PLDI: G: Seamlessly Crossing ISA Boundaries for Performance and Security

Robert Lyerly – Ph.D. Student, Advisor: Binoy Ravindran
Bradley Department of Electrical and Computer Engineering, Virginia Tech
{rlyerly, binoy}@vt.edu

1 Problem and Motivation

With the slowdown of performance scaling due to the end of Moore's law, processor architects have increasingly turned to heterogeneity in order to continue advancing performance and energy efficiency [5]. In particular, chip designers have begun incorporating heterogeneous computing elements into systems in order accelerate workloads or achieve different levels of parallelism and energy efficiency. With this explosion of heterogeneity, however, has come a rising cost in application complexity. Each new processor arrives with its own development environment, forcing developers to continually learn both new architectures and new programming models. For example, using OpenCL to offload computation onto Intel's Xeon Phi processor requires on average doubling an application's lines of code versus a single threaded implementation [2]. This programmability burden has spurred new interest in system software to help tame application complexity.

Recently, several works have proposed thread migration between heterogeneous instruction set architecture (ISA) CPUs to obtain better performance and energy efficiency [4, 2, 13]. Developers do not have to learn a new programming model or refactor applications into a new framework to take advantage of heterogeneity; instead they simply use a custom compiler and runtime [4, 13] or new operating system abstractions [2] to migrate thread contexts between different-ISA processors. However, these systems have many limitations. DeVuyst et al. [4] and Venkat and Tullsen [13] only provide the ability to migrate single-threaded applications in a simulated heterogeneous-ISA processor. They additionally scope out any operating system (OS) details for running on such a system. Conversely, Barbalace et al. [2] only describe OS mechanisms necessary for thread migration in overlapping-ISA systems (i.e., no compiler support is required). None of these systems work across emerging heterogeneous-ISA systems like those being introduced into datacenters [12].

The first contribution of this work is a compiler and runtime for heterogeneous-ISA thread migration on real hardware. These mechanisms are built on top of Popcorn Linux [1], an OS which provides thread migration and distributed shared memory (DSM) capabilities for cross-machine application execution. The compiler and runtime are co-designed with the OS to completely automate the process of inter-ISA thread migration. This allows programmers to transparently leverage commodity heterogeneous-ISA systems, and can be used to easily exploit ISA-specific extensions, e.g., cryptography or SIMD instructions.

In addition to providing the core compiler and runtime, the other contributions of this work exploit heterogeneity in previously unseen scenarios. First, this new infrastructure can be leveraged to scale applications across heterogeneous-ISA clusters. In a cluster with an x86 Intel Xeon E5-2620v4 (8 cores/16 threads, max 3GHz clock) and a two-socket ARM Cavium ThunderX (96 cores, 2GHz clock), fork-join applications could run single-threaded phases on a high-power Xeon core and parallel phases on both the lower core count Xeon and the highly parallel ThunderX processors. Application developers can easily take advantage of processors designed to meet different performance, power and parallelism goals for various parts of an application. Second, by removing machine and ISA boundaries, this system could be used to mitigate a host of security threats, such as return-oriented programming (ROP) [3] or side-channel attacks such as Meltdown and Spectre [9, 8]. ROP attacks, which construct ISA-specific "gadgets" from existing application code, could be disrupted by migrating threads and data between different-ISA processors, thereby randomizing code and data layouts. Migrating threads between physically separate machines also eliminates side channels, i.e., shared resources observed to leak sensitive information or manipulated to redirect application execution.

Thus, the contributions of this work are the compiler and runtime components of Popcorn Linux, and how they are leveraged in novel settings.

2 Background and Related Work

Current programming models such as OpenCL and OpenMP provide a functionally-portable interface for offloading computation to heterogeneous processors such as GPUs. However, these models only support a subset of functionality on the target (e.g., no I/O) and require the developer to manually marshal data between the host and device (e.g., flattening data structures, maintaining consistency). Additionally, target architectures are hard-coded, preventing flexible execution in multiprogrammed settings. Using a message-passing framework like MPI allows processes to utilize the system's full functionality but does not solve the problems of data marshaling and static application partitions.

Previous works such as the TUI system [11] studied process migration in heterogeneous-ISA platforms using state transformation techniques across networks of machines. However, they incur extensive overheads as they translate the entire virtual address space between ISA-specific formats, including transforming global data and the heap between per-ISA data layouts in addition to generating per-ISA machine code. More recently, Barbalace et al. developed OS mechanisms to remove execution boundaries between overlapping-ISA processors such as Xeon/Xeon Phi systems [2]. In particular, they describe thread and page migration mechanisms to transparently move threads of execution between different processors while providing a single system abstraction to the application. Other works study offloading from mobile systems to the cloud [7], but require writing the application in a managed

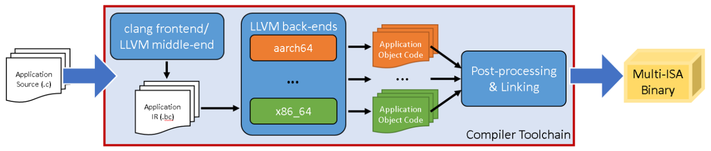

Figure 1: Popcorn Linux's compiler, built on clang/LLVM

language, object semantics for data migration, and extensive modifications to the language VM. Additionally, they only support offloading statically-selected portions of applications.

DeVuyst et al. [4] and Venkat and Tullsen [13] describe compiler and runtime thread migration mechanisms in a simulated heterogeneous-ISA processor. Threads migrate between different-ISA cores at equivalence points [14], or points at which the application has reached a semantically equivalent state of the computation and there exists a valid transformation of a thread's execution state (stack, registers) between ISA-specific formats. In their system, applications run using native execution until a migration is triggered, at which point a thread's state is exported to QEMU. Upon reaching an equivalence point inside the emulator, a runtime translates the thread's stack to the destination ISA's format and returns the thread to native execution.

Popcorn Linux [1] synthesizes these OS and compiler techniques into a complete system which runs on commodity heterogeneous-ISA CPUs networked via commercially available links, e.g., Ethernet, Infiniband and point-to-point PCIe. Popcorn Linux allows applications to transparently migrate between heterogeneous-ISA CPUs without developer intervention. The compiler builds applications instrumented with migration points (unlike previous works which import/export thread contexts into a complex dynamic binary translation runtime [4, 13]). At runtime, threads are signaled by a scheduler to migrate between systems. Threads use metadata generated during compilation describing ISA-specific function activation layouts to transform their stack between ISA-specific formats. Threads then invoke the OS' thread migration mechanism, which starts the thread on the destination node using the transformed register state. As threads access memory, the OS' distributed shared memory (DSM) protocol brings pages over on-demand by intercepting the page fault handler. Thus, applications can seamlessly migrate and access data on heterogeneous-ISA architectures without any specialized programming model or middleware.

3 Uniqueness of the Approach

Heterogeneity has long been believed to be the main path forward to scaling performance. However, system software has only recently begun to make utilizing these architecture easier. Popcorn Linux is a system software innovation whose power arises from extending well-studied OS mechanisms into the hardware diversity jungle. It allows developers to not only utilize existing programming idioms to evaluate and leverage diverse architectures, but also allows developers to enjoy the benefits afforded by the robust Linux development ecosystem.

Figure 1 shows the compiler toolchain used for transparently supporting inter-ISA migration. The compiler and runtime mechanisms for migration integrate cleanly into LLVM, as it provides a strong separation between ISA-agnostic and ISA-specific components. Thus, instrumentation needed to create migratable applications can be inserted at only a few choice locations in the compilation process. Coupled with LLVM's extensive list of supported architectures, these mechanisms can be extended to new architectures with minimal engineering cost once the core components have been built. Developers do not have to refactor any component of their application to make them migratable.

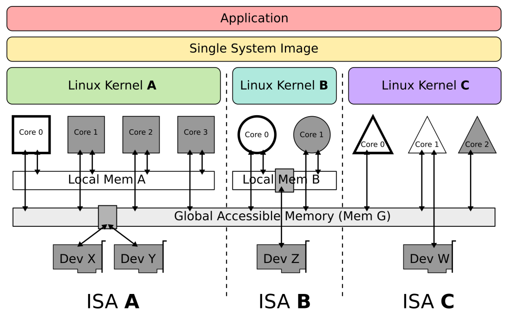

Figure 2: Popcorn Linux Operating System

Figure 2 shows how Popcorn Linux provides a single system image to applications. With full-fledged OS capabilities across distinct machines, applications can take advantage of facilities such as the scheduler, I/O, and shared-memory synchronization (including OS-supported capabilities like futexes). This is in contrast to current programming models for heterogeneous architectures, which have numerous restrictions and only provide low-level building blocks. Developers are required to continually re-build these abstractions for each new architecture in addition to learning a new development environment. Popcorn Linux provides a middle ground, where developers can achieve the benefits of heterogeneous processors with the ease of programming a shared-memory system.

However, there are new challenges that make it distinct from traditional shared-memory SMP systems. The main challenge arises by the move from hardware cache coherency to software DSM. Although applications run as-is, under the covers nodes exchange significant numbers of messages in order to provide a single operating environment. Unlike NUMA systems where memory accesses on remote zones may incur a fractional latency penalty, remote memory accesses incur orders of magnitude more latency – from 100s of nanoseconds to tens of microseconds. Traditional microarchitectural latency hiding techniques such as simultaneous multithreading are no longer effective, meaning the application's page access patterns must be optimized in order to reduce or eliminate performance bottlenecks. Innovations in the compiler and runtime are required in order to fully exploit the capabilities of emerging heterogeneous-ISA systems.

4 Results and Contributions

In this section we describe the compiler and runtime contributions designed for Popcorn Linux's existing thread migration and DSM layers. Section 4.1 describes the compiler and

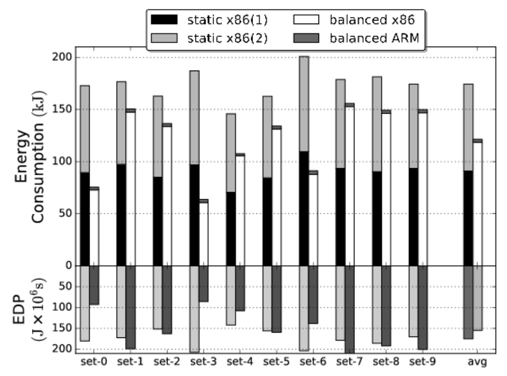

Figure 3: Energy (top) and energy-delay product (bottom) of several sets of applications load balanced across homogeneous- and heterogeneous-ISA systems.

runtime mechanisms developed to migrate threads between different-ISA processors. Section 4.2 describes techniques for leveraging multiple machines simultaneously, include reducing contention on Popcorn Linux's DSM subsystem. Finally, Section 4.3 describes ongoing work related to using these capabilities to defend against ROP or side-channel attacks.

4.1 Compiler/Runtime For Dynamic Migration

The compiler toolchain, based on clang/LLVM, takes POSIX-compliant C or C++ source code as input and uses a custom compiler, linker and post-processing tools to generate *multi-ISA binaries* suitable for runtime migration. At runtime, a state transformation library converts a thread's stack between ISA-specific formats and a migration library handles migrating and restarting a thread on the destination architecture.

Design. The migration mechanisms are designed both to add as little overhead as possible and to seamlessly integrate into Popcorn Linux. Thus, most pieces of the application (global memory, code) are laid out in a common format so the virtual address space does not need to be rearranged at migration time. The thread's execution state (registers, stack) is maintained in an ISA-specific format and transformed before migration, as forcing a common stack layout across all ISAs would cause the compiler to generate sub-optimal code (all live values would have to reside in memory). Instead, the compiler generates metadata describing stack layouts for runtime state transformation. Additionally, rather than allowing migration at arbitrary points by emulating up until an equivalence point (which causes high overheads from importing/exporting state into the emulator and code-cache effects), threads migrate at call-outs inserted by the compiler.

Compiler. First, clang lowers the source code to LLVM bitcode. Next, custom passes analyze the bitcode to both automatically instrument the application with migration points and tag all call sites (i.e., possible transformation sites) with *stackmaps* denoting all live values at the call site; these are the values that must be handled during the state transformation process. Then, the ISA-agnostic bitcode is lowered through each target architecture's backend to generate machine code for each ISA in the system. As the bitcode is lowered, each backend records implementation information about each function (e.g., call frame size, callee-saved registers) and where live values at each call site are allocated (e.g., register, stack slot, constant pool). After compilation, a custom linker creates a common thread local storage layout and aligns all global symbols, so that references to each program object are valid across all architectures.

Runtime. At runtime, a scheduler cooperates with threads to invoke migrations. When a migration is triggered, the thread passes a snapshot of its registers to the state transformation runtime. The runtime first unwinds the thread's stack to both read in the metadata for each live function activation (for both source and destination ISAs) and to calculate the size of the transformed stack. Next, the runtime iterates from the most-recently called function inwards, copying live values between their ISA-specific locations and fixing up other frame data (e.g., return values, callee-saved registers, etc.). The runtime returns a transformed register set to the migration library, which invokes the kernel's thread migration service. The kernel returns the thread to userspace with the transformed register set, where the migration library returns the thread to normal execution. The kernel brings data over on-demand through page faults as threads access both code and data, as the restarted threads have no pages mapped into memory upon returning to userspace.

Results. We evaluated the system on an Intel Xeon E5-1650v2 interconnected to an ARM AppliedMicro X-Gene via Dolphin PXH810 PCIe point-to-point cards. Figure 3 [1] shows the energy consumption (top) and energy-delay product (bottom) of several sets of application workloads drawn from the NASA parallel benchmark suite. The left bars show the results for a homogeneous-ISA setup with two Xeon machines versus the right bars which represent the heterogeneous-ISA setup. Over all sets, the heterogeneous-ISA setup demonstrates on average a 30% reduction in energy and a 11% reduction in EDP versus the homogeneous setup, demonstrating the benefits of using Popcorn Linux for system flexibility when faced with dynamic workloads.

4.2 Ongoing: Scaling to a Heterogeneous Cluster

Compute kernel work splitting, i.e., distributing parallel computation across multiple architectures simultaneously, has been shown to achieve large performance gains in CPU/GPU systems [10]. However, these systems have programmability limitations and limited ability to automatically distribute work across processors (e.g., developer-guided data partitioning). With Popcorn Linux, developers can utilize standard threading models like OpenMP as-is on heterogeneous-ISA clusters without redesigning applications. Popcorn Linux's programmability advantages allow developers to utilize multiple machines as a single large scale-up server.

Popcorn Linux enables this flexibility through on-demand page migration and DSM. Popcorn Linux's DSM protocol operates like a MSI-style cache coherence protocol at the granularity of a page (as the OS observes memory accesses via CPU page faults). As threads migrate between machines and read memory, pages are replicated in a read-only state and incur a one-time migration cost. Threads writing to memory instead cause the system to grant exclusive access to the machine executing the write. Thus, machines collectively have sequential consistency and existing synchronization primitives, e.g., spinlocks, can run unmodified across a cluster.

However, Popcorn Linux's DSM protocol and page migrations do come with a cost. As mentioned in Section 3, a memory access that causes an inter-machine page migration takes on the order of tens of microseconds. Thus, while applications run correctly under Popcorn Linux, their structure may lead to sub-optimal page access patterns and inter-machine migration thrashing. For example, 2 threads executing on different machines and writing to separate global objects mapped to the same page in the virtual address space will cause the DSM protocol to repeatedly unmap the page from one machine and migrate it to the other. This "ping-pong" behavior is similar to false-sharing of cache lines, although at a much larger granularity and latency.

Fortunately, due to Popcorn Linux's flexibility oftentimes small adjustments to the application mitigate or eliminate false sharing. For example, placing per-thread data structures or heavily-used global data onto separate pages often reduces page migrations caused by threads on separate nodes. These optimizations have low memory overhead, especially with 64-bit virtual address spaces. Popcorn Linux makes it easy to profile, debug performance bottlenecks and iteratively improve applications with minimal development effort.

Profiling page access patterns. In order to help developers analyze memory access patterns, the kernel can dump a *page access trace* describing the DSM protocol's actions. For each inter-machine page migration, the DSM protocol writes a tuple containing fault information (address, instruction, permission type, etc.). A page access trace tool uses this information together with the binary to generate statistics and identify program locations causing the most inter-machine memory contention. Using this information allows developers to spot sub-optimal memory access patterns, including cyclical page fault patterns (e.g., read-replicating a page across all machines followed by writes causing invalidation storms) or separate global objects mapped to the same page causing unnecessary contention. Developers can then make adjustments to alleviate these issues (e.g., applying alignment attributes to data structures).

Per-Node Memory Allocator. One source of sub-optimal memory access patterns that is hard for developers to change is the memory allocator. Applications that allocate many small objects on the heap may cause enormous DSM pressure, as the allocator may naively allocate lots of objects accessed by threads on different nodes to the same page. Rather than forcing developers to refactor `malloc()` usage within applications, we instead developed a node-aware memory allocator. We extended the idea of *heap arenas* from other allocators like jemalloc [6] and modified the default memory allocator in Popcorn Linux's libc library. Our malloc implementation uses a per-node heap so that threads allocate heap memory local to the node on which they're executing, minimizing unnecessary conflicts with threads on other nodes.

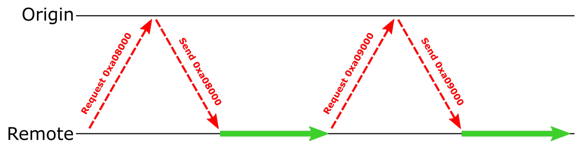

Figure 4: Lazy/on-demand page migration. Every new page access incurs an inter-node fault.

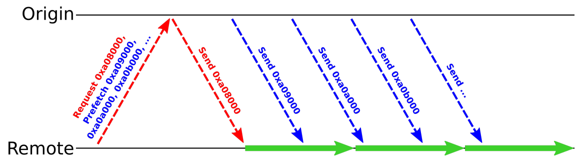

Figure 5: Prefetching pages minimizes the long latencies associated with inter-node faults.

Threads can also move an allocation between per-node heaps using `realloc()` if necessary. Data allocated in any of the per-node heaps is accessible on any node in order to not break the shared-memory abstraction, but the heap itself is logically partitioned between nodes to prevent contention.

Prefetching. Even with reduced memory contention, most non-trivial applications still execute cross-node memory accesses. Figure 4 demonstrates the long latencies associated with on-demand migration – threads spend significant time waiting for pages in-between bursts of computation. If applications demonstrate predictable memory access patterns, the toolchain can give the DSM layer prefetching hints on where to place pages in the system. A small prefetching library batches together per-node prefetch requests from application threads and informs the DSM layer via `madvise()`. Figure 5 shows an example of how prefetching can reduce page access times for threads. Prefetching does not impact the correctness of the application, but instead gives the DSM layer information on where it can place pages to reduce contention.

Rather than requiring developers to manually instrument applications with prefetching hints, we developed compiler extensions inside clang to insert prefetching hints based on loop iteration ranges and array access patterns. Developers add a `#pragma popcorn prefetch` annotation to a structured block (e.g., for-loop), giving the compiler scoping information for analysis. The frontend analyzes access patterns and any loop bounds to generate calls to the prefetching library. This not only reduces the amount of inter-node migration traffic, but also can utilize output from the page access trace tool to generate better hints and inform the developer of potential bottlenecks in a feedback-driven optimization loop.

OpenMP-specific optimizations. OpenMP provides a strong semantic base onto which we can add Popcorn Linux-specific optimizations. We extended OpenMP's work sharing constructs in several ways to optimize for Popcorn Linux. First, we added a `prefetch` clause which when added to work sharing directives informs the compiler to analyze and emit prefetch calls as described above. A small difference is that rather than prefetching for entire loop iteration ranges, the compiler inserts prefetch calls for the partitioned loop iteration range assigned to each thread. Additionally, variables included in an OpenMP `shared` clause (semantically, all threads share a copy of the same variable) are copied into global memory for the duration of the parallel section and copied back into stack memory after. Normally the compiler allocates space for these variables on the main thread's stack and passes a reference to all team threads. This leads to false sharing caused by all threads accessing the main thread's stack data and contending with the main thread's normal stack usage.

In addition to compiler-specific features, we also are experimenting with OpenMP runtime changes. The first change is to the barrier implementation, which is implicitly called at the end of most OpenMP directives. The current implementation spins for a number of cycles and eventually waits using futexes. While semantically correct, the spinning portion of the barrier is sub-optimal on Popcorn Linux, as it can lead to short bursts of rapid page migrations. Instead, we are modifying the barrier implementation to use a hierarchical approach. Instead of spinning globally, threads first synchronize on a per-node local barrier. The last thread to reach each of the local barriers then waits on a global barrier, reducing both synchronization pressure and page migrations.

The last runtime change we are exploring is dynamic loop splitting. OpenMP by default distributes work evenly across all available cores. However in a heterogeneous-ISA system, the cores may have vastly different performance profiles. Rather than evenly splitting the work across cores, we are developing new work sharing heuristics, such as extending OpenMP's `dynamic` loop scheduler based on a hierarchical approach. For example, a two level scheduler would first split the loop iteration ranges based on a performance capacity estimate of each system executing the parallel region. Next, loop iterations could be assigned to each core in each system from that system's total range of loop iteration ranges. This will also have an impact on the prefetching capabilities of the compiler, as its ability to insert prefetch calls requires a-priori knowledge of how loop iterations are assigned to each node.

4.3 Future Work: Popcorn Linux for Security

Inter-machine and inter-ISA migration can also be used to defend against several classes of attacks. As mentioned previously, ROP attacks chain together small sequences of machine code to construct functionality. Inter-ISA migration disrupts these attacks by changing the code stitched together by the attacker. Small sequences of instructions turn into random noise, breaking the gadget. The metadata generated by the compiler also provides some interesting capabilities – for example, when unwinding the stack to look up transformation metadata, the compiler can perform a control-flow integrity check to see if the stack has been disrupted for use by a ROP gadget. Additionally, the metadata gives the transformation runtime the ability to dynamically adjust both locations of stack and register data, and locations of global data (e.g., runtime address space randomization). This requires the ability to find live value locations and global memory references at runtime, capabilities that can be easily built into the transformation process. Because state transformation only takes hundreds of microseconds [1], frequent or randomized migrations could also be used to disrupt side-channel attacks. These attacks require priming microarchitectural state – migrating between machines which do not share this state would eliminate these information leakages. We plan to explore using these capabilities to defend against a number of security attacks with low overhead on commodity hardware.

References

[1] Barbalace, A., Lyerly, R., Jelesnianski, C., Carno, A., Chuang, H.-R., Legout, V., and Ravindran, B. Breaking the Boundaries in Heterogeneous-ISA Datacenters. In *Proceedings of the Twenty Second International Conference on Architectural Support for Programming Languages and Operating Systems* (2017), ASPLOS XXII. To Appear.

[2] Barbalace, A., Sadini, M., Ansary, S., Jelesnianski, C., Ravichandran, A., Kendir, C., Murray, A., and Ravindran, B. Popcorn: Bridging the Programmability Gap in heterogeneous-ISA Platforms. In *Proceedings of the Tenth European Conference on Computer Systems* (New York, NY, USA, 2015), EuroSys '15, ACM, pp. 29:1–29:16.

[3] Checkoway, S., Davi, L., Dmitrienko, A., Sadeghi, A.-R., Shacham, H., and Winandy, M. Return-oriented programming without returns. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2010), CCS '10, ACM, pp. 559–572.

[4] DeVuyst, M., Venkat, A., and Tullsen, D. M. Execution migration in a heterogeneous-isa chip multiprocessor. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2012), ASPLOS XVII, ACM, pp. 261–272.

[5] Eeckhout, L. Is moore's law slowing down? what's next? *IEEE Micro 37*, 4 (2017), 4–5.

[6] Evans, J. A scalable concurrent malloc (3) implementation for freebsd. In *Proc. of the BSDCan Conference, Ottawa, Canada* (2006).

[7] Gordon, M. S., Jamshidi, D. A., Mahlke, S., Mao, Z. M., and Chen, X. Comet: Code offload by migrating execution transparently. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2012), OSDI'12, USENIX Association, pp. 93–106.

[8] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., L ipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y. Spectre attacks: Exploiting speculative execution. *ArXiv e-prints* (Jan. 2018).

[9] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., and Hamburg, M. Meltdown. *ArXiv e-prints* (Jan. 2018).

[10] Luk, C.-K., Hong, S., and Kim, H. Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, 2009), MICRO 42, ACM, pp. 45–55.

[11] Smith, P., and Hutchinson, N. C. Heterogeneous process migration: The Tui system. *Software-Practice and Experience 28*, 6 (1998), 611–640.

[12] Teich, P. Deep Dive Into Qualcomm's Centriq Arm Server Ecosystem, December 2017. `https://www.nextplatform.com/2017/12/06/deep-dive-qualcomms-centriq-arm-server-ecosystem/`.

[13] Venkat, A., and Tullsen, D. M. Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture* (Piscataway, NJ, USA, 2014), ISCA '14, IEEE Press, pp. 121–132.

[14] von Bank, D. G., Shub, C. M., and Sebesta, R. W. A unified model of pointwise equivalence of procedural computations. *ACM Trans. Program. Lang. Syst. 16*, 6 (Nov. 1994), 1842–1874.