OS Support for Thread Migration and Distribution in the Fully Heterogeneous Datacenter

Pierre Olivier, Sang-Hoon Kim, Binoy Ravindran
ECE, Virginia Tech
polivier,sanghoon,binoy@vt.edu

ABSTRACT

The datacenter is becoming fully heterogeneous, integrating multiple OS-capable CPUs of different Instruction Set Architectures in separate machines. These machines present diverse performance and power consumption profiles and we show that significant potential benefits for both metrics can be expected, should these machines be able to cooperate in the processing of datacenter, multi-programmed workloads. We advocate that this cooperation should be enabled at the level of the OS, relieving the programmer from any effort related to the heterogeneity of the managed machines. We propose a distributed OS architecture running on a fully heterogeneous computer cluster, enabling this cooperation through three main components: the abstraction of the entire cluster in a single system image, a distributed shared memory system, and a heterogeneous scheduler.

CCS CONCEPTS

•**Software and its engineering** →**Operating systems;** *Distributed memory; Scheduling;*

KEYWORDS

Operating Systems, Heterogeneous Computing, Distributed Shared Memory

ACM Reference format:
Pierre Olivier, Sang-Hoon Kim, Binoy Ravindran. 2017. OS Support for Thread Migration and Distribution in the Fully Heterogeneous Datacenter. In *Proceedings of HotOS '17, Whistler, BC, Canada, May 08-10, 2017,* 6 pages.
DOI: 10.1145/3102980.3103009

1 INTRODUCTION

Processing speed must keep on increasing. However, it can no longer be done at the expense of an increased power consumption. In the meantime, datacenters are becoming increasingly heterogeneous, integrating multiple, OS-capable, processors based on different Instruction Set Architectures (ISAs). While x86 has been for a long time the main ISA in the datacenter [23], ARM [1, 3, 11, 17, 29] and PowerPC [22] are currently gaining traction. This leads to the notion of *fully heterogeneous datacenter* [13, 21, 30]. Each integrated type of machine has its own performance and power consumption profile for certain classes of applications, and even for specific phases in a single application execution lifetime. This has been shown in the heterogeneous Chip Multi-Processor (CMP) community [35], and we demonstrate that it is also valid in a computer cluster. Efficient distribution and migration of workloads among machines of different types is necessary to leverage the flexibility brought by this heterogeneity. Currently, heterogeneous machines with different ISAs are already present in datacenters. However, they do operate separately. We foresee significant performance and power consumption gains by having these machines cooperating towards the completion of multi-programmed workloads. We argue that all these issues can and should be tackled at the operating system level, without significant effort from application programmer, in order to maximize the solution acceptance.

In this position paper, we propose to implement an OS architecture aiming to enable efficient processes and threads mapping/migration in a fully heterogeneous computer cluster, in order to maximize a workload performance and/or to minimize its power consumption.

The OS architecture we propose is based on three main concepts. First, we advocate for a **distributed, multi-kernel operating system** running on a heterogeneous computer cluster providing a **single system image** of the cluster to the programmer. Multi-kernel systems already communicate through message passing and well-defined interfaces, making them a good fit for distributed implementation on heterogeneous machines. The single system image abstracts the complexity from working with several heterogeneous nodes, and increases programmability. In addition, it permits datacenter-oriented applications to seamlessly share resources such as filesystem or IPC, a feature well needed in current environments [36]. Second, an efficient **Distributed Shared Memory** (DSM) system should be implemented at the operating system level, allowing on-demand address space transfer during task migration between heterogeneous machines, as well as the distribution of a task threads across multiple heterogeneous machines; all of this while conserving the convenient shared memory programming model. Finally, such a system brings the need for a **task scheduler aware of the heterogeneity of the managed cluster**, and capable of taking efficient decisions based on the affinity of the workload towards specific machines. In addition to OS-level components, compiler toolchain support is also needed to provide multi-ISA binaries with the ability to execute/migrate at runtime on/between heterogeneous machines.

We propose to implement these concepts by augmenting the Popcorn Linux [4, 6] OS. Based on Linux, Popcorn is currently capable of migrating tasks between the arm64 and x86_64 architectures (back and forth) in a two machine setup. We propose to enable Popcorn to run at the scale of a heterogeneous cluster.

In this paper we focus on native execution without the use of virtualization. While some virtualization technologies [7, 15] allow for cross-ISA migration, the performance impact is high and unacceptable in many situations. Other virtualization applications such as containers allow to ease the management of computer clusters

HotOS '17, Whistler, BC, Canada
2017. 978-1-4503-5068-6/17/05…$15.00
DOI: 10.1145/3102980.3103009

and are widely used in datacenter today. However, these technologies do not support cross-ISA migration. More generally, while it is going down, virtualization overhead is still a concern in large scale datacenters [36].

Our focus is also made on native shared memory applications. Exploring performance and power consumption benefits for distributed applications such as MPI jobs is an interesting research direction, however we believe that shared memory applications present a strong programmability argument compared to distributed applications.

It should be noted that heterogeneity in terms of performance and power consumption profiles can also be found in a cluster that is homogeneous from the ISA standpoint. The differences between machines characteristics in such an environment might not be as strong as when considering multiple ISAs, and thus would be the benefits to reap from exploiting such a cluster. Even though, to obtain these benefits from such a set of machines, the components of the system proposed in this paper will still be needed.

This paper is organized as follows: in Section 2, we make the case for a fully heterogeneous computer cluster in which machines of different ISAs cooperate in processing a datacenter workload. In Section 3, we detail our system proposal, before concluding.

2 THE CASE FOR HETEROGENEOUS COLLABORATION

Static Job Mapping. For a given workload, different types of machines will exhibit different performance and power consumption behaviors due to the workload characteristics, defined according to multiple metrics: potential parallelism inherent in workloads, need for fast sequential processing, needs in terms of memory latency or bandwidth, disk or network I/O latency, etc.

We ran standard benchmarks on a set of heterogeneous machines, equipped with the following processors: (1) A desktop-class Intel Xeon E5-1650 [18] with 6 cores clocked at 3.5 GHz; (2) a server-class Cavium ThunderX [12] with 2 GHz clock frequency, both in single node (48 cores) and dual node (96 cores) versions; (3) an APM XGene [24] with 8 cores clocked at 2.4 GHz. These machines have different ISAs: x86_64 for the Xeon, arm64 for the Cavium and XGene. They also exhibit strongly different profiles in terms of power consumption and various performance related characteristics such as clock frequency, number of cores, cache size, etc. We used the popular NPB OpenMP (class B and C) and PARSEC (*native* inputs) suites. We also performed a simple I/O test using the dd command to read on each machine a 512 MB file mounted on a NFS filesystem served from a remote machine.

Concerning the best performance, the Xeon is the best machine in 48% of the benchmarks because of its high clock frequency, followed by the dual Cavium (41%) because of its high core count. Concerning the energy consumed, the single Cavium arrives first (56%) as it is a good core count/power consumption trade-off followed by the Xeon (41%). XGENE gives the best power consumption only for the dd benchmark that is not CPU intensive. From these results, we can draw several conclusions. First, **considering each metric, there is no ideal machine that would always yield the best results for the entire set of benchmarks**. Second, **for one particular benchmark, very often the machine offering the highest performance is not the one yielding the lowest energy**, showing that going faster does not necessarily translate into lower energy consumption. Moreover, this second observation shows that according to the metric one wants to optimize (performance/power), tasks should be mapped to different machine types. These conclusions show that the gains brought by a fully heterogeneous cluster promise to be significant, assuming an efficient static job mapping.

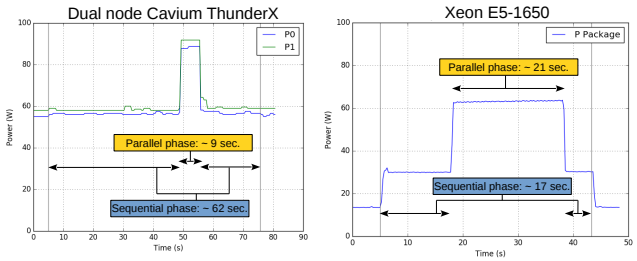

Figure 1: Phases in the PARSEC blackscholes

Leveraging Time Affinity through Task Migration. Some of the benchmarks from the suites we considered exhibit multiple phases in their execution lifetime. We define a *phase* as a portion of a program lifetime during which the program shows some affinity towards a specific type of machines for a particular metric (e.g., performance, power consumption, etc.). Some programs are constituted of multiple phases, and in each phase the affinity for one metric goes to a different machine.The concept of program phases has already been investigated in the chip multiprocessor community. Research in that field defines the program phases according to low level parameters extracted from hardware performance counters, such as instructions per cycle or CPU cache misses [31–33, 35]. Such phases are *micro-phases* and the migration overhead in a cluster will be longer than in a CMP. Thus, it will be difficult to exploit such phases. We propose to work on *macro-phases*, defined according to high-level machine characteristics: number of cores, clock frequency, NUMA memory organization, etc. They are significantly longer than micro-phases.

To illustrate the concept of macro-phases, we take the *blackscholes* benchmark from the PARSEC suite as an example. The power consumption observed over time for that benchmark on the Xeon and the dual Cavium are represented in Figure 1. The benchmark exhibits some strongly sequential initialization and deinitialization phases on both machines. We confirmed that by monitoring the CPU activity, showing that only one core is active (utilized at 100%). The second phase of the workload is a highly parallel one, using all the cores and leading to the maximum power consumption on both platforms. The sequential phase is considerably faster (3.6X) on the Xeon due to its high single threaded processing power. For the parallel phase, it is more than twice faster on the Cavium because of its high number of threads. Assuming a zero-cost migration time, one can estimate that for this benchmark, executing the right phase on the right machine would yield a 32% execution time reduction compared to an only Xeon execution, and a 63% execution time reduction / 76% energy consumption reduction compared to an only Cavium execution.

Preliminary results indicate that such sequential and parallel phases can be found in other benchmarks. The analysis presented here is made according to the relation between the potential for parallelism of a phase, and the number of cores in a machine. One can imagine other types of program/phases to machine affinity such as *memory access pattern of a program according to the memory hierarchy organization of a machine* (in particular the NUMA characteristics of some machines); or *low idle power consumption on I/O intensive workloads*.

Space distribution: Splitting Threads on ISAs. a Distributed Shared Memory (DSM) system is needed to provide efficient on-demand address space transfer in the case of a task migration between heterogeneous machines. Such a DSM implementation will provide the additional benefit of enabling the distribution of threads from the same application over multiple machines of different ISA. In the next paragraphs, we discuss the potential benefits to be gained in doing so. As cross-ISA migration on phase boundaries could be defined as affinity in time, distributing threads of an application over different ISAs would be leveraging affinity in space.

This raises the following question: *Would certain applications benefit of having some of their threads executing one one ISA, and some other threads on another ISA?*. Some applications spawn threads that perform similar work (for example the thread pool model), and it is very probable that there is no space affinity in that case: indeed, because all threads have the same behavior, the affinity would probably go to a single machine. However, space affinity might arise when the work performed by the threads is significantly different.

An interesting example would be a Database Management System (DBMS). DBMS divides the entire query processing into several tasks, such as query parsing, index lookup, index building, logging, storage management, and so forth. These tasks are handled by dedicated threads that communicate each other via messaging queue. This DBMS design maximizes the query processing capacity by pipelining tasks, improves its scalability, and provides the flexibility. Due to the diversity between these tasks, these threads should exhibit different characteristics, showing affinity towards different ISAs.

Another potential benefit brought by the capability to distribute threads among ISAs would be fine-grained power capping or resource management in a heterogeneous cluster, when it is possible to migrate a single thread of one application to free some cores on a powerful machine for an upcoming job with a strong affinity toward that machine, or to accommodate a power cap.

3 SYSTEM MODEL

Hardware Organization. To provide the heterogeneity in a datacenter, an intuitive approach is to keep the homogeneity (in terms of ISA) within a rack but to allow the heterogeneity between racks. Each rack is comprised of a number of machines having the same ISA, computational power, and configurations, whereas different racks may run on different ISA and exhibit different performance/power consumption profiles.

In other words, the heterogeneity is provided on a per-rack granularity. This approach is simple and easy to apply to datacenter environments. However, we argue that this per-rack granularity cannot fully utilize the benefits of the fine-grained ISA affinity nor thread migration. Indeed, in order to take advantage of the fine-grained ISA affinity through thread migration, the migration overhead between heterogeneous nodes should be low enough not to offset the benefit. However, in the per-rack heterogeneous configuration, the distance between two heterogeneous nodes is important as inter-rack interconnects have narrow bandwidth and long latencies, making crossing the rack boundary costly. Thus, it might be difficult or infeasible to capitalize the benefit from the per-rack heterogeneous configuration.

In this sense, we claim that the heterogeneity should be provided within a rack. We define our heterogeneous rack model as follows. The rack contains a given number of *bundles*. A bundle is defined as a group of heterogeneous nodes that are connected each other via a high-speed low-latency, point-to-point interconnect technology. In a concrete example of the heterogeneous rack we have built, a bundle consists of one Xeon E5-2620v4 server connected to a dual-node Cavium ThunderX server [12] using a point-to-point interconnect over PCIe using Dolphin PXH810 [14]. The PCIe interconnect provides 56 Gb/s bandwidth and a stable low latency without switching latency, which is essential to lower the thread migration overhead. In addition, a Xeon Phi 7120p [19] is installed on a PCIe 16x slot in the Xeon server as a co-processor. This configuration provides us with diverse heterogeneity from various aspects; different ISAs (x86_64, K1OM, and arm64), core counts (16, 61, and 96), single-core performance, energy efficiency, interconnect speed, etc.

This bundle is the building block for the heterogeneous rack. We believe this tightly coupled model is indispensable to keep the networking latency low between heterogeneous machines during the performance-critical thread migration and memory consistency protocols. Currently, our rack system is configured with eight bundles (8 Xeon servers, 8 Cavium servers, 8 Xeon Phis). All servers in the rack are connected via InfiniBand using a Mellanox SX6036 switch to communicate each other. We envision this rack as being a building block example for the heterogeneous datacenter by deploying an additional switching layers between multiple racks.

Figure 2 illustrates the organization of our proposed system. While we currently consider only two servers and one co-processor in a bundle, we plan to investigate on different configurations comprising additional nodes of various ISAs such as Power8, SPARC, and Intel Knight's Landing.

Systems Software. As discussed in the literature [10, 34], converting a regular shared memory program written for a single-node system to a distributed model, using for example the message passing interface (MPI), requires a huge amount of effort, impairing the programmability. Furthermore, the diverse heterogeneity aspects between nodes will pose more complexity on top of the reduced programmability. This will prevent users and systems from fully utilizing the advantages of the heterogeneous system, and thus will reduce the acceptance of the proposed solution. In this sense, we argue that the OS is the proper level to integrate systems software managing the complexity of having heterogeneous machines within a rack, and to abstracts that complexity from the application programmer and the system administrator.

As the operating system for a heterogeneous rack-scale system, we will extend our previous work called Popcorn Linux [5, 9]. Popcorn Linux is an open-source project based on the Linux Kernel.

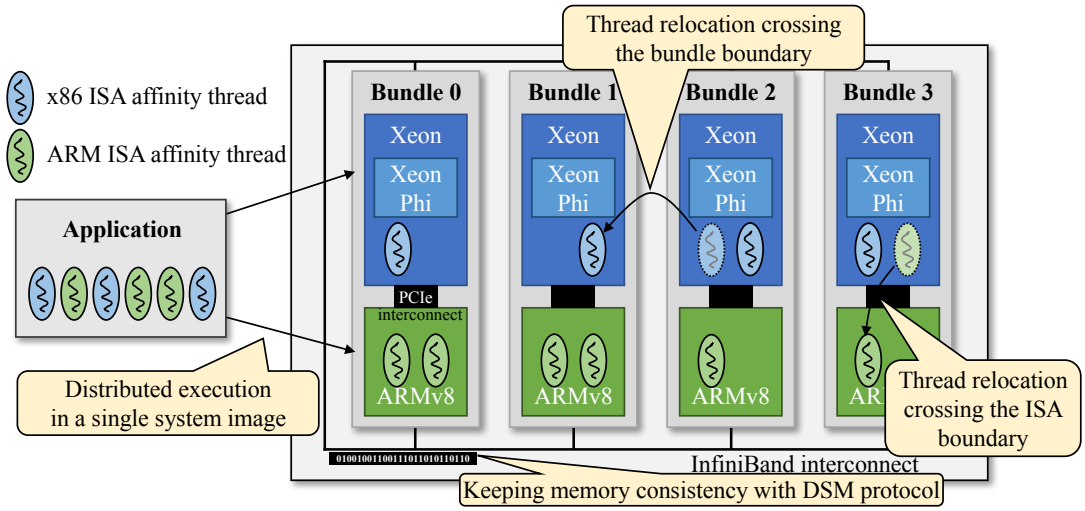

Figure 2: System organization of the proposed heterogeneous rack system

Originally, it aimed to run replicated Linux kernels on each core in a multi-core machine. Based on the feature that provides a single system image over multiple kernels, the project was extended to span over multiple machines having different ISAs. As a *multi-kernel*, Popcorn Linux is particularly well suited for a distributed implementation at a larger (rack) scale, as kernels already communicate with message-passing through well defined interfaces. Currently, Popcorn Linux supports Xeon-Xeon Phi and Xeon-ARM XGene configurations, i.e. pairs of machines. In particular, Popcorn Linux is able to migrate tasks between the x86_64 and the arm64 architectures with a sub-second latency [4]. Note that this number does not contain the address space transfer latency that happen on-demand after migration time.

Popcorn Linux lacks features such as migrating a thread to a machine running the same ISA, spanning a process over more than two machines, and so forth. These features are not only critical for the multiple nodes setup but also required to capitalize from the heterogeneity in the rack. Thus, we are extending the current Popcorn Linux to support these features on various heterogeneous configurations.

In order to provide flexible thread migration within a rack comprised of many heterogeneous nodes, we are primarily extending the memory consistency protocol that Popcorn Linux currently implements. Currently, Popcorn Linux only deals with the case for two hosts; a page that does not exist in the current node is in the other node. In this case, the page is brought to the node via the interconnect to resume the execution. The system calls that manipulate a virtual memory area layout (e.g., `mmap`, `munmap`, etc.) are played at the home location of the current thread. This minimalist feature for the distributed execution has been the major bottleneck even in the two-node setup.

To tackle that limitation, we are working on implementing a per-process level distributed shared memory (DSM) protocol in the kernel space. DSM is a long-discussed concept since the 1990's [2, 8, 20, 28]. Although the promising strong point of DSM is that users can run regular shared memory applications without modification on several nodes in a distributed way, DSM did not gain much popularity outside of the research area at that time. One of the main reasons for this was the low performance of complex DSM protocols over slow interconnect technologies. Indeed, DSM protocols keep the consistency of pages by exchanging data over the network. Thus, the bandwidth and latency of the interconnect are critical to the system performance. In 1990s and 2000s, commodity interconnect technologies were not fast enough to guarantee the desirable performance. Some devices providing fast interconnects were very expensive so that they cannot be applied to a rack-scale computing environment. However, these days, the interconnect speed has increased considerably. For example, PCI Express v4.0 is designed to provide 16Tb/s [27]. Such a high interface performance helps to close the bandwidth gap between the memory bus and peripheral interconnects. Also, recent research discussing the DSM topic in modern interconnect setups shows promising results [16, 25, 26].

Another important point about DSM concerns coping with failures of nodes. While this is not the primary focus of the research presented here, we plan to address this issue through regular checkpointing of DSM-enabled processes address spaces. We foresee that solutions based on replication will introduce overheads that could be very high. On the contrary, given the type of workloads we focus on (long running, compute intensive jobs) regular checkpointing overhead should be acceptable.

Based on these premises, we are reviving the case for the DSM in the rack-scale computing. We expect that the tightly coupled configuration in the bundle helps to minimize the overhead to communicate data for DSM protocol.

To fully utilize the cores abstracted into a single system image, the system requires a *heterogeneous scheduler* that assigns processes and threads to heterogeneous cores scattered in the rack, according to multiple metrics such as the affinity of a thread (or the current macro-phase a thread executes) towards certain ISA, and the current system load. We anticipate that the strong heterogeneity will make the design of the job scheduler a very challenging issue. Followings

are a number of questions that need to be answered in designing such a scheduler.

First, *where will the scheduler run?* It could be ran on a single machine, or in a distributed fashion among the nodes. In either case, one need to consider communication costs of exchanging scheduling information. We plan to design the scheduler in a distributed way. Second question is: *where should an upcoming job first be dispatched?* We are also working on predicting the ISA affinity for a job based on past behavior, or on information that can be extracted from the binary. We also consider embedding affinity information in the binary at compile time. Another question is: *how to assess migration cost?* The cost to migrate a thread will not be static given that the DSM protocol complicates the thread migration. The scheduler should thus estimate the benefit or overhead to migrate a thread.

A final question is: *how to assess at runtime the value of the parameters based on which scheduling decisions will be taken?* Following our example with the PARSEC *blackscholes* benchmark that exhibits several phases in terms of needs for parallelism or fast sequential processing power, an intuitive idea is as follows: Detecting that an application currently executing on a high number of slow cores (for example a Cavium) should be migrated to another machine and given less but faster cores (for example a Xeon) can easily be done by checking CPU activity: a good indicator of that scenario would be an application using 100% of a single core, as it is the case with the first phase of *blackscholes*. Detecting the inverse scenario, i.e. an application currently executing on a small number of cores that is in need for more parallel execution units, i.e. more cores, we still plan to monitor CPU activity. In addition, assuming cores are dedicated to applications, we plan to investigate monitoring the number of context switches for the application's threads on these CPUs. A high number of context switches combined with a high CPU usage might indicate the fact that the application has currently more active threads than its number of assigned cores, and it should benefit from a migration to a high core count machine such as the Cavium.

The design of the job scheduler is still an open question for now, and we will investigate on these cases and try to find reasonable answers to them.

4 CONCLUSION

In order to leverage the potential benefits in terms of performance and power consumption for the fully heterogeneous datacenter, we advocate for OS support using a multi-kernel providing a single system image, DSM support and heterogeneous scheduling.

This work is supported in part by ONR under grant N00014-16-1-2711 and AFOSR under grant FA9550-16-1-0371.

REFERENCES

[1] AMD. 2016. AMD Opteron A-Series Processors. http://www.amd.com/en-us/products/server/opteron-a-series. (2016).

[2] Cristiana Amza, Alan L Cox, Sandhya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony, Weimin Yu, and Willy Zwaenepoel. 1996. Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer* 29, 2 (1996), 18–28.

[3] Applied Micro Circuits Corporation. 2016. X-Gene Product Family. https://www.apm.com/products/data-center/x-gene-family/x-gene/https://www.apm.com/products/data-center/x-gene-family/x-gene/. (2016).

[4] Antonio Barbalace, Rob Lyerly, Christopher Jelesnianski, Anthony Carno, Ho-Ren Chuang, Vincent Legout, and Binoy Ravindran. 2017. Breaking the Boundaries in Heterogeneous-ISA Datacenters. In *Proceedings of the 22th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOSXXII)*. (to appear).

[5] Antonio Barbalace, Binoy Ravindran, and David Katz. 2014. Popcorn: a replicated-kernel OS based on Linux. In *Proceedings of the Linux Symposium, Ottawa, Canada*.

[6] Antonio Barbalace, Marina Sadini, Saif Ansary, Christopher Jelesnianski, Akshay Ravichandran, Cagil Kendir, Alastair Murray, and Binoy Ravindran. 2015. Popcorn: bridging the programmability gap in heterogeneous-ISA platforms. In *Proceedings of the 10th ACM European Conference on Computer Systems (EuroSys'15)*. 29.

[7] Fabrice Bellard. 2005. QEMU, a fast and portable dynamic translator.. In *USENIX Annual Technical Conference, FREENIX Track*. 41–46.

[8] John K Bennett, John B Carter, and Willy Zwaenepoel. 1990. Munin: Distributed shared memory based on type-specific memory coherence. In *Proceedings of the 2nd ACM SIGPLAN Symposium on Priciples and Practice of Parallel Programming (PPoPP'90)*. 168–176.

[9] Sharath K. Bhat, Ajithchandra Saya, Hemedra K. Rawat, Antonio Barbalace, and Binoy Ravindran. 2015. Harnessing Energy Efficiency of heterogeneous-ISA Platforms. In *Proceedings of the Workshop on Power-Aware Computing and Systems (HotPower '15)*. 6–10.

[10] F. Cantonnet, Y. Yao, M. Zahran, and T. El-Ghazawi. 2004. Productivity analysis of the UPC language. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*.

[11] Cavium. 2016. ThunderX ARM Processor; 64-bit ARMv8 Data Center & Cloud Processors for Next Generation Cloud Data Center, HPC and Cloud Workloads. http://www.cavium.com/ThunderX_ARM_Processors.html. (2016).

[12] Cavium. 2016. ThunderX ARM Processors. http://www.cavium.com/ThunderX_ARM_Processors.html. (2016).

[13] Byung-Gon Chun, Gianluca Iannaccone, Giuseppe Iannaccone, Randy Katz, Gunho Lee, and Luca Niccolini. 2010. An Energy Case for Hybrid Datacenters. *SIGOPS Oper. Syst. Rev.* 44, 1 (March 2010), 76–80. DOI: http://dx.doi.org/10.1145/1740390.1740408

[14] Dolphin Interconnect Solutions. 2016. Dolphin PCI Express PXH830 Adapter. http://www.dolphinics.com/download/PX/OPEN_DOC/PXH830_users_guide.pdf. (2016).

[15] Joachim Gehweiler and Michael Thies. 2010. Thread migration and checkpointing in java. *Heinz Nixdorf Institute, Tech. Rep. tr-ri-10-315* (2010).

[16] Isaac Gelado, John E Stone, Javier Cabezas, Sanjay Patel, Nacho Navarro, and Wen-mei W Hwu. 2010. An asymmetric distributed shared memory model for heterogeneous parallel systems. 38, 1 (2010), 347–358.

[17] Hewlett Packard Enterprise. 2016. HPE Moonshot System. https://www.hpe.com/us/en/servers/moonshot.html. (2016).

[18] Intel. 2012. Xeon Processor E5-1650. http://ark.intel.com/products/64601/Intel-Xeon-Processor-E5-1650-12M-Cache-3_20-GHz-0_0-GTs-Intel-QPI. (2012).

[19] Intel. 2013. Xeon Phi Coprocessor 3120p. http://ark.intel.com/products/75798/Intel-Xeon-Phi-Coprocessor-3120P-6GB-1_100-GHz-57-core. (2013).

[20] Pete Keleher, Alan L. Cox, and Willy Zwaenepoel. 1992. Lazy release consistency for software distributed shared memory. In *Proceedings of the 19th ACM Annual International Symposium on Computer Architecture (ISCA '92)*. 13–21.

[21] Boston Limited. 2016. Boston Viridis; Presenting the World's First Hyperscale Server – Based On ARM Processors. (July 2016). https://www.boston.co.uk/solutions/viridis/default.aspx

[22] R. Luijten, D. Pham, R. Clauberg, M. Cossale, H. N. Nguyen, and M. Pandya. 2015. 4.4 Energy-efficient microserver based on a 12-core 1.8GHz 188K-CoreMark 28nm bulk CMOS 64b SoC for big-data applications with 159GB/S/L memory bandwidth system density. In *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*. 1–3. DOI: http://dx.doi.org/10.1109/ISSCC.2015.7062933

[23] Jason Mars and Lingjia Tang. 2013. Whare-map: heterogeneity in homogeneous warehouse-scale computers. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 619–630.

[24] Applied Micro. 2016. XGENE. https://www.apm.com/products/data-center/x-gene-family/x-gene/. (2016).

[25] Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. 2015. Latency-tolerant software distributed shared memory. In *Proceedings of the 2015 USENIX Annual Technical Conference (USENIX ATC'15)*. USENIX Association, 291–305.

[26] Stanko Novakovic, Alexandros Daglis, Edouard Bugnion, Babak Falsafi, and Boris Grot. 2014. Scale-out NUMA. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOSXIX)*. 3–18.

[27] PCI-SIG. 2017. Specifications. http://pcisig.com/specifications. (2017).

[28] Jelica Protic, Milo Tomasevic, and Veljko Milutinović. 1998. *Distributed shared memory: Concepts and systems*. Vol. 21. John Wiley & Sons.

[29] Qualcomm. 2015. Qualcomm Makes Significant Advancements with its Server Ecosystem. https://www.qualcomm.com/news/releases/2015/10/08/qualcomm-makes-significant-advancements-its-server-ecosystem. (Oct. 2015).
[30] Scaleway. 2016. Cloud Computing Features for your Infrastructure. (July 2016). https://www.scaleway.com/features/hardware/
[31] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. 2002. Automatically characterizing large scale program behavior. In *ACM SIGARCH Computer Architecture News*, Vol. 30. ACM, 45–57.
[32] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder. 2003. Discovering and exploiting program phases. *IEEE Micro* 23, 6 (Nov 2003), 84–93. DOI: http://dx.doi.org/10.1109/MM.2003.1261391
[33] Timothy Sherwood, Suleyman Sair, and Brad Calder. 2003. Phase tracking and prediction. In *ACM SIGARCH Computer Architecture News*, Vol. 31. ACM, 336–349.
[34] J Silcock and Andrzej Goscinski. 1995. *Message passing, remote procedure calls and distripbuted shared memory as communication paradigms for distributed systems*. Deakin University, School of Computing and Mathematics.
[35] Ashish Venkat and Dean M Tullsen. 2014. Harnessing ISA diversity: Design of a heterogeneous-ISA chip multiprocessor. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 121–132.
[36] Matei Zaharia, Benjamin Hindman, Andy Konwinski, Ali Ghodsi, Anthony D Joesph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. The datacenter needs an operating system. In *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*. USENIX Association, 17–17.