Offloading Datacenter Jobs to RISC-V Hardware for Improved Performance and Power Efficiency

Balvansh Heerekar
Virginia Tech
Blacksburg, VA, USA
balvansh@vt.edu

Cesar Philippidis
Rasec Tech
San Jose, USA
cesar@rasec.tech

Ho-Ren Chuang
Virginia Tech
Blacksburg, VA, USA
horenc@vt.edu

Pierre Olivier
The University of Manchester
Manchester, UK
pierre.olivier@manchester.ac.uk

Antonio Barbalace
University of Edinburgh
Edinburgh, UK
antonio.barbalace@ed.ac.uk

Binoy Ravindran
Virginia Tech
Blacksburg, VA, USA
binoy@vt.edu

Abstract

The end of Moore's Law has brought significant changes in the architecture of servers used in data centers, increasingly incorporating new ISAs beyond x86-64 as well as diverse accelerators. Further, single-board computers have become increasingly efficient and can run certain Linux applications at significantly lower equipment and energy costs compared to traditional servers. Past research has demonstrated that offloading applications at runtime from x86-based servers to ARM-based single-board computers can result in increases in throughput and energy efficiency. The RISC-V architecture has recently gained significant commercial interest, and OS-capable single-board computers with RISC-V cores are increasingly available at the commodity scale.

In this paper we propose a system that offloads jobs from an x86 server to a RISC-V single-board computer at runtime, with the goals of improving job throughput and energy saved. Towards this, we port the Popcorn Linux multi-ISA toolchain and runtime framework to RISC-V, enabling the live migration of applications between an x86 Xeon server and a SiFive HiFive RISC-V board. We further propose a scheduling policy, *Lowest Slowdown First* (LSF) that drives the offloading of long-running and stateful datacenter background jobs from the server to the board, to alleviate workload congestion on the server. LSF's policy relies on monitoring jobs' performance on the server, predicting the slowdown they would suffer if running on the board, and migrating the jobs with the lowest estimated slowdown. Our evaluation shows that LSF yields up to 20% increase in throughput while also gaining 16% more energy efficiency for compute-intensive workloads.

CCS Concepts

• **Computer systems organization** → **Heterogeneous (hybrid) systems**; *Embedded systems*; • **Software and its engineering** → *Operating systems.*

Keywords

RISC-V, x86, heterogeneous ISA, execution migration

1 introduction

As CPUs cannot get faster any easier, modern computers include a plethora of heterogeneous processing units alongside the main CPU. While the CPU is still in charge of the control path, various types of computations can be offloaded to heterogeneous processing units such as GPUs, TPUs, and FPGAs. Heterogeneous processing units have been introduced to accelerate specific classes of computations; hence, improving application performance. Heterogeneous processing units can also reduce energy consumption, for example in mobile phones and laptops with microarchitectural heterogeneity, including ARM big.LITTLE and Intel P-core/E-core.

An interesting emerging data point in the heterogeneous computing landscape is *instruction set architecture (ISA) heterogeneity*, such as general-purpose CISC (e.g., x86-64) and RISC (e.g., ARM64) cores integrated together in the same hardware. An exemplar case is "smart" I/O devices such as Smart NICs [46] and Smart SSDs [37]. These currently integrate mostly ARM CPUs, and are expected to embed CPUs of more diverse ISAs in a near future, e.g. RISC-V [12]. When these smart devices are attached to a host machine, generally integrating an Intel x86-64 CPU, the result is a heterogeneous-ISA system. The research community has

SYSTOR '24, September 23–24, 2024, Virtual, Israel

ACM ISBN 979-8-4007-1181-7/24/09
https://doi.org/10.1145/3688351.3689152

been actively investigating heterogeneous ISA systems to understand their performance, energy, and security advantages [3, 8, 19, 21, 25, 29, 32, 42].

Hardware resource sharing is a common practice used in data centers to improve overall resource utilization and reduce costs. Major factors that drive up data center costs include server acquisition, power consumption, and equipment cooling costs [9]. Significant research works focused on reducing these costs. An interesting research direction in this space is to augment high-performance servers with low-cost embedded boards in order to improve performance and energy efficiency at a low cost, instead of buying additional servers. We showed in past research [29] that an x86-based server can be connected to one or more low-cost ARM-based embedded boards that are a fraction of the server's price/power consumption. Such a setup enables the offloading of certain jobs from the server to the boards to alleviate the server's resource congestion. This allows job throughput to be increased by up to 67%, and energy consumption to be reduced by up to 56%.

Continuing that line of work, we are motivated to investigate whether and how RISC-V-based embedded boards, when coupled with x86 servers, may increase the overall throughput and reduce energy consumption. While RISC-V is based on the same RISC principles as ARM, it has recently gained traction due to its open-source nature. The cost of producing CPUs with ARM processors includes the cost associated with procuring the IP from ARM for production. This is avoided in the case of RISC-V as the ISA is open-source and royalty-free. This has caused significant industry interest in RISC-V. For example, several storage device manufacturers are incorporating RISC-V-based CPUs in their newer storage device products [13, 38].

To support this motivation, we conducted experimental studies to understand the difference in application execution times and energy consumption on an x86 server (Intel Xeon E5-2637v3) and a RISC-V board (SiFive HiFive). The experiments (Section 4) focused on compute-intensive benchmark programs that are representative of datacenter background jobs. The results revealed that for some programs, the execution time slowdown on the RISC-V board is relatively small, whereas it is very high for a few other programs. We also observed that programs with higher slowdowns do consume more energy (see Figure 3): although the power consumption (in watts) of the SiFive board in activity is much lower than that of a server, the execution time of these programs is so long on the board that the total energy consumed ends up higher than on a server. Thus, we conclude that a setup consisting of a low-cost, low power RISC-V embedded board attached to an x86 server can be optimized for consolidation by selectively offloading to the board the jobs that would suffer the lowest slowdowns.

To enable such a setup, we fist port a heterogeneous ISA systems software infrastructure, Popcorn Linux [4, 6], to the RISC-V architecture, to allow runtime execution migration of an application between an x86-64 and a RISC-V machine. Popcorn Linux consists of an LLVM-based compilation and linking toolchain that produces multi-ISA binaries, a runtime system software that dynamically rewrites ISA-specific program state from one ISA/ABI format to another, and a Linux-based operating system (OS) that enables process and thread migration and distributed shared memory across ISA-different processors. Popcorn Linux's original version only supports machines with the x86-64 and aarch64 ISAs, and the port to RISC-V necessitated the update of several software components in the build toolchain and runtime.

Next, building upon our x86-RISC-V cross-ISA runtime migration mechanism, we develop an application job scheduler that targets job consolidation on a setup made of an x86 server and a RISC-V board. The scheduler implements a policy called Lowest Slowdown offloaded First (LSF), which relies on monitoring job performance on the server using performance counters, and estimating from these measurements job performance on the board. These predictions can then be used to optimize offloading decisions.

Our work targets datacenter compute-intensive workloads, also known as *background jobs*. These jobs are long-running [27] (from minutes to days) and stateful, hence offloading cannot be achieved by killing and restarting regular native binaries [16], as the loss of progress would be unacceptable: such jobs require runtime migration. We integrate LSF into Popcorn Linux's runtime system and evaluate its efficiency using a set of micro- and macro-benchmarks representative of datacenter background workloads. Our evaluation platform includes a SiFive HiFive RISC-V board that is connected to an Intel Xeon E5-2637v3 server using a low-speed Ethernet switch (1 Gbps). The evaluation shows that LSF can obtain up to 20% throughput increase and up to 16% energy efficiency. We consider these gains significant as the cost of a SiFive board is at minimum 3X less than that of a Xeon server.

The contributions presented in this paper as follows:

(1) The port of the multi-ISA Popcorn Linux toolchain and runtime framework to RISC-V, enabling live migration of a program from an x86 to a RISC-V machine.
(2) A job scheduler driving the migration of workloads from an x86 server to a RISC-V board, targeting job consolidation.
(3) The evaluation of that scheduler, demonstrating job throughput increases and a lowering in energy consumption vs. single-ISA setups.

This paper is organized as follows: Section 2 presents some background information about Popcorn Linux. Section 3 presents our port of Popcorn Linux to the RISC-V ISA,

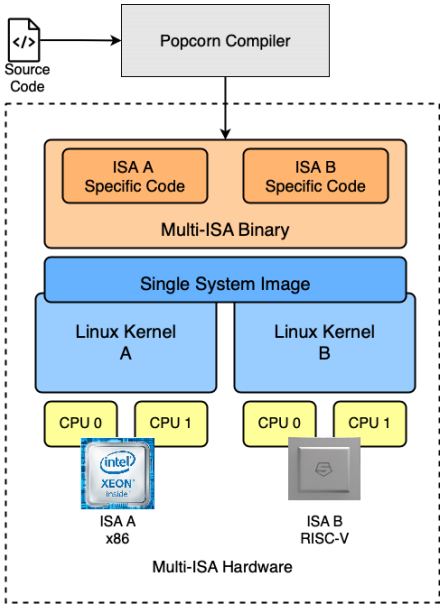

Figure 1: Popcorn Linux's architecture. Popcorn runs an instance of the Linux kernel on each ISA, and instances communicate to expose a single system image to applications. Applications are compiled through Popcorn's toolchain into multi-ISA binaries that can run on top of that system images, i.e. their threads can span/transparently migrate between cores of different ISAs.

and Section 4 describes our application job scheduler. We evaluate the performance and power consumption benefits of the scheduler in Section 5. Related works are discussed in Section 6, before concluding in Section 7.

2 Background: Popcorn Linux

2.1 Overview

To enable the execution migration of an application at runtime between heterogeneous ISA processors, several capabilities are needed. First, a compilation and linker toolchain producing binaries that can execute on such processors is required. Second, the underlying operating system must provide the ability to migrate threads between heterogeneous ISA processors. Third, to provide fast migration with on-demand data transfer, and to allow different threads of a single multi-ISA application to span ISAs, the OS should have the capability to provide distributed virtual shared memory across heterogeneous ISA processors. Popcorn Linux [3] is a system software stack, i.e., an OS, compiler, and runtime system, that provides all these features.

Popcorn Linux, whose architecture is presented in Figure 1, uses a replicated OS kernel model [5] wherein one OS kernel instance is run on each machine of an ISA-heterogeneous cluster of networked computers. The kernel instances communicate with each other via message passing over the local network, to coordinate and ensure consistency of a part of the kernel state. In particular, they provide a single system image including a globally consistent namespace for CPUs, application address spaces, and process IDs. The OS also provides all the necessary support for threads to dynamically migrate and execute across processors of different ISAs [20, 24], including the abstraction of distributed shared virtual memory (across ISAs) as a first-class Linux kernel abstraction [28, 36].

The OS is implemented as a collection of kernel subsystems with designated functionalities: a *Virtual Memory Area (VMA) server* that builds a migrated process's VMA by raising page faults, a *process server* that implements thread migration, termination, and cancellation services, and a *page server* that handles page faults and message invalidation requests/responses for implementing distributed shared virtual memory. The OS's messaging layer for kernel-to-kernel communication is implemented as a high-performance, low-latency in-kernel messaging layer that consists of a messaging interface, a transport layer, and various device drivers that support different interconnects (e.g., Ethernet, RDMA, PCIe).

2.2 Building Multi-ISA Binaries

Popcorn's compilation and linking toolchain produces binaries that can execute on heterogeneous ISA processors. Figure 2 shows a high-level overview of this toolchain. The toolchain extends the LLVM compiler infrastructure, takes as input unmodified C files, and generates a multi-ISA binary that contains a single data section (`.data`), multiple code sections (`.text`), one per ISA, and state transformation metadata that enables rewriting of the program's ISA-specific state from one ISA/ABI format to another, thereby enabling program's execution migration between ISA-different CPUs. To enable a common data section, we assume the same alignment and sizes for primitive C types, as well as the same endianness, across all considered ISAs (true for all modern 64-bit ISAs). The same memory allocator (i.e., heap algorithm) is used on all ISAs.

Execution migration across ISA-different CPUs is semantically consistent only at an *equivalence point* [43], i.e., a point in the program that generates an identical application state independently of the ISA in which it has been compiled for. An LLVM pass, part of the toolchain, can automatically insert migration points at equivalence points (which are function boundaries) in the application. Additionally, the pass determines all the live variables at each migration point (using LLVM's stack map mechanism). Since each ISA's backend (i.e., the register allocator) may make a different decision about where to store these live variables (i.e., stack slot or

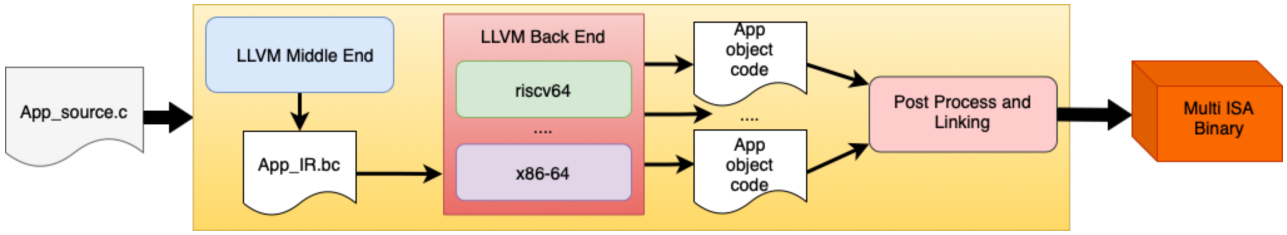

Figure 2: Popcorn Linux compiler and linker toolchain. The toolchain takes an application's sources as input and outputs a multi-ISA binary capable of migrating at runtime between different ISAs. A modified LLVM compiler instruments the program at the intermediate representation level with metadata that will allow architecture-specific state transformation at migration time. A modified linker finalizes the binary by creating two distinct code segments (one per ISA), with possible targets of function pointers (function addresses) aligned at the same virtual addresses, and a single data section with a format compatible with both ISA.

register), the pass also instruments the LLVM intermediate representation to inform each ISA's backend to generate information about the destination of the live variables at each migration point. These destinations need to be preserved if the application were to be migrated to a different architecture at a migration point to ensure that the application resumes after migration in a consistent state and with high performance.

Apart from the live variables and their destinations, the compiler also notes the information about the callee saved registers and the frame sizes. To correlate all the call site information at runtime, each site is also assigned a unique identifier. In the final step, the linker takes in all the object files and produces a multi-ISA binary that has an aligned virtual address space, i.e., all static variables, global variables, and functions are given the same virtual address. This alignment is done using a post-processing script that generates custom linker scripts. By aligning the virtual address space and using the same dynamic memory allocator on all ISAs, pointers remain valid across migrations. The generated multi-ISA binary also contains the metadata required for transforming the application's program state across ISAs during migration.

2.3 Runtime Cross-ISA Thread Migration

When an application is executed, its threads execute normally until they hit a migration point, which is a call to Popcorn Linux's run-time system. The run-time system checks if a cross-ISA migration for the thread in question has been requested. This is done by issuing a system call to check a per-thread flag that is used to signal migration by Popcorn's scheduler. If a migration is requested, the run-time system takes a snapshot of the thread's current stack and registers, and begins the transformation of the stack and register set from the source ISA/ABI format to the destination ISA/ABI format. The first step is to unwind the stack, check all the activations that are active, and load each of these functions' metadata. The stack is then traversed, moving all the active live values to the destination ISA location. Non-stack memory (e.g. global variables and heap allocations) will be transferred on demand later when the application resumes on the target machine.

With the transformed register set, the run-time system issues a special system call to the OS, which migrates the outermost function's transformed register set to the destination machine. The OS kernel on the destination machine creates a new thread with the transformed register set and returns the thread to the user space. The thread then resumes its execution on the destination machine as if returning from a system call.

After a thread is migrated from a source to a destination machine, the migrated thread does not have any pages mapped into its memory, and all memory accesses, therefore, result in page faults. When page faults occur, the OS page fault handler (on the destination machine) intercepts them, enabling it to observe the page data accessed by the thread. The destination OS kernel informs the source OS kernel about the missing page data, which are unmapped from the source and sent to the destination where they are mapped into the thread's address space.

3 Popcorn Linux's RISC-V Extension

Popcorn is originally a distributed operating system kernel based on Linux [5], which was extended to run on heterogeneous hardware with partially overlapping ISAs, i.e., an Intel Xeon and Intel Xeon Phi platform [6]. Subsequently, Popcorn Linux for fully non-overlapping ISAs, i.e., x86 and ARM CPUs, was developed [3], integrating a toolchain made of LLVM, GNU binutils, the gold linker, and the Musl C standard library in order to create multi-ISA binaries i.e. applications able to migrate at runtime between machines of different ISAs. That version of Popcorn was subsequently

enhanced to support an advanced distributed shared memory implementation across ISAs [21, 26] and the migration of containers [2, 47]. It is the first time since the inception of multi-ISA (x86-ARM) Popcorn [3] that the toolchain is ported to a new ISA, and we describe the port in the rest of this Section.

Popcorn Linux's port to 64-bit RISC-V processors targets the RV64G subset of the RISC-V ISA [45], providing hardware support for atomic, multiplication, and single and double precision floating point instructions. Conceptually, integer multiplication and floating-point support could be emulated in software, but that has been left as a future project.

Porting the Popcorn software ecosystem to RISC-V was not a straightforward switch from the existing x86-64/ARM64 version, and involved changing part of the toolchain (e.g. the linker) as well as updating the version of other tools. There are three major components involved in this port, compiler, runtime, and operating system, all described below.

3.1 RISC-V Compiler Toolchain

The RISC-V compiler toolchain is based on LLVM version 9 and utilizes the binutils BFD linker. From a historical standpoint, it should be noted that this port utilized the first version of LLVM that supported the RISC-V ISA. As such, porting Popcorn Linux to an ISA with more mature compiler support may not have involved as much effort. This process was completed within six months by a single engineer. Likewise, the binutils BFD linker is selected for this port out of necessity, as the original Popcorn Linux linker, gold, has largely been abandoned by the binutils developer community, and it did not support RISC-V.

After rebasing the Popcorn Linux code generation patches from LLVM version 3.4 to LLVM version 9, the first task in the RISC-V port is to add support to generate stack map metadata for the RISC-V ISA. Stack map metadata essentially contains debug information to encode the locations of variables both in hardware registers and on the stack. Stack maps required all variables to be accessed relatively to the frame pointer register. However, RISC-V LLVM aggressively avoids using the frame pointer. Therefore, in the RISC-V port of Popcorn Linux toolchain, we force the utilization of frame pointer-based addressing for Popcorn applications.

Next, the RISC-V LLVM port needs to be updated to disable usage of the *small data* (`.sdata`) segment for global variables because x86-64 does not support it. By design, Popcorn Linux requires all the global variables and functions to be placed at common addresses across multi-ISA executables. As such, each multi-ISA application is compiled with the `-ffunction-sections` and `-fdata-sections` compiler flags, placing each function and global variable into its own section, that can later be aligned at link stage. This necessitates preventing LLVM from utilizing architecture-specific data segments. By utilizing the `.sdata` segment, the RISC-V ISA can access those variables using the `gp` register, and therefore generate more compact code when accessing global variables.

The key challenge in porting Popcorn Linux to utilize the binutils BFD linker is to align thread local storage (TLS) symbols at the same addresses on x86-64 and RISC-V. The GNU TLS implementation [14] provides two variants for TLS data layout. Most ISAs utilize variant I, but RISC-V and x86, utilize variant II. All Popcorn Linux ISAs utilize a modified version of variant I, because that is what ARM utilizes. Fortunately, this does not involve any compiler changes, but it does require the linker and C library to collaborate with their interpretation of TLS data. In terms of the BFD linker port, this only involves minimal changes to the way that the linker interprets offsets for TLS relocations.

The last RISC-V-specific linker change involves disabling the relocation relaxations. The RISC-V ISA was specifically designed for simplicity. Unlike other ISAs that support a plethora of addressing modes, RISC-V only supports three addressing modes. As such, the RISC-V ISA relies heavily on the linker to perform relocation relaxations to optimize the executable code for size. For example, instead of using two instructions to perform a jump, if the linker knows that offset to the jump target can fit within a single instruction, then it will only generate one instruction to perform the jump. These aggressive relaxations present a challenge in the RISC-V port in the context of the post-processing script that generates the custom linker scripts used to align symbols at the same addresses in both ISAs. Different alignments cause the relocations to be relaxed differently, and hence the symbol alignment can be off in different executables. We disable this optimization to ensure the proper alignment of symbols across ISAs.

The modifications brought by the Popcorn toolchain over a standard binary, such as disabling the small data section and aligning symbols introduce a certain performance and memory overhead. To our experience in most cases this overhead has been shown to be acceptable, and we believe improving the toolchain on that aspect is orthogonal to the contributions presented in this paper.

3.2 RISC-V Runtime Environment

The core Popcorn Linux runtime libraries, including the stack transformation and migration libraries are mostly written in C, but it does also contain snippets of assembly code to handle critical migration functionality when the stack is unavailable during its rewriting. Those aforementioned libraries utilize objects with callback functions to facilitate

stack rewriting and initiate migration. Porting the Popcorn Linux runtime environment to the RISC-V ISA involves replicating a lot of boilerplate code in both the stack transformation and migration libraries. Besides for some assembly routines, most of the changes involve populating data structures with RISC-V-specific register information and adding code to register the RISC-V ISA as a target.

The Linux kernel 5.2 version is the basis for the RISC-V effort described in this paper. The port retains the Musl C library version 1.1.18. Official support for RISC-V was added in Musl 1.1.23. Rather than rebase the Popcorn Linux patches to Musl 1.1.23, the RISC-V functionality is backported to Musl 1.18. Notable RISC-V changes in the backport involve adding assembly directives to place each function in a separate section, adding Popcorn-specific system calls, and changing RISC-V TLS layout to variant II.

Lastly, the post-processing data alignment and stack map metadata tools have to be updated to support RISC-V. The changes to the stack map metadata tool are relatively trivial, as the stack map metadata is itself generic and ISA-independent. The changes to the data alignment tool are more involved, as a new linker script template needs to be introduced for RISC-V. Specifically, the placement of the RISC-V ELF segments has to match that of x86.

3.3 RISC-V Linux Kernel

We utilize the 5.2 Linux kernel for both x86 and RISC-V for the RISC-V port. One issue preventing migration between x86 and RISC-V is the size of the virtual address space. Intel x86-64 processors support 47 bits of addressable virtual memory whereas 64-bit RISC-V hardware only supports 38 bits of virtual memory at the time of development. As a consequence, the x86 Linux kernel was modified to only support 38 bits of virtual memory. Without this change, the x86 stack may not be addressable on RISC-V if located above the limit of a 38-bit virtual address space. Beyond that, due to different cache writeback strategies, Popcorn Linux's page server subsystem (which handles page faults and related distributed shared memory chores) needs to be more aggressive in flushing the page cache on RISC-V processors.

Due to a technical limitation in the RISC-V port of Popcorn Linux (lack of synchronization at migration points), our system currently only supports single-threaded applications. It is however not a fundamental issue, and other research works have demonstrated cross-ISA migration of multi-threaded applications [2, 47].

4 Offloading from x86 to RISC-V CPUs

With the Popcorn Linux software stack extended for the RISC-V architecture, we consider a use case inspired by HEXO [29]: augment high-performance x86 servers with inexpensive RISC-V-based embedded boards, as a way to improve performance and energy consumption at a low cost vs. purchasing additional servers. Olivier et al. [29] connect to an Intel Xeon (x86-64-based) server ARM-based single board computers that are a fraction of the server's price. The study shows that offloading jobs from the server to the boards helps alleviate the server's resource congestion, and can yield as a result throughput increase of 67% and energy efficiency gains of 56% for certain workloads. Since data center costs are dominated by the costs for server acquisition, power consumption, and equipment cooling [9], such gains in performance and energy efficiency obtained by coupling servers and single board computers have a value proposition in reducing capital costs. Therefore, armed with Popcorn Linux's extension for RISC-V, we consider offloading jobs from an x86 server to a RISC-V-based embedded board to improve throughput and energy efficiency.

The intuition behind HEXO's idea is that, although all workloads will suffer a slowdown when executing on the single board computer vs. on the server, that slowdown is highly variable among programs, and for certain jobs it will be inferior to one order of magnitude. To investigate this hypothesis, we conducted two experiments to understand the difference in application execution times and energy consumption on an x86 server (Intel Xeon E5-2637v3) and RISC-V (SiFive HiFive) board. We used the NAS Parallel Benchmarks (NPB) suite [1] which is representative of long-running compute-intensive workloads from the domain of high-performance computing that may run as background jobs in the datacenter. The suite include computation kernels: integer sort (IS), embarrassingly parallel (EP), discrete 3D fast Fourier transform (FT), unstructured adaptive mesh (UA), conjugate gradient (CG), and multi-grid (MG), as well as pseudo applications: block tri-diagonal solver (BT), scalar penta-diagonal solver (SP) and lower-upper Gauss-Seidel solver (LU). A single instance of each benchmark program (serial version) is executed on each machine, and we measure the execution times and energy consumption. We also use a set of memory-intensive micro-benchmarks, that allocate a fixed large amount of memory and then access it in a specific pattern such that each operation results in a cache miss, i.e. a memory fetch. The memory is accessed in step sizes that are factors of the cache line size to create different execution loads. As the step size increases, the number of memory fetches (cache misses) increases, impacting negatively the total execution time. This pattern of accessing the memory is repeated for a fixed number of iterations for all step sizes to increase the overall execution time and generate enough memory load. We consider step sizes of 16, 32, and, 64 bytes, and name the respective micro-benchmarks as *cache_step_16*, *cache_step_32*, and *cache_step_64*.

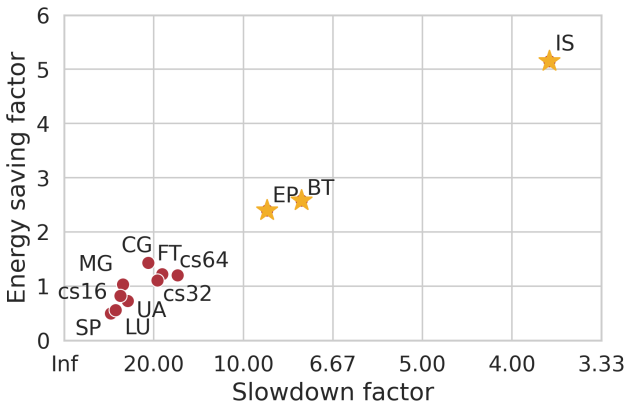

Figure 3: Execution time slowdown and energy savings experienced by the NPB benchmark suite and a set of memory-intensive micro-benchmarks when running on the SiFive RISC-V board vs. on an Intel Xeon E5-2637v3 server. The micro-benchmarks cache_step_16/32/64 are abbreviated as cs16/32/64. Stars represent good candidates for offloading, showing low slowdown and high energy savings.

Figure 3 shows on its X axis the execution time slowdown factor the benchmarks suffer when running on the SiFive board compared to execution on the Xeon server. The overall slowdown range is wide: from 3.6x to 38x. We observe that the slowdown is relatively small for some of the programs: it is less than 10x for IS, BT and EP. Conversely, the slowdown can be very high for certain programs: for SP, LU, cache_step_16, MG and UA it is superior to 20x. This is not surprising as the SiFive board has weaker micro-architectural properties (e.g., lower clock speeds). However, it is important to consider the slowdown range in perspective: the SiFive board costs ($999) only one-third, approximately, of the Xeon server ($3,049). Furthermore, it is a prototype board, and the cost of next-generation RISC-V boards is likely to get much lower due to its royalty free ISA as well as higher manufacturing volumes resulting in economies of scale: in fact, several RISC-V-based single-board computers with prices expected to be below $100 have been announced or are already available [34, 40].

Overall, programs with lower slowdowns are the ones demonstrating the highest energy savings, which can be easily explained by the fact that longer execution times means higher energy consumption. Still, there are a few noteworthy outliers, such as CG: although it has a higher slowdown vs. FT or cache_step_32/64, CG presents higher energy savings than these programs. This may indicate a lower power consumption (in watts) for CG during the entirety or part of its execution on the board, compared to other programs.

The Y axis of Figure 3 shows the energy saved for the benchmarks when running on the SiFive board compared to execution on the Xeon server. Values are normalized to the power consumption when running on the server, i.e. 1 on the y-axis represents the Xeon's energy consumption. Some programs consume significantly less energy on the board: the energy consumption reduction for IS is 5x, and more than 2x for EP and BT. Other programs, among those with high slowdowns, consume more energy on the board: SP, LU, UA, and cache_step_16. This is because the execution time of these programs is so high on SiFive that it offsets the board's lower power consumption.

From these experiments, we can conclude that by attaching such a RISC-V board to a Xeon server and offloading *lower* slowdown applications (which overall are also the ones showing the highest energy savings) to the board, consolidation benefits can be obtained, i.e. higher job throughput or lower power consumption for a fixed workload. Still, because certain jobs present such a high slowdown on the board, offloading them would undoubtedly be detrimental. Hence, offloading needs to be selective, and we need a scheduler.

Integrating low-power RISC-V processors in the datacenter raises the question of the form this integration would take, in terms of form factors, space available in server cabinets, thermal constraints, etc. We consider these issues to be orthogonal to LSF's objective, which is to enable the runtime migration of software between traditional servers and RISC-V machines, independently of the way these are integrated. Still, a possibility for a transparent integration would be to have RISC-V compute nodes in the form of boards present within traditional (e.g. x86) servers, connected through PCI Express (note that RISC-V-based smart I/O devices are already here and connected similarly).

4.1 The LSF Scheduler

The key idea of LSF's scheduler is to assume that all jobs initially start on the server and are monitored there so that the system can build an estimation of the slowdown they would suffer if offloaded to the RISC-V board. LSF estimates the slowdown by monitoring hardware performance counters and using a simple linear regression technique. To assess the efficiency of that approach, we profiled all applications from the NPB benchmark suite on the server, and gathered data from the following performance counters: last level cache misses per second, number of instructions per second, cache loads per second, and number of branches per second. We found a good correlation between the number of instructions per second observed for an application running on the

server, and the slowdown suffered by that application when executing on the board: the correlation coefficient r = 0.80. This indicates a relationship between the slowdown and this metric: by monitoring it, the scheduler can dynamically estimate slowdown information for each job. This differs from HEXO's scheduler [30] that, for an x86 server and ARM64 single board computer setup, used the amount of last level cache references per second to predict slowdown. That choice was explained by the relatively slow memory subsystem of the ARM64 board used in that research effort, something which seems to be less of a problem with the RISC-V machine we use.

LSF works as follows. We assume that LSF has been trained on a set of applications representative of the jobs it will have to schedule at runtime, in our case the NPB benchmarks. We also assume that a particular (set of) performance counter(s) can be found to have a good correlation coefficient with the slowdowns jobs will suffer should they be offloaded from the server to the board: in our case, the number of instructions per second. All jobs start on the server, and when a job begins its execution, LSF checks the server's load in terms of the number of available cores. Due to the compute-intensive and single-threaded nature of the workloads we target, we do not consolidate more than one job per core. If no cores are available on the server, LSF selects a *victim* job to be offloaded to the board. Based on our previous observation, the victim job is one that has the lowest number of instructions per second among the currently running set and has been executing the longest on the server. This effectively means that the scheduler offloads the job that will suffer the lowest estimated slowdown when moved to the RISC-V board.

LSF performs bookkeeping tasks such as tracking the available cores, the currently running set of jobs, and each job's instructions per second, amount of time it has been executing, and on which cores. This information is updated at different scheduling events including job arrival and departure, offloading decisions, and periodic events that notify jobs' performance metrics. When an application has completed its execution, LSF is notified of this event. The system then updates the usage of the servers accordingly and waits for the next event.

LSF's implementation follows an event-based client-server model using socket communication (Figure 4). The Popcorn Linux compiler was modified to include client code in all applications. A userspace server, containing the scheduler code, listens for connections and waits to offload jobs. Once a victim job is selected for offloading, the scheduler notifies it through sockets. The job's client code then sets a shared flag which is checked on the job's next migration event and triggers Popcorn's runtime system for cross-ISA state transformation and execution migration.

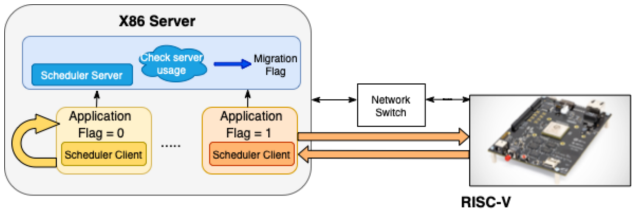

Figure 4: The LSF scheduler's client/server model for offloading jobs in a heterogeneous-ISA system with x86 and RISC-V CPUs. Flag represents the target machine where a job is scheduled for execution.

Table 1: Details of the hardware setup used in evaluation.

Machine	Xeon	SiFive HiFive Unleashed
CPU model	Xeon E5-2637v3	U54 RISC-V
ISA	x86-64	RV64IMAFDC
CPU Frequency	3.5 (Turbo 3.70) GHz	1.5 GHz
Cores	4 (8 HT)	4
RAM	64 GB	8 GB
Power (idle)	60 W	4.15 W
Price	$3049	$999

The design of LSF is driven by simplicity. It enables, as we demonstrate in evaluation, notable gains on the set of CPU/memory-intensive programs we profiled to determine the correlation between application behavior and the slowdown observed on the board vs. on the server. It is likely that more diverse applications or the use of RISC-V boards of various specifications would necessitate different estimation methods. We scope out advanced estimation techniques (e.g. ML-based approaches, or considering specialized workloads such as SIMD-intensive applications), as well as various other scheduling policies (e.g. offloading to the board even if the x86 server is not saturated to save power) and their comparison to LSF as future work.

5 Evaluation

5.1 Methodology

We conducted an experimental study to evaluate LSF's gains on throughput and energy efficiency. Our hardware consists of an Intel Xeon x86 server and a SiFive RISC-V board [39] that are connected using a 1 Gbps Ethernet switch. The specifications of both computers are given in Table 1. Both machines run Popcorn Linux kernel 5.2.21. Our baseline is a single Xeon server, allowing us to quantify LSF's throughput and energy gain for one-third of the capital cost of a second Xeon server. For both machines we measure the entire power consumed by the system with the help of a HOBO Plug Load Data Loggers [31]. To calculate the idle power consumption,

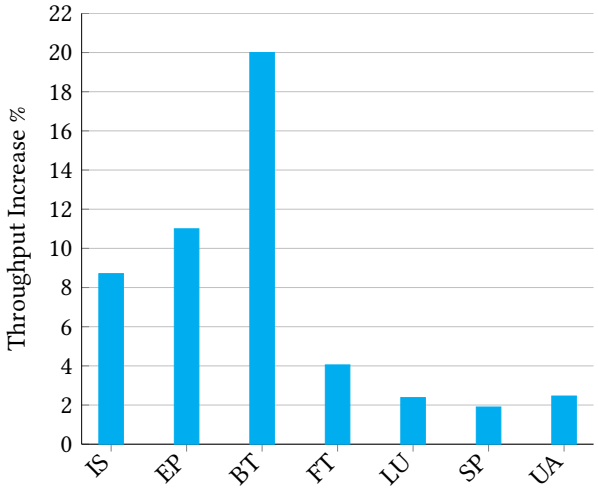

Figure 5: LSF's throughput gain over a single Xeon server with a queue of single job instances.

we measure the average power consumed by the server and the RISC-V board when idle for a period of 10 minutes.

Our workload consists of macro- and micro-benchmarks. Regarding macro-benchmarks, we use the NPB [1] suite (classes A and B). In terms of micro-benchmarks we use the memory-intensive applications *cache_step_16*, *cache_step_32*, *cache_step_64*, previously described in Section 4.

5.2 Job Throughput

We ran two sets of throughput experiments. In the first experiment, we feed the system with an "infinite" queue of jobs, each job being an instance of the same benchmark, and we repeat the experiment varying the benchmark with NPB IS, EP, BT, FT, LU, SP and UA. We ran each experiment for 75 minutes and counted the number of jobs that finished executing, in order to compute the total job throughput. This experiment represents a specific case when job instances with the same slowdown are launched on the server.

Figure 5 shows LSF's throughput increase over the baseline single Xeon server without the capacity to offload jobs. The benchmark BT has the highest throughput increase of 20%. Interestingly, IS's throughput gain is lesser than BT's and EP's, although these have higher slowdowns (see Figure 3). This is because IS and EP have a relatively short execution time and their migration overhead partially offsets the potential performance benefit of offloading when compared to BT. The other benchmarks have gains proportional to their slowdowns. In particular, SP has the lowest throughput gain of 1.9%, due to its high slowdown when executing on the board (more than 35x, see Figure 3).

Table 2: Consolidated set of benchmark programs.

Set characteristics	Set	Benchmark composition	NPB Class
Lowest Slowdown (<11%)	low-A	NPB BT, EP, IS	A
	low-B		B
Medium Slowdown (11% - 21%)	med-A	NPB FT, cache_step_32, cache_step_64	A
	med-B		B
Highest Slowdown (>21%)	high-A	NPB CG, LU, MG, SP, UA, cache_step_16	A
	high-B		B
Mixed set of low and high slowdown applications	mix-A	NPB BT, NPB SP	A
	mix-B		B
NPB entire suite	npb-A	NPB BT, CG, EP, FT, IS, LU, MG, SP, UA	A
NPB entire suite + Micro Benchmarks	full-A	NPB BT, CG, EP, FT, IS, LU, MG, SP, UA, cache_step_16, cache_step32, cache_step_64	A
	full-B		B

In our second experiment, the queue we feed to the system now contains a mix of NPB programs (both classes A and B) and of the previously presented micro-benchmarks. Table 2 shows the individual queues (sets) composition. These sets were carefully constructed to consolidate different slowdown ranges:

- Sets *low-A* and *low-B* contain programs with less than 10% slowdown (LSF's best case).
- Sets *mid-A* and *mid-B* contain programs with 11%-20% slowdown (a "middle case" for LSF).
- Sets *high-A* and *high-B* contain programs with greater than 21% slowdown (LSF's worst case).
- Sets *mix-A* and *mix-B* contain programs with the best and worst slowdowns.
- We also defined three other sets, *npb-A*, *full-A*, and *full-B* that included all the micro- and macro-benchmark programs.

To measure the job throughput, we ran the sets containing programs from the A class of NPB (low-/med-/high-/mix-/npb-/full-A) for 75 minutes. The sets containing programs from the B class (low-/med-/high-/mix-/full-B) were run for 480 minutes as their execution time is longer. By increasing the experimental duration, we ensure that LSF can migrate multiple sets of applications to the SiFive board.

Figure 6 shows the throughput gain for the consolidated sets. Set low-A has a gain of 19.5% and low-B has a gain of 20%, which is similar to the throughput gain for an infinite queue of BT-only instances (Figure 5). When combining the best and worst case (BT and SP) for both classes A and B, LSF's gain is 16.6%, which is less than BT-only's throughput

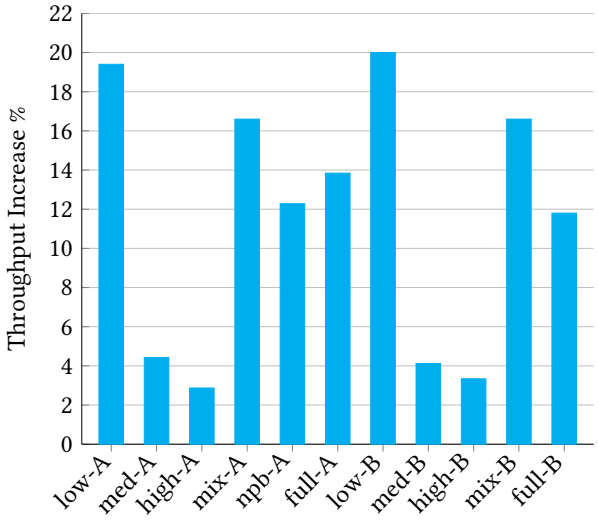

Figure 6: LSF's throughput gain over a single Xeon server with a queue of consolidated sets.

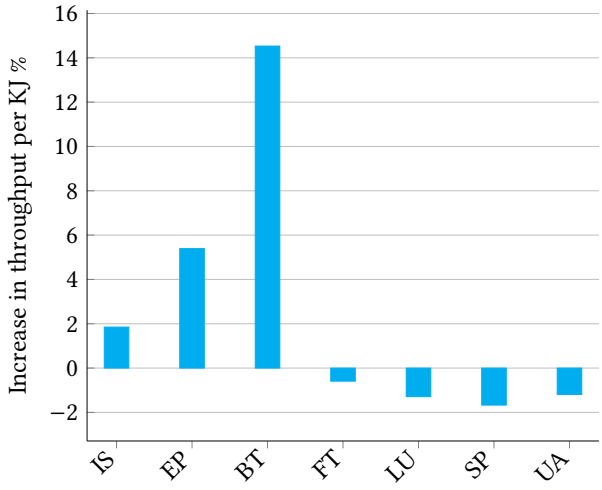

Figure 7: LSF's energy efficiency gain over a single Xeon server with a queue of single job instances.

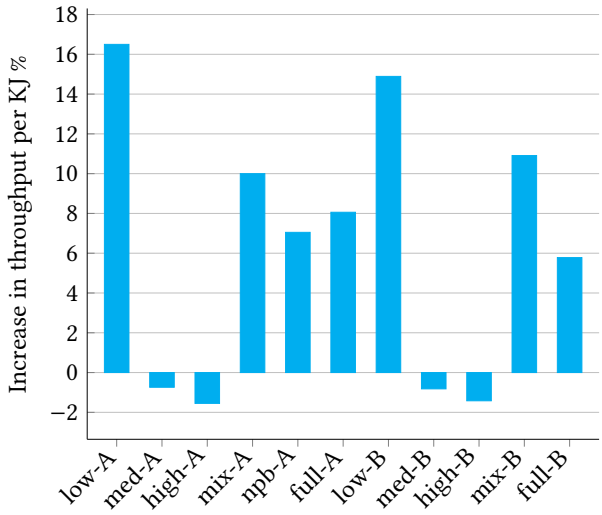

Figure 8: LSF's energy efficiency gain over a single Xeon server with a queue of consolidated sets.

increase. This is because when the jobs are first launched and no profiling information is available, some of the SP instances are migrated to the SiFive board.

Sets med-A and med-B represent the middle case and have a gain of 4.4% and 2.8%, respectively. Sets high-A and high-B which represent LSF's worst case with the highest slowdown programs, have throughput gains of 4.12% and 3.34%, respectively. These sets have a higher throughput gain than SP-only's, as they also contain programs with slowdown less than SP's that are more likely to be offloaded to the SiFive board.

Sets npb-A, full-A, and full-B have gains of 12.28%, 13.84%, and 11.8%, respectively. The overall throughput is less than the combination of best and worst case. This is because BT has the longest time for execution during which no information about its instructions per second metric is available. Therefore, LSF offloads programs with the lowest instructions per second metric available during that time. Overall, sets with NPB/class A and sets with NPB/class B show a similar trend.

5.3 Energy Efficiency

We also measured SiFive's and Xeon's energy consumption for the experiments which throughput numbers are given in Figures 5 and 6. For each set, we calculated the *energy efficiency* by dividing the total number of programs that completed their execution during the evaluation time interval (75 minutes) by the total energy consumed by the machines (in kilo-joules), respectively, during the time interval. This gives us the number of jobs completed per kilo-joule.

The energy savings for the job queues made of single job instances are presented on Figure 7, and the savings for consolidated mixes of jobs are presented on Figure 8. From Figure 7, we observe that LSF's increase in energy efficiency is less than its throughput increase. This is largely

due to high-slowdown programs, i.e., those with a slowdown higher than 10x (e.g., FT, LU, SP, UA, cache_step_16). Their large execution times offset SiFive's low power consumption (SiFive's idle power is 4.15W compared to Xeon's 60W) and they end up consuming more energy than when running on Xeon. Still, some queues leads to energy savings, e.g. more than 14% for BT, and 5% for EP.

From Figure 8, we observe that sets low-A and low-B with the lowest-slowdown programs have a 16.5% and 14.9% energy efficiency gains, respectively, which is higher than BT-only's. This is because these sets also contain other programs which consume relatively less energy on the Xeon than BT. For sets high-A and high-B with the highest-slowdown programs, the energy efficiency gain is better than SP-only's as the energy efficiency loss is partially offset by other programs in the set. The sets mix-A and mix-B have a relatively high energy efficiency. This is because these sets execute a high ratio of BT (vs. the other program in the set, SP), as this program present a very low slowdown on the board. This results overall in high energy efficiency for mix-A and mix-B.

The sets npb-A containing all of NPB/class A, and full-A containing all the micro- and macro-benchmarks (class A) follow a similar trend: full-A has a higher energy efficiency gain than npb-A as it executes a higher ratio of programs with better energy efficiency. The set full-B containing all the micro- and macro-benchmarks (class B) has an energy efficiency gain of 5.78%. Similar to full-A, it is lower than the best case for class B applications and the set high-A containing BT and SP applications.

In summary, our evaluation shows that LSF can achieve throughput gains of up to 20%, as well as energy efficiency gains of up to 16%. We consider these gains significant as the cost of a SiFive board is only one-third of Xeon's. We anticipate future gains to be even more significant as the next generations of RISC-V boards are likely to cost even less (due to higher manufacturing volumes), to be more power efficient, and to embed network cards which speed is on par with that of modern datacenter networking equipment (the HiFive board's NIC has a speed of 1 Gbps).

6 Related Work

Past and related works in LSF's problem space can be broadly classified into two categories: a) scheduling in single-ISA heterogeneous systems and b) scheduling in multi-ISA heterogeneous systems.

6.1 Single-ISA Heterogeneous Scheduling

The problem of single-ISA heterogeneous scheduling has been extensively studied, in part due to ubiquitous commodity-scale hardware in that category such as the ARM big.LITTLE architecture. The common scheduling principle of approaches in this category is to estimate an application's performance on one class of cores using its performance (or other metrics) on another class of cores and use that estimate to guide scheduling decisions. For example, a typical approach is to predict an application's slowdown when executed on the "wimpier" core using its observed performance metrics on the "beefier" core and use that estimate to decide what threads to execute on which cores. For instance, Koufaty et al. [22] propose bias scheduling, an approach that characterizes an application's slowdown using the internal and external stalls faced by its threads. Such information is then used to match applications to cores with appropriate performance and micro-architectural properties. Van Craeynest et al. [41] present with fairness-aware scheduling a proportional fair scheduling approach, wherein the scheduler balances threads by giving equal time to all threads on a core, which indirectly ensures that all the threads experience similar degrees of slowdown.

Petrucci et al. [33] present lucky scheduling, a lottery-based approach originally developed by Waldspurger and Weihl [44], wherein an application's resource usage is monitored using performance counters such as instructions per second and LLC misses, which are then used to proportionally allocate "tickets": the larger the number of its tickets, the greater is the application's chance of executing on a beefier core. Jibaja et al. [17] present the WASH scheme, a scheduling algorithm that is trained offline using performance metrics to predict slowdowns. At runtime, the predicted slowdown and other monitored performance data such as criticality, thread sensitivity, and priority are used to guide scheduling decisions. Scheduling approaches that use applications' memory and cache contention to make scheduling decisions have also been studied [7, 11, 15, 23].

6.2 Multi-ISA Heterogeneous Scheduling

Scheduling in Multi-ISA heterogeneous systems has received relatively less attention. Barbalace et al. [3] first presented Popcorn Linux for fully non-overlapping ISAs, presenting two scheduling policies. Unlike the principle of predicting application performance and using that prediction to guide scheduling decisions, these policies use a simpler approach. The first policy balances the number of threads on both machines (i.e., x86 and ARM), while the second policy runs a greater number of threads on the beefier machine (i.e., x86). In contrast, Karaoui et al. [19] use a slowdown-based scheduling policy: high-slowdown applications are placed on beefier cores and low-slowdown ones on wimpier cores, and applications are migrated from wimpier to beefier cores when the latter's queue becomes empty. Pang et al. [32] divide an application into different regions depending on their

slowdown, and assign regions to cores depending on the slowdown faced when executing on that core. Both Karaoui et al. [19] and Pang et al. [32] assume knowledge of application slowdown before execution.

LSF represents an adaptation of the principles defined by HEXO's scheduler [29], predicting an application's slowdown on an ARM-based embedded board using last-level cache references per second observed on an x86 server, which is shown to have a high degree of correlation with the predicted slowdown (correlation coefficient, $r = 0.95$). Applications with (predicted) low slowdown are migrated to the board in priority. Additionally, HEXO's scheduler considers factors such as available memory and cores on the embedded board in deciding which jobs to offload to the board. Contrary to HEXO's scheduler, LSF predicts an application's slowdown on the RISC-V board using the number of instructions per second observed on the x86 server.

Past work has also investigated how the quality of machine learning (ML)-based performance predictors affects scheduler's performance, in both single-ISA and ISA-heterogeneous systems. Note that both LSF and the HEXO scheduler can be viewed as using ML-based performance predictors, although they are simple linear regression models. Prodromou et al. [35] conduct a quantitative evaluation and show that some schedulers exhibit large tolerance in prediction error. Moreover, they identify that the processor's degree of diversity, including both ISA and microarchitectural properties, has the most significant influence on the level of prediction error that schedulers can tolerate.

7 Conclusion

This paper explores the idea of pairing low-power RISC-V as co-processors to energy-hungry x86 CPUs processing long-running stateful background jobs in the datacenter, with the goal of improving performance and energy footprint. To evaluate this idea we update and port the Popcorn Linux kernel and LLVM compiler to RISC-V. We further introduce the Lowest-Slowdown-offloaded-First Heterogeneous-ISA scheduler (LSF), driving the migration of long-running compute-intensive workloads *at runtime* between a RISC-V CPU and an x86 CPU. This is achieved by predicting the job performance on the RISC-V embedded board and migrating the applications which face the least amount of slowdown when executing on the RISC-V itself. We use a SiFive RISC-V board and an Intel Xeon E5-2367 x86 server to evaluate the LSF scheduler. Our evaluations show that LSF achieves throughput gains of up to 20% while getting energy efficiency of 16%. The cost of RISC-V single-board computers is expected to decrease in the future as economies of scale start to kick in, and we are hopeful that this will strengthen the benefits of LSF.

Perspectives of future work include running with LSF more diverse applications, and studying advanced slowdown prediction methods on such programs. Further, as the LSF scheduler uses one of the first-generation RISC-V boards, with SiFive and other vendors releasing new RISC-V boards and servers, investigating the throughput gain and energy efficiency on such newer hardware and designing "portable" schedulers Jibaja et al. [18] is another promising direction. Another interesting future research area relates to the concept of ISA affinity: past research showed that phases of a program can be identified as performing better on different ISAs [10, 42]. In that context, LSF could be used to schedule tasks on the optimal ISA at phase boundaries.

This work is available online under an open source license at the following URL: http://popcornlinux.org/.

Acknowledgments

We thank the anonymous reviewers and our shepherd Orna Agmon Ben-Yehuda for their insightful comments and suggestions, which helped to greatly improve the paper.

This work is supported in part by the US Office of Naval Research (ONR) under grants N00014-18-1-2022, N00014-19-1-2493, and N00014-22-1-2672 and the US Naval Surface Warfare Center Dahlgren Division under grant N00174-20-1-0009.

References

[1] David Bailey, E. Barszcz, Barton J.T, Browning D.S, Carter R.L, Dagum D, Fatoohi R.A, Paul Frederickson, Lasinski T.A, Robert Schreiber, Horst Simon, Venkat Venkatakrishnan, and Weeratunga K. 1991. The Nas Parallel Benchmarks. *International Journal of High Performance Computing Applications* 5 (09 1991), 63–73. https://doi.org/10.1177/109434209100500306

[2] Antonio Barbalace, Mohamed L. Karaoui, Wei Wang, Tong Xing, Pierre Olivier, and Binoy Ravindran. 2020. Edge computing: the case for heterogeneous-ISA container migration. In *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Lausanne, Switzerland) *(VEE '20)*. Association for Computing Machinery, New York, NY, USA, 73–87. https://doi.org/10.1145/3381052.3381321

[3] Antonio Barbalace, Robert 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 Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 645–659. https://doi.org/10.1145/3037697.3037738

[4] Antonio Barbalace, Robert Lyerly, Christopher Jelesnianski, Anthony Carno, Ho-Ren Chuang, Vincent Legout, and Binoy Ravindran. 2017. Breaking the Boundaries in Heterogeneous-ISA Datacenters. *SIGARCH Comput. Archit. News* 45, 1 (April 2017), 645–659. https://doi.org/10.1145/3093337.3037738

[5] Antonio Barbalace, Binoy Ravindran, and David Katz. 2014. Popcorn: a replicated-kernel OS based on Linux. http://popcornlinux.org/images/publications/barbalace_ols.pdf.

[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 Tenth European Conference on Computer Systems* (Bordeaux, France) *(EuroSys '15)*. Association for Computing Machinery, New York, NY, USA, Article 29, 16 pages. https://doi.org/10.1145/2741948.2741962

[7] Michela Becchi and Patrick Crowley. 2006. Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures. In *Proceedings of the 3rd Conference on Computing Frontiers* (Ischia, Italy) *(CF '06)*. Association for Computing Machinery, New York, NY, USA, 29–40. https://doi.org/10.1145/1128022.1128029

[8] 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* (Monterey, California) *(HotPower '15)*. Association for Computing Machinery, New York, NY, USA, 6–10. https://doi.org/10.1145/2818613.2818747

[9] Ricardo Bianchini. 2017. Improving Datacenter Efficiency. *SIGARCH Comput. Archit. News* 45, 1 (apr 2017), 327. https://doi.org/10.1145/3093337.3046426

[10] Nirmal Kumar Boran, Dinesh Kumar Yadav, and Rishabh Iyer. 2020. Classification based scheduling in Heterogeneous ISA Architectures. In *2020 24th International Symposium on VLSI Design and Test (VDAT)*. IEEE, 1–6. https://doi.org/10.1109/VDAT50263.2020.9190559

[11] Jian Chen and Lizy K. John. 2009. Efficient program scheduling for heterogeneous multi-core processors. In *2009 46th ACM/IEEE Design Automation Conference*. IEEE Computer Society, USA, 927–930.

[12] Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beránek, Luca Benini, and Torsten Hoefler. 2021. A RISC-V in-network accelerator for flexible high-performance low-power packet processing. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 958–971. https://doi.org/10.1109/ISCA52012.2021.00079

[13] Western Digital. 2019. RISC-V SweRV CoreTM Available to Open Source Community. https://blog.westerndigital.com/risc-v-swerv-core-open-source/

[14] Ulrich Drepper. 2005. *Elf handling for thread-local storage*. Technical Report. Technical report, Red Hat. https://uclibc.org/docs/tls.pdf

[15] Xiaokang Fan, Yulei Sui, and Jingling Xue. 2015. Contention-Aware Scheduling for Asymmetric Multicore Processors. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE Computer Society, USA, 742–751. https://doi.org/10.1109/ICPADS.2015.98

[16] Peter Garraghan, Paul Townend, and Jie Xu. 2013. An analysis of the server characteristics and resource utilization in google cloud. In *2013 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 124–131.

[17] Ivan Jibaja, Ting Cao, Stephen M. Blackburn, and Kathryn S. McKinley. 2016. Portable Performance on Asymmetric Multicore Processors. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization* (Barcelona, Spain) *(CGO '16)*. Association for Computing Machinery, New York, NY, USA, 24–35. https://doi.org/10.1145/2854038.2854047

[18] Ivan Jibaja, Ting Cao, Stephen M. Blackburn, and Kathryn S. McKinley. 2016. Portable performance on Asymmetric Multicore Processors. In *2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE Computer Society, USA, 24–35.

[19] Mohamed L. Karaoui, Anthony Carno, Rob Lyerly, Sang-Hoon Kim, Pierre Olivier, Changwoo Min, and Binoy Ravindran. 2019. Scheduling HPC Workloads on Heterogeneous-ISA Architectures: Poster. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming* (Washington, District of Columbia) *(PPoPP '19)*. Association for Computing Machinery, New York, NY, USA, 409–410. https://doi.org/10.1145/3293883.3295717

[20] David Katz, Antonio Barbalace, Saif Ansary, Akshay Ravichandran, and Binoy Ravindran. 2015. Thread Migration in a Replicated-Kernel OS. In *2015 IEEE 35th International Conference on Distributed Computing Systems*. 278–287. https://doi.org/10.1109/ICDCS.2015.36

[21] Sang-Hoon Kim, Ho-Ren Chuang, Robert Lyerly, Pierre Olivier, Changwoo Min, and Binoy Ravindran. 2020. DeX: Scaling Applications Beyond Machine Boundaries. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society Press, Washington, DC, USA, 864–876. https://doi.org/10.1109/ICDCS47774.2020.00021

[22] David Koufaty, Dheeraj Reddy, and Scott Hahn. 2010. Bias Scheduling in Heterogeneous Multi-Core Architectures. In *Proceedings of the 5th European Conference on Computer Systems* (Paris, France) *(EuroSys '10)*. Association for Computing Machinery, New York, NY, USA, 125–138. https://doi.org/10.1145/1755913.1755928

[23] Luca Lugini, Vinicius Petrucci, and Daniel Mossé. 2012. Online Thread Assignment for Heterogeneous Multicore Systems. In *2012 41st International Conference on Parallel Processing Workshops*. 538–544. https://doi.org/10.1109/ICPPW.2012.73

[24] Robert Lyerly, Antonio Barbalace, Christopher Jelesnianski, Vincent Legout, Anthony Carno, and Binoy Ravindran. 2016. Operating System Process and Thread Migration in Heterogeneous Platforms. http://www.cs.utexas.edu/~mars2016/workshop-program/ The 2016 Workshop on Multicore and Rack-scale Systems, MaRS 2016 ; Conference date: 18-04-2016 Through 18-04-2016.

[25] Robert Lyerly, Changwoo Min, Christopher J. Rossbach, and Binoy Ravindran. 2020. An OpenMP Runtime for Transparent Work Sharing Across Cache-Incoherent Heterogeneous Nodes. In *Proceedings of the 21st International Middleware Conference* (Delft, Netherlands) *(Middleware '20)*. Association for Computing Machinery, New York, NY, USA, 415–429. https://doi.org/10.1145/3423211.3425679

[26] Javier Malave. 2020. Popcorn Linux Distributed Thread Execution. RFC post on LKML, https://lkml.org/lkml/2020/4/29/1111.

[27] Asit K Mishra, Joseph L Hellerstein, Walfredo Cirne, and Chita R Das. 2010. Towards characterizing cloud backend workloads: insights from google compute clusters. *ACM SIGMETRICS Performance Evaluation Review* 37, 4 (2010), 34–41.

[28] Pierre Olivier, Sang-Hoon Kim, and Binoy Ravindran. 2017. OS Support for Thread Migration and Distribution in the Fully Heterogeneous Datacenter. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems* (Whistler, BC, Canada) *(HotOS '17)*. Association for Computing Machinery, New York, NY, USA, 174–179. https://doi.org/10.1145/3102980.3103009

[29] Pierre Olivier, A. K. M. Fazla Mehrab, Stefan Lankes, Mohamed Lamine Karaoui, Robert Lyerly, and Binoy Ravindran. 2019. HEXO: Offloading HPC Compute-Intensive Workloads on Low-Cost, Low-Power Embedded Systems. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing* (Phoenix, AZ, USA) *(HPDC '19)*. Association for Computing Machinery, New York, NY, USA, 85–96. https://doi.org/10.1145/3307681.3325408

[30] Pierre Olivier, A. K. M. Fazla Mehrab, Stefan Lankes, Mohamed Lamine Karaoui, Robert Lyerly, and Binoy Ravindran. 2019. HEXO: Offloading HPC Compute-Intensive Workloads on Low-Cost, Low-Power Embedded Systems. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing* (Phoenix, AZ, USA) *(HPDC '19)*. Association for Computing Machinery, New York, NY, USA, 85–96. https://doi.org/10.1145/3307681.3325408

[31] ONSET. 2014. HOBO Plug Load Data Logger UX120-018. https://www.onsetcomp.com/products/data-loggers/ux120-018/.

[32] Yihan Pang, Robert Lyerly, and Binoy Ravindran. 2019. Cross-ISA Execution of SIMD Regions for Improved Performance. In *Proceedings of the 12th ACM International Conference on Systems and Storage* (Haifa, Israel) *(SYSTOR '19)*. Association for Computing Machinery, New York, NY, USA, 55–67. https://doi.org/10.1145/3319647.3325832

[33] Vinivius Petrucci, Orlando Loques, and Daniel Mosse. 2012. Lucky Scheduling for Energy-Efficient Heterogeneous Multi-Core Systems. In *Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems*. USENIX Association, Hollywood, CA, 7. https://www.usenix.org/conference/hotpower12/workshop-program/presentation/Petrucci

[34] Pine64. 2023. Pine64 Wiki: Star64. https://wiki.pine64.org/wiki/STAR64.

[35] Andreas Prodromou, Ashish Venkat, and Dean M. Tullsen. 2019. Deciphering Predictive Schedulers for Heterogeneous-ISA Multicore Architectures. In *Proceedings of the 10th International Workshop on Programming Models and Applications for Multicores and Manycores* (Washington, DC, USA) *(PMAM'19)*. Association for Computing Machinery, New York, NY, USA, 51–60. https://doi.org/10.1145/3303084.3309492

[36] Marina Sadini, Antonio Barbalace, Binoy Ravindran, and Francesco Quaglia. 2013. A Page Coherency Protocol for Popcorn Replicated-kernel Operating System. http://www.mscs.mu.edu/~brylow/SPLASH-MARC-2013/ Many-Core Architecture Research Community (MARC) Symposium
at SPLASH 2013, MARC 2013 ; Conference date: 28-10-2013 Through 28-10-2013.

[37] Samsung. 2019. Samsung SmartSSD - Samsungsemiconductor-us.com. https://samsungsemiconductor-us.com/smartssd-archive/pdf/SmartSSD_ProductBrief_13.pdf

[38] Seagate. 2023. RISC-V Enables a System on a Chip. https://www.seagate.com/gb/en/innovation/risc-v/

[39] SiFive. 2018. SiFive HiFive Unleashed. https://www.sifive.com/boards/hifive-unleashed

[40] StarFive. 2023. StarFive VisionFive 2. https://www.starfivetech.com/en/site/boards.

[41] Kenzo Van Craeynest, Shoaib Akram, Wim Heirman, Aamer Jaleel, and Lieven Eeckhout. 2013. Fairness-Aware Scheduling on Single-ISA Heterogeneous Multi-Cores. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques* (Edinburgh, Scotland, UK) *(PACT '13)*. IEEE Press, 177–188.

[42] Ashish Venkat and Dean M. Tullsen. 2014. Harnessing ISA Diversity: Design of a Heterogeneous-ISA Chip Multiprocessor. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture* (Minneapolis, Minnesota, USA) *(ISCA '14)*. IEEE Computer Society Press, Washington, DC, USA, 121–132.

[43] David G. von Bank, Charles M. Shub, and Robert W. Sebesta. 1994. A Unified Model of Pointwise Equivalence of Procedural Computations. *ACM Trans. Program. Lang. Syst.* 16, 6 (nov 1994), 1842–1874. https://doi.org/10.1145/197320.197402

[44] Carl A. Waldspurger and William E. Weihl. 1994. Lottery Scheduling: Flexible Proportional-Share Resource Management. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation* (Monterey, California) *(OSDI '94)*. USENIX Association, USA, 1–es.

[45] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanovi. 2014. *The risc-v instruction set manual. volume 1: User-level isa, version 2.0.* Technical Report. California Univ Berkeley Dept of Electrical Engineering and Computer Sciences.

[46] Xilinx. 2020. Xilinx Alveo U25 SmartNIC Platform Launch. https://www.xilinx.com/publications/product-briefs/xilinx-smartNIC-media-briefing-final.pdf

[47] Tong Xing, Antonio Barbalace, Pierre Olivier, Mohamed L. Karaoui, Wei Wang, and Binoy Ravindran. 2022. H-Container: Enabling Heterogeneous-ISA Container Migration in Edge Computing. *ACM Trans. Comput. Syst.* 39, 1–4, Article 5 (jul 2022), 36 pages. https://doi.org/10.1145/3524452