Journal of Computational Physics* 1994; **111**(1):136–155.

DOI: 10.1002/cpe