Lowest-Slowdown-offloaded-First Heterogeneous-ISA Scheduler

Balvansh Heerekar

Report submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Engineering
in
Computer Engineering

Dr. Binoy Ravindran, Chair
Dr. Ali R. Butt
Dr. Ruslan Nikolaev

February 17, 2022
Blacksburg, Virginia

Keywords: System Software, Heterogeneous ISA, Scheduling, RISC-V

Lowest-Slowdown-offloaded-First Heterogeneous-ISA Scheduler

Balvansh Heerekar

(ABSTRACT)

The end of Moore's Law has brought a significant change in the architecture of servers used in data centers. ARM, FPGA, and RISC-V processing cores are increasingly incorporated in NICs/SSDs along with host x86 CPUs. Lately, embedded boards have become increasingly OS-capable and can run Linux applications while consuming lesser power than servers and at significantly low equipment costs. Previous works have demonstrated that offloading applications during run-time from servers to embedded boards can result in increases in throughput and energy efficiency. The RISC-V architecture has recently gained significant commercial interest, and OS-capable embedded boards with RISC-V cores are increasingly available at the commodity-scale. In this project, we show that offloading jobs from an x86 server to a RISC-V embedded board at run-time can also yield significant throughput and energy benefits. Towards this, we propose an application job scheduler called **LSF** that offloads jobs from an x86 server to a RISC-V embedded board to alleviate workload congestion on the server. LSF monitors job performance on the server and predicts its performance on the board, which is then used to guide offloading decisions. We implement the LSF scheduler in the Popcorn Linux software stack, and evaluate it using a heterogeneous-ISA system consisting of an Intel Xeon server and a SiFive HiFive Unleashed RISC-V embedded board. Our evaluations reveal that **LSF** yields upto 20% increase in throughput while also gaining 16% more energy efficiency for compute-intensive workloads.

Dedication

This is dedicated to my family.

Acknowledgments

I would like to thank Dr. Binoy Ravindran for being patient with me and giving me a chance to learn and grow. Thank you for always being supportive and pushing me towards success. This work wouldn't have been possible without your constant and unending support. I learnt a lot being a part of the SSRG family.

I would like to also the members of my committee Dr. Ali R. Butt and Dr. Ruslan Nikolaev for their guidance and support.

Words cannot describe how thankful I am to my friends Naga Sanjana, Vaibhav Sundharam and Aayush Sureshchander for their support and love during the tough times.

Furthermore, I would like to thank Ho-Ren (Jack) Chuang for acting like an elder brother to me, being the guiding light and help me navigate my way through various difficulties over the last two years.

Finally, I like to thank my parents and sister for their never ending moral support and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For decades Moore's law has been the fundamental guiding force in the design of computer chips. The faster hardware developed each year provided improvements independently of the software hence allowing the hardware and software to grow more independently. However, increasingly cramming more number of transistors in a small area has brought issues such as power consumption, heat dissipation, increasing manufacturing costs and limitation in shrinking transistors to fit on smaller sizes. The key takeaway from this trend is that increasing the number of transistors may not be the continuing solution for increases in performance. In response to this trend, chip vendors have designed and manufactured vastly different chip designs, including multicore and heterogeneous computing architectures.

The landscape on heterogeneous be broadly divided into three categories depending on how cores are paired with host. GPU is used as an accelerator to perform massively parallel, single-instruction/multiple data (SIMD)-style computations [23][30]. The second category includes single-ISA heterogeneous architectures. First proposed by Kumar [20], ISA heterogeneous multicore machines [2][24] consist of multiple cores of the same ISA but with different microarchitectural properties, i.e., clock speeds, cache sizes, pipeline stages. This design is helpful in getting performance gains when running diverse application workloads by dividing and matching applications with similar performance demands as the underlying cores. For example, running performance-hungry applications on microarchitecturally "beefier" cores and switching to microarchitecturally "wimpier" cores for power-sensitive applications can yield

overall gains in both performance and energy. The third category includes heterogeneous-ISA machines. Unlike the previous two designs, these machines have heterogeneity at the ISA level, such as cores of the CISC ISA family (e.g., x86-64) and the RISC ISA family (e.g., ARM64) integrated together in the same hardware. There has not been a commodity shared memory machine with ISA-heterogeneous cores. However, numerous examples of heteregeneous-ISA machines with physically discrete memory exists. An exemplar case is "smart" I/O devices such as SmartNICs [39] and SmartSSDs [31] that include ARM and RISC-V CPUs, which when attached to the PCIe slot of a high-performance x86-based server yields a heterogeneous-ISA machine with discrete memory (for ISA-heteregeneous cores). In addition, the research community has done much work to understand the performance and energy gains that can be achieved by using heterogeneous ISA systems, [4][36][25][27][22] [18] recent works have also demonstrated that migrating applications from high-performance x86 CPUs to low-performance and low-power ARM CPUs can yield higher throughput with better energy efficiency [25]

1.1 Motivation

Hardware resource sharing is a common practice used in datacenters to improve overall resource utilization and reduce capital costs. Factors that drive up datacenter costs are acquisition of servers, power consumption, and cooling of equipment [7]. Datacenter providers are constantly on the lookout for new methods that could be used to lower the price of deployment while not compromising on performance and the research community is responding in kind. An interesting research direction in this space is to augment high performance servers with low-cost embedded boards, instead of buying additional servers, as a way to improve performance and energy at low cost. Embedded boards that are being produced lately have

become more OS capable and have also become highly efficient while being available at very low price points. Olivier et al. [25] show that by connecting ARM-based embedded boards to an Intel Xeon (x86-based) server that are a fraction of the server's price and offloading jobs from the server to the boards to alleviate server's resource congestion can yield throughput increase of upto 67% and energy efficiency of 56%.

RISC-V is a new ISA that is open source and royalty free [38]. It has a set of fixed instructions and any custom instructions required by manufacturers can be added as extensions. While RISC-V is based on the same RISC principles as ARM, it has recently gained traction due to it's open source nature. The cost of producing boards with ARM processors have become expensive due to the cost associated with procuring the IP from ARM for production. In the case of RISC-V manufacturers would only pay the cost of manufacturing the processors since the ISA is open source and thereby reduce the cost of RISC-V processor. This has caused significant interest in RISC-V by the industry. For example, many Smart I/O device vendors, especially SmartSSD manufacturers, are incorporating RISC-V-based CPUs in their newer SSD products [31]. Given this trend, we are motivated to investigate how a RISC-V based board when coupled with x86 servers may increase the overall throughput and reduce energy consumption, similar to Olivier et. al [25].

To support this motivation, we conducted two experiments to understand the difference in application execution times on a x86 server and RISC-V board as well as the energy consumption difference. For these experiments we used an Intel Xeon E5-2637 x86 processor and the SiFive HiFive RISC-V board. We selected benchmarks from the NAS Parallel Benchmark suite [3], as representative of long-running compute-intensive applications. We ran a single instance of the serially-compiled Class A workload of this benchmark on each of the machines at a time and compared their execution times and energy consumption.

The results of the slowdown experienced by an application when running on the RISC-V

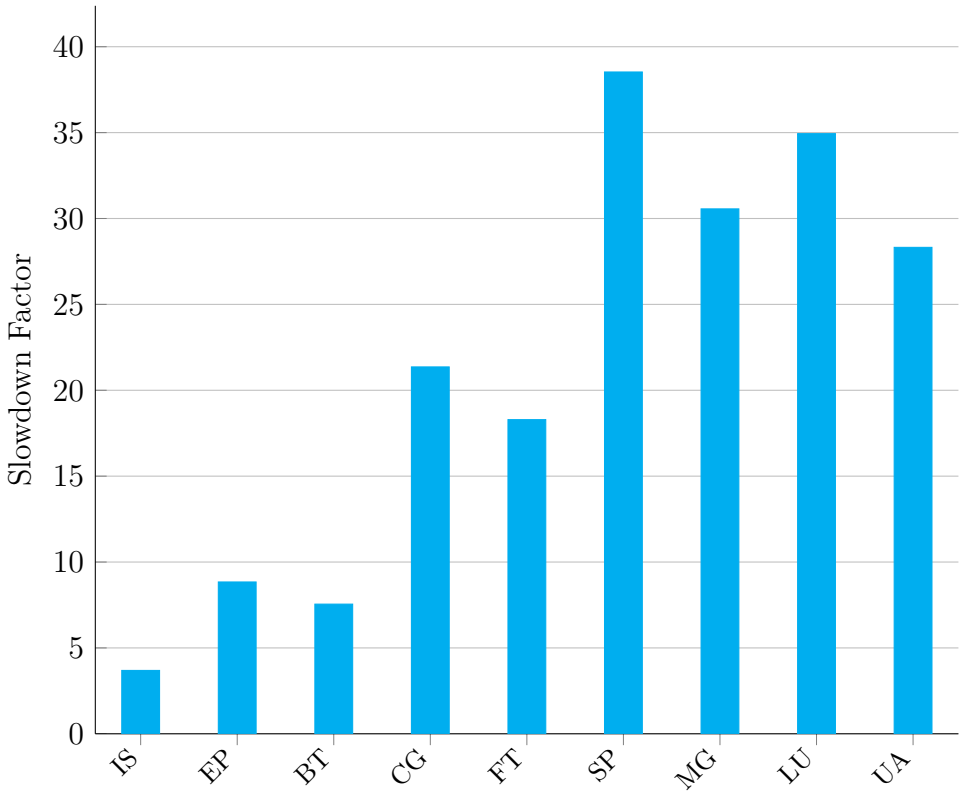

Figure 1.1: Application Slowdown on SiFive RISCV Board vs 4-Core Intel Xeon E5-2637Serve

machine are shown in the Figure 1.1. The x-axis shows the benchmarks being run and the y-axis shows their corresponding slowdowns. The figure reveals that all the applications are slower on the RISC-V machine but differ in the degrees of slowdown they experience. It is worth noting that the Xeon core is clocked faster than the RISC-V core.

RISC-V's idle power is 4.15 watts, whereas x86's is 60 watts, we also investigated the energy saved when an application runs on the RISC-V machine. The energy consumed by each of the NPB application is calculated by using the HOBO Power Data Logger [26]. Figure 1.2

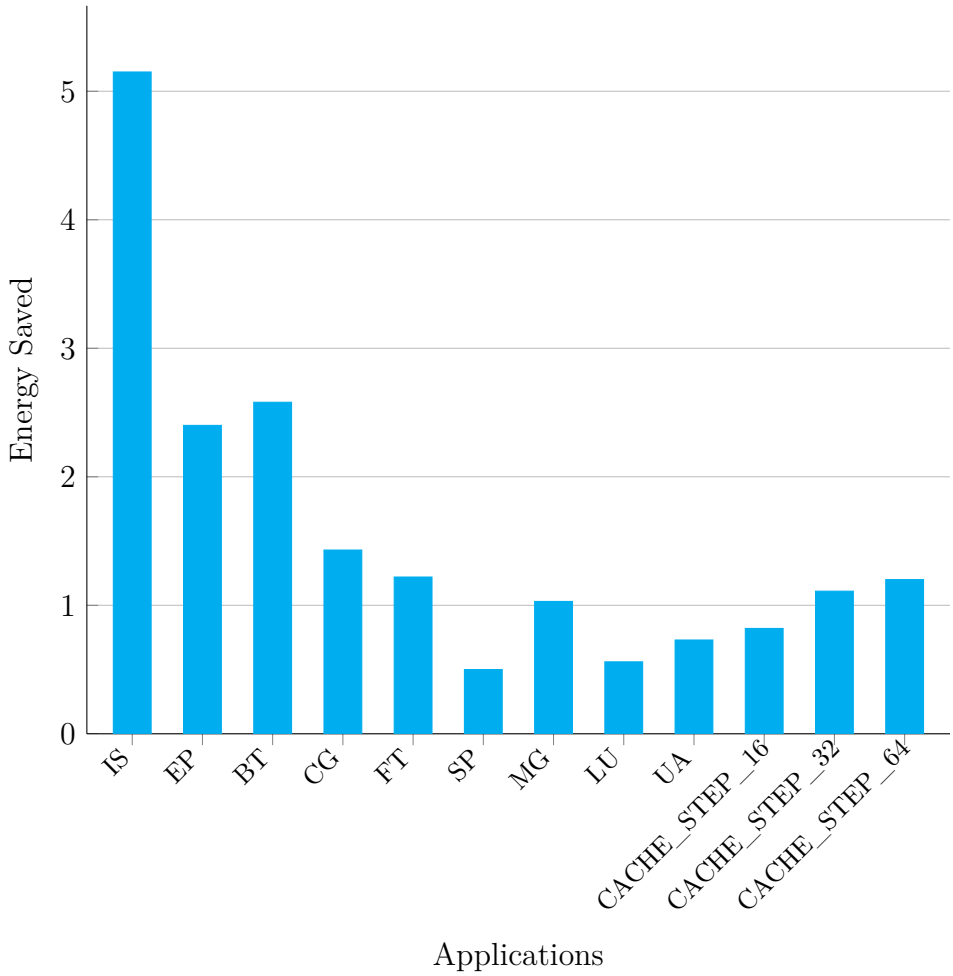

Figure 1.2: Power Saved when running on RISCV vs 4-core Xeon

shows the energy saved when an application executes on the SiFive board. The x-axis show the application and the y-axis shows the energy saved. We can observe that the factor energy saved by each of the applications is different. Applications that face higher slowdowns when running on the SiFive RISC-V board also consume more energy than the Intel Xeon server. This is because the execution time of applications on the RISC-V is so high that is offsets the low energy consumption of the SiFive RISC-V board. Hence when considering energy efficiency it is beneficial to run an application that has low slowdown on the RISC-V board.

Traditionally compiled applications cannot migrate between heterogeneous ISA CPUs. Pop-

corn Linux [4] and Venkat and Tullsen [36] over come this issue by creating multi ISA compilation frameworks and run-time systems that enable cross-ISA execution migration. Since it is not possible to migrate threads at any arbitrary instruction, these works insert migration points instruction post-states where an application's memory state is semantically consistent across ISAs by modifying the source code at Intermediate Representation (IR) level. The multi-ISA compiler creates program code for each ISA and aligns the virtual addresses so that all program symbols are located at the same addresses. The compiler also generates metadata that describe the layout of the runtime stack, live variables at migration points and the target locations of the live variables (i.e., stack slots or registers), so that the stack can be recreated for the destination ISA at a migration points. The compiler generated object files are then linked with application libraries and a run-time system to generate multi-ISA executable binaries.

As Popcorn Linux is the only infrastructure that currently supports multi-ISA framework and cross-ISA migration between x86 and RISC-V machine [29] our scheduler utilizes this framework to produce the multi-ISA binaries and is built to run on Popcorn Linux.

1.2 Project Contribution

We developed an application job scheduler that decides what jobs must be offloaded, at run-time, from an x86 server to a RISC-V board (that is attached to the server) to alleviate resource congestion on the server. The scheduler, called the Lowest-Slowdown-offloaded-First (or LSF) monitors job performance on the server (using performance counters) and uses the monitored performance to predict job performance on the board. The prediction is then used to guide offloading decisions. We evaluate **LSF**'s efficiency using a set of compute intensive micro-macro benchmarks. The evaluation platform includes a SiFive HiFive RISC-

V board that is connected to an Intel Xeon Silver 4110 server using a low-speed Ethernet switch (1 Gbps). **LSF** can obtain upto 20% of throughput increase and upto 16% energy efficiency. We consider these gains significant as the cost of a SiFive board is 3X less than that of the server.

The project we makes the following contributions:

- The design and implementation of the **LSF** scheduler.
- Evaluation of the **LSF** scheduler which demonstrates a throughput increase of upto 20% and energy efficiency gain of upto 16%.

To the best of our knowledge, no prior work proposes a scheduler to offload jobs to a RISC-V board in an x86/RISC-V heterogeneous-ISA hardware.

1.3 Report Organization

The rest of this report is organized as follows: Chapter 2 discusses the necessary background for completeness. Chapter 3 describes past and related works in the space of scheduling jobs in heterogeneous systems setup. Chapter 4 and Chapter 5 discusses the design and implementation of the **LSF**. Chapter 6 discusses the performance evaluation and energy efficiency of the **LSF**. Finally, in Chapter 7, we conclude this report, discuss **LSF**'s limitations and identify directions for future work.

Chapter 2

Background

This chapter aims to provide the background information that is necessary to understand this report. We first briefly introduce the RISC-V architecture, how it competes in the present processor market, and why it is gaining momentum. As currently, there is no off-the-shelf machine with a heterogeneous multiprocessor, we use a Xeon server and RISC-V system to evaluate our scheduler. These machines are connected via Ethernet to form a heterogeneous ISA system. The Popcorn Linux makes multi-ISA migration of applications between these two machines possible. Hence, we also briefly discuss working of Popcorn Linux [4], and how it handles migrating applications between cross-ISA machines.

2.1 RISC-V

RISC-V is an open-source, royalty-free ISA that is based on well-established RISC principles. It was commenced by the engineers at UC Berkeley [38]. RISC-V is suitable for specific application fields such as storage, computing, and AI applications.

Compared to the other group of processors that are based on the same principle, i.e; ARM, RISC-V has a smaller base instruction set architecture. It is highly customizable as instruction sets can be added as extensions depending on the requirement. One of the main advantages of RISC-V over ARM is the cost of a RISC-V machines is much lower than that of ARM. This is mainly due to the reason that ARM earns their revenue by selling IP to

the processor manufacturers, hence the consumer would end up paying for both the manufacturing of the processor and cost of using the IP. However in the case of RISC-V as it is open source the consumers can use the open source designs or develop new designs according to their needs and they will only be paying for the manufacturing of the chips. As there is a spike in deployment of ARM machines in the datacenters [9] one of the main ways the consumers can reduce the costs of their deployments is by using RISC-V processors.

The present RISC-V offerings cannot match ARM processors in performance. But since RISC-V is a royalty-free and open source architecture, it has been receiving much interest from corporate and academia. RISC-V has made it easier for the academia and the industry to develop their custom chips. The demand for custom chips has also increased. With the increasing use of RISC-V as an accelerator in networking devices [11] and the release of a server RISC-V core [34] it is worth investigating the amount throughput gain and power efficiency that can be achieved by using RISC-V in a heterogeneous ISA system setting.

2.2 Popcorn Linux

To enable migration of an application during runtime between heterogeneous ISA processors, there are two requirements that need to be met. The underlying operating system must provide the ability to migrate threads between heterogeneous ISA processors; it should also have the ability to provide distributed virtual shared memory across heterogeneous ISA processors. We use Popcorn Linux [4] which gives us the ability to migrate threads and applications data between heterogeneous ISA processors.

Popcorn Linux uses a replicated kernel OS design. The kernels on different machines communicate with each other via message passing, hence providing an image that the applications are executing on a single machine. The OS also provides all the necessary support for the

threads to migrate and execute between processors of different ISAs

Popcorn Linux has the facility to migrate pages between the kernels on demand. After a cross-ISA migration, the migrated application does not have any pages mapped into the memory and all memory accesses by the application results in page faults. When the page faults occur the OS page fault handler is notified. The OS then intercepts and observes the faults and migrates the relevant page data between the machines. To enable the ease

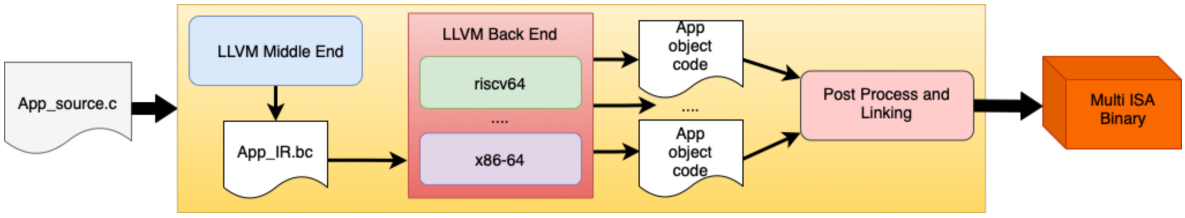

Figure 2.1: Popcorn Compiler Toolchain

of development for such an environment, the Popcorn Compiler toolchain is used. This toolchain is responsible for taking as input unmodified C files and generating multi-ISA binaries that contain the modified data, code sections and the state transformation meta-data that enable migration of applications between cross-ISA machines.These multi-ISA binaries are generated by building upon the modularity of LLVM.

The virtual address space in these binaries is aligned so that the referenced symbols are the same across all the machines. Migration in these multi-ISA binaries can happen only at an equivalence point [37], i.e; the point at which the execution of the application has reached a point where a valid mapping between the ISAs can be established. An LLVM pass inserts migration points at all the equivalence points in the application. The pass tag contains the information about all the live values at that migration point that would need to be recreated if the application were migrated at that migration point. This information is passed to all the binaries of different ISAs.

All the ISAs contain the same information about the live values at the migration points.

The machine code for each of the multi-ISA binaries is different, hence the backend stores this live value information in a stack or a register. Apart from the live values, the compiler also notes the information about the callee saved register and the frame sizes. To correlate all the call site information at run-time, each site is also assigned a unique identifier. In the final step the linker takes in all the object files and produces the multi ISA binaries that have aligned virtual address space and contain the metadata required for transforming the application during migration.

When an application is executed, its threads execute like normal threads until it hits a migration point. At this point they check if the migration for this thread has been requested. This check is done by issuing a syscall to check the per thread flag that is used to signal migration. If a thread sees that migration has been issued to it, it takes a snapshot of its current registers and begins it's stack transformation and register set transformation. The first step is to unwind the stack and check all the activations that are active and load each of these functions metadata. It then traverses the stack moving all the active live values to the destination ISA location. The OS's thread migration service is responsible for migrating the outermost functions' transformed register set to the right destination node. This migration process creates a new thread with all the transformed register set information and returns the thread to the userspace. The application then continues executing normally on the destination node.

This project builds upon the work done by Phillipidis [29] of porting the Popcorn Linux to x86 - RISC-V architecture.

Chapter 3

Related Work

This chapter provides an overview of related works that have been done in the research areas that LSF targets. The research effort in this field is divided into two sections, in Section 3.1 we discuss the works done in the field of Single-ISA Heterogeneous machines and in Section 3.2 we discuss the efforts done to schedule applications in Heterogeneous-ISA systems.

3.1 Scheduling in Single ISA Heterogeneous Systems

The fundamental problem of a heterogeneous system is to understand how best the system resources can be utilized. This problem has been studied extensively.

Koufaty et al. [19] propose the design of a scheduler depending on the slowdown of the applications. It matches applications to the core with appropriate performance and micro-architecture. The slowdown of an application in this scheduler is characterized by the internal and external stalls faced by the threads of the application.

Van Craeynest et al. [35] propose a proportional fair scheduling approach to solve the problem of barrier synchronization. The scheduler balances threads by giving equal time to all threads on a core, or by making sure all the threads face same amount of slowdown. This scheduler does not take into consideration the benefits of core sensitivity of an application.

Petrucci et al. [28] proposes Lucky Scheduler that use lottery based approach to schedule jobs in heterogeneous system. Depending on Million Instructions per Second and LLC misses each application are given tickets. Higher the number of tickets, higher the chance of executing on bigger core.

Jibaja et al. [16] in their scheduler WASH presents a dynamic thread scheduling algorithm for AMP (Asymmetric Multi-core Processors) to improve performance and optimize energy consumption. Their dynamic scheduler is based on Principle Component Analysis(predictive model), which is trained offline using performance metrics to predict runtime slowdowns. At runtime, the VM collects the performance data such as criticality, thread sensitivity, and priority. Thus at every fixed interval, WASH sets thread affinity to a specific core by collecting VM performance data in addition to checking core sensitivity and dynamic parallelism metric.

Another approach that has been studied extensively is to schedule jobs by profiling them for their memory and cache contention. [12][21][5][8].

However, all of these approaches have been developed for single ISA heterogeneous systems that have heterogeneity in terms of their micro architectures, frequency of the processors, and cache sizes [15].

3.2 Scheduling in Multi-ISA Heterogeneous Systems

The field of heterogeneous-ISA has received less attention than that of single-ISA heterogeneous systems. Work has been done in the field of accelerator-based heterogeneous environments [13]. The work done by Beisel et al. [6] extends the Linux Complete Far Scheduler for CPU/GPU. Ardalani et al, work XAPP [1], proposed an ensemble prediction mechanism

that, when given a source code, predicts which core (CPU/GPU) to run on. This scheduler would require a profiling phase to understand which core an application needs to run on. [14] work studies scheduling parallel applications by making decisions at runtime by observing the metadata of the applications.

Barblace et al. [4] proposes two scheduling policies for heterogeneous system, the first policy balances the number of threads on both the machines making up the hetereogenous system, while the second policy runs more number of threads on the x86 machine. This policy doesn't take into consideration the slowdown an app would face when running on the core with lower performance.

Karaoui et al. [17] present a scheduler for heterogeneous-ISA systems, that takes into consideration the slowdown factor and the core count on both the processors. It migrates application that has their slowdown factor less then the ratio of number of cores.

Yihan et al.[27] proposes a scheduler for SIMD region migration that divides the applications into regions depending on their slowdown and assigns them to cores depending on the slowdown faced when executing on that core. Both Karaoui et al. [17] and Pang et al. [27] assume we have the knowledge of the slowdown of an application before their execution, however it is hard to argue such information would be available prior.

The idea of using correlation between performance statistics and the slowdown incurred, to make informed scheduling decisions has been investigated before. Olivier et al. [25], in their paper HEXO, developed a correlation value framework that collects the performance metrics of a job and presents this data to a scheduler when the instance of the job begins to run. However this work targets embedded ARM boards coupled with x86 servers.

As RISC-V boards that have the ability to execute Linux applications are comparatively new, to our knowledge there exists no such scheduler that offloads jobs to a RISC-V board.

In this project we extend the work done in HEXO [25] to a RISC-V - x86 heterogeneous setup.

Chapter 4

Design

The motivation of our work is to design a scheduler that offloads applications at run-time from an x86 server to a RISC-V board to alleviate resource congestion on the server. In this chapter, we present the design and implementation of **LSF** scheduler that solves this problem by monitoring application performance on the server.

In Section 4.1 we profile our workload for slowdown and energy efficiency by running the considered micro-macro benchmarks on the SiFive RISC-V board and comparing the results with Intel Xeon E5-2637 x86 server. In Section 4.2 we present the design of **LSF** scheduler.

4.1 Profiling Phase

This section describes the experiments we conducted to understand the slowdown and energy efficiency characteristics of the micro-macro benchmarks we considered. This information can help us in taking a more informed decision towards the design of the scheduler.

LSF scheduler targets compute intensive HPC workloads. Hence to measure the performance and power characteristics we selected benchmarks from NAS Parallel Benchmark suite [3] as they are representative of long running compute intensive applications. We ran a single instance of serially compiled Class A workload of this benchmark on each of the machines at a time and compared their execution times and energy consumption. We also use a set

of custom memory intensive micro-benchmarks to measure the execution times and energy consumption of the system. We discuss the workings of this micro-benchmark in Section 6.1.

The slowdown incurred when running the benchmarks on the SiFive RISC-V board to Intel Xeon server is shown in the Figure 4.1. The x-axis shows the benchmarks being run and they-axis shows their corresponding slowdown. The slowdown is normalized to the execution time of the applications on the Xeon server, that is 1 on the y-axis represents the performance of the Xeon server.

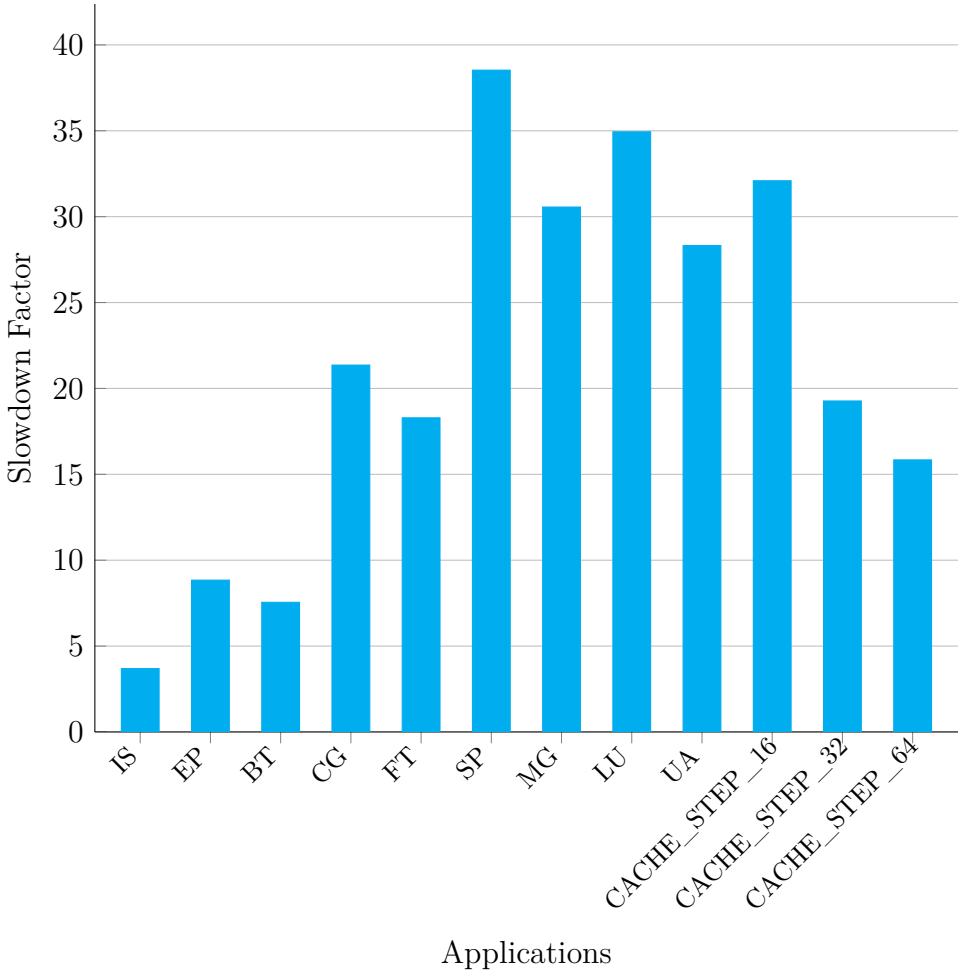

Figure 4.1: Application Slowdown on SiFive RISCV Board vs 4-Core Inetl Xeon E5-2637Serve

From the Figure 4.1, we can observe that the slowdown is relatively small for some of the

benchmarks, while few applications see very high slowdowns. Overall we see slowdown in the range of 3.6X - 38X when running an application on the RISC-V board. It is important to put into perspective that this slowdown is being faced by a RISC-V machine which is three times lower in cost than that of the Xeon server. The result is also not surprising, considering the SiFive RISC-V is clocked lower than the Intel Xeon server and compared to the SiFive RISC-V the Intel x86 processors have been in market for decades and have been highly optimized. The next generations of the SiFive RISC-V boards are likely to yield more slower down than the present ones.

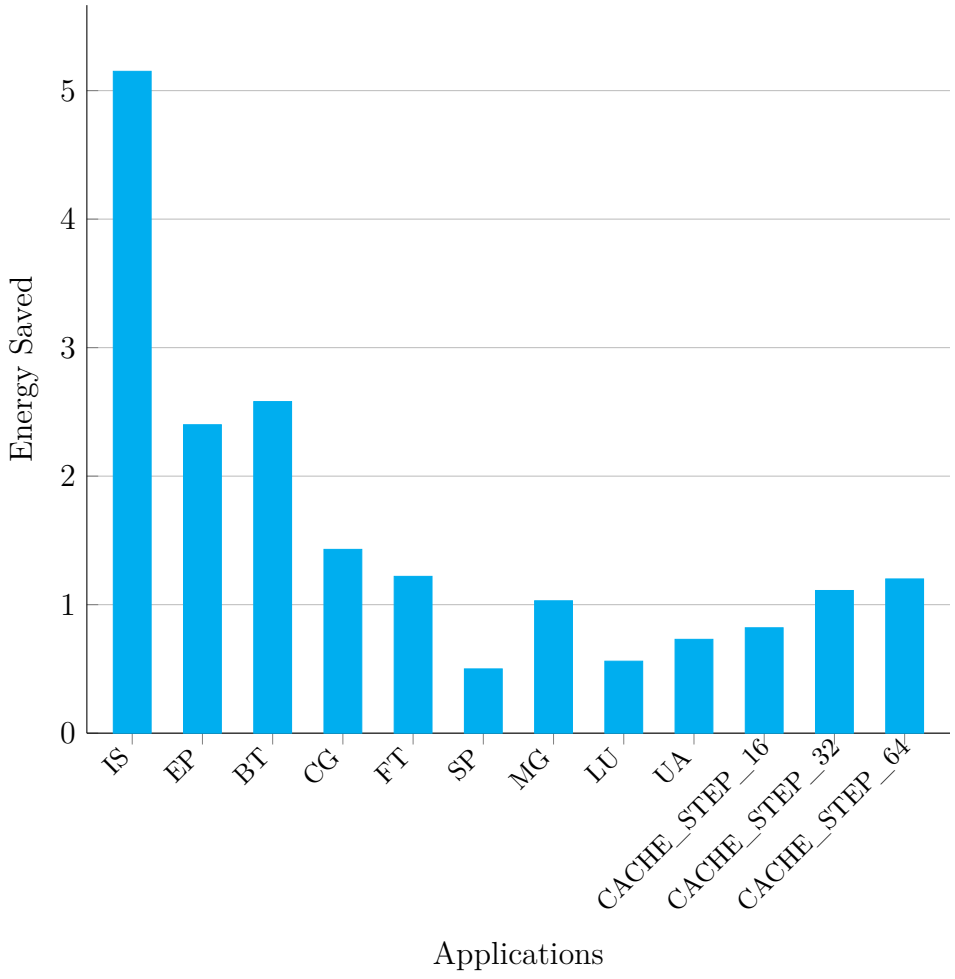

Figure 4.2: Power Saved when running on RISCV vs 4-core Xeon

We also profile these set of applications for their energy consumption on the 4-core Xeon server. Figure 4.2 shows the energy saved when an application runs on the SiFive RISC-V board. The x-axis shows the micro-macro benchmarks y-axis shows the energy saved. The factor energy saved in normalized to that of Xeon's energy consumption, that is 1 on the y-axis represents energy consumption of Xeon.

From the figure 4.2 we can observe for applications that face higher slowdown also consume more energy. This is because the execution time of these applications is so high that it offsets the low energy consumption of the RISC-V board. From these experiments it can be concluded to get higher throughput and low energy consumption applications with lower slowdowns should be scheduled to execute on the SiFive RISC-V board.

4.2 Design

In this chapter we take into consideration our learnings from the profiling experiments and present the design of **LSF** scheduler migrate jobs between an Intel Xeon server and a SiFive RISC-V board to alleviate resource congestion on the server.

LSF's objective is to maximize the throughput and power efficiency of the system. This is done by offloading jobs to the RISC-V board such that at any point, neither of the machines are overloaded.

The criteria by which the scheduler decides what jobs are to be migrated onto the RISC-V machine is by checking the applications' slowdown. In a data center scenario, we do not always have prior information about a job's slowdown when they arrive. Scheduling decisions in such systems can be made by observing and comparing the application's behavior before migrating and after migrating. We can assess the behavior of the applications by monitoring

performance metrics like last-level cache metrics, cache-loads, cache-stores, and the number of instructions. These metrics vary depending on the machine specifications and micro-architecture.

We profiled the selected applications for last-level cache misses per second number of instructions per second, cache-loads per second and number of branches per second. To get these metrics, we ran a single instance of the application on the server. We found a strong correlation between number of instructions per second and the slowdown observed. The correlation r is 0.80, and R^2 is 0.64. This correlation factor indicates that there is a direct relationship between these metrics and the amount of slowdown. Hence by monitoring these metrics, the scheduler can dynamically learn the slowdown information associated with each job. As the objective of the scheduler is to gain higher throughput, all the jobs start their execution on the server, hence we don't need to measure these metrics for the applications when running on the RISC-V machine. In our design we assume that the scheduler is trying to schedule an infinite queue of jobs, and hence it server or the RISC-V board will never be idle. In the case of a finite queue we would also have to measure the performance metrics on the RISC-V board so that applications can be migrated back to the server, if the server becomes idle. Another factor that affects the performance of the applications is resource availability. As we consider compute-intensive jobs, we do not run more than one job on a single core, both on the server and the RISC-V board. For this project's scope, we assume that there exists enough memory for the applications to run on either of the machines. Hence, the scheduler doesn't consider resource availability while scheduling and migrating applications.

Figure 5.1 illustrates the workflow of **LSF**. By default the scheduler is idle and is waiting for four different events -

1. Application begins it's execution

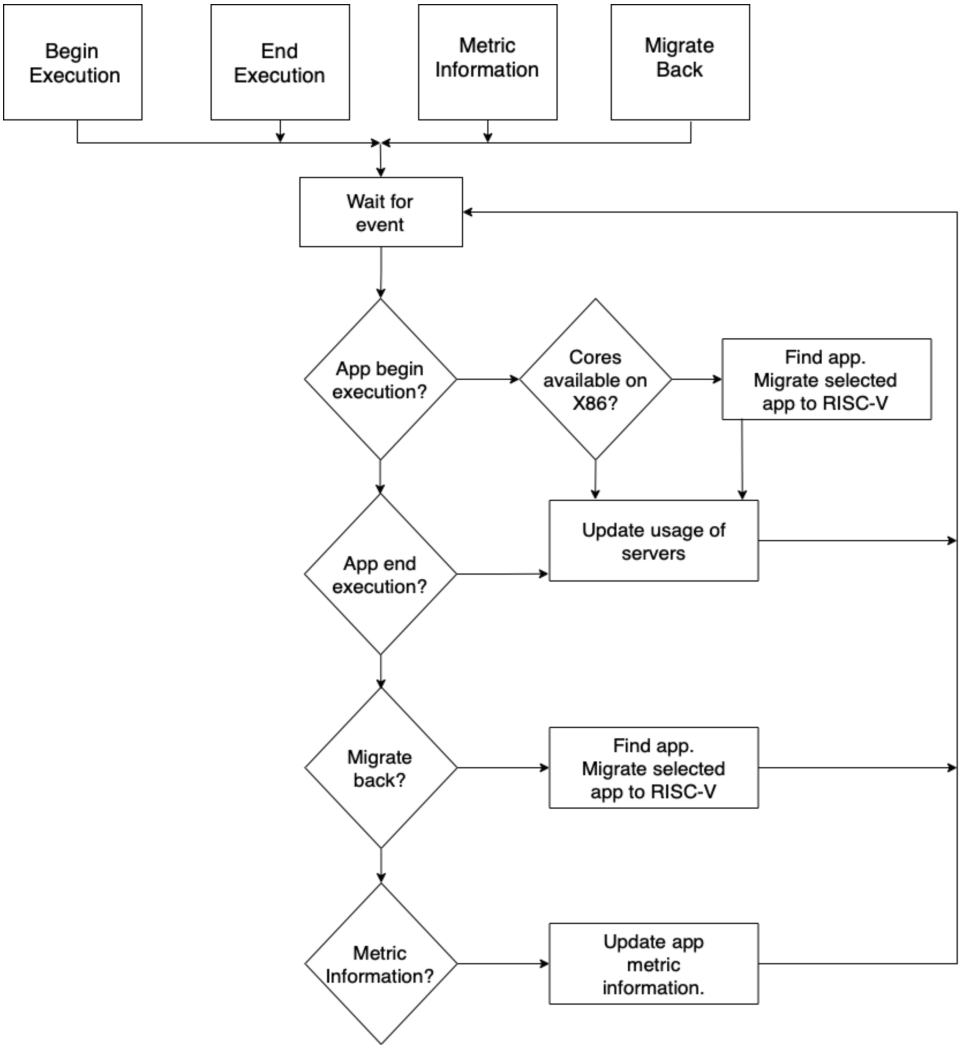

Figure 4.3: Scheduler Workflow

2. Application migrates back from the RISC-V board to the server.

3. Application ends.

4. An event notifying the scheduler of the performance metrics of the applications.

Algorithm 1 LSF

```
1: x86_size: Number of available server cores
2: risc − v_size: Number of available RISC-V cores
3: x86_list: List of (apps,pid) running on server
4: risc − v_list: List of (apps,pid) running on risc-v board
5: metric[apps]: Array of values containing performance metric information corresponding
   to application
6: procedure LSF(command, value)
7:     command: app_name/ end_signal/ migrate_back/ perf_metric_singal
8:     value: PID/ perf_value
9:     if command = end_signal then
10:        // App completed execution
11:        // Here the value = PID
12:        if (value) in x86_list then
13:            Remove corresponding value = pid from x86_list
14:        else
15:            Remove corresponding value = pid from risc − v_list
16:    else if command = migrate_back then
17:        // App migrates back to x86 server.
18:        // Here the value = PID
19:        Remove corresponding value = pid from risc − v_list
20:        Insert (apps, pid) in x86_list
21:        victim_app_pid = get_victim_x86()
22:        Migrate victim_app_pid to RISC-V
23:        Remove corresponding victim_app_pid from x86_list
24:        Insert victim_app_pid to risc-v_list
25:    else if command = perf_metric_signal then
26:        // Receives Instr per second information
27:        // of an app.
28:        // Here the value = perf_value;
29:        // command = name_of_app + perf_metric_signal
30:        Update metric[app] value
```

```
    else                                   ▷ Here value = PID; command = Name of app
        // App begun it's execution
        // Here the value = PID; command = Name of App
        if len(x86_list) < x86_size then
            Insert (command = app,value = pid) in x86_list
            Continue executing on the server
        else
            victim_app_pid = get_victim_x86()
            Migrate victim_app_pid to RISC-V
            Remove corresponding victim_app_pid from x86_list
            Move victim_app_pid to risc − v_list
            Insert (command = app,value = pid) in x86_list
procedure GET_VICTIM_X86
    if len(metric) = 0 then
        return Head of x86_list
    else
        Find (apps, pid) in x86_list where metric[app] is least in x86_list
        return pid
```

In the case of an event where an application has just begun its execution, the **LSF** checks the load of the server, and if there are cores available on the server, the application continues executing on the server. If there are no cores available on the server, the **LSF** finds a "victim" to migrate and free up cores on the server. A victim application is one that has the least instructions per second value from the currently running set and has been executing for the most amount of time on the server. We select an application which has been already executing on the server to migrate because the slowdown brought by this application would be lower than migrating an application that has just begun it's execution. When no information about the profiled metrics is available, the scheduler migrates the job that has been executing on the server for the most amount of time.

Currently, the RISC-V port of Popcorn Linux cannot support applications to end their execution on the remote node. Hence, we migrate the applications back to the RISC-V machine just before they finish. The **LSF** is notified about this back migration by the

application. The LSF then finds a victim job from the currently running set of jobs on the server and migrates it, preventing the server from overloading.

In the case when an application has completed it's execution the LSF is notified of this event. The LSF then updates the usage of the servers accordingly and waits for the next event.

When an application completes it's execution the workload script notifies the LSF of the instructions per second performance metric of that application. This is done for all the applications. Hence LSF always has the most up to date information about the instructions per second of an application and can take an informed decision while offloading applications. The scheduling policy's pseudo code after it receives an event is shown in Algorithm 1.

Chapter 5

Implementation

In this chapter, we discuss how **LSF** was implemented.

We implement **LSF** as an event-based client-server model, using socket communication in C as shown in Figure 5.1. This implementation is done by modifying the Popcorn Linux Compiler to include the client code in all the applications. The scheduler code contains the server code that is listening for connections and is waiting to schedule or migrate jobs depending on the usage of the server.

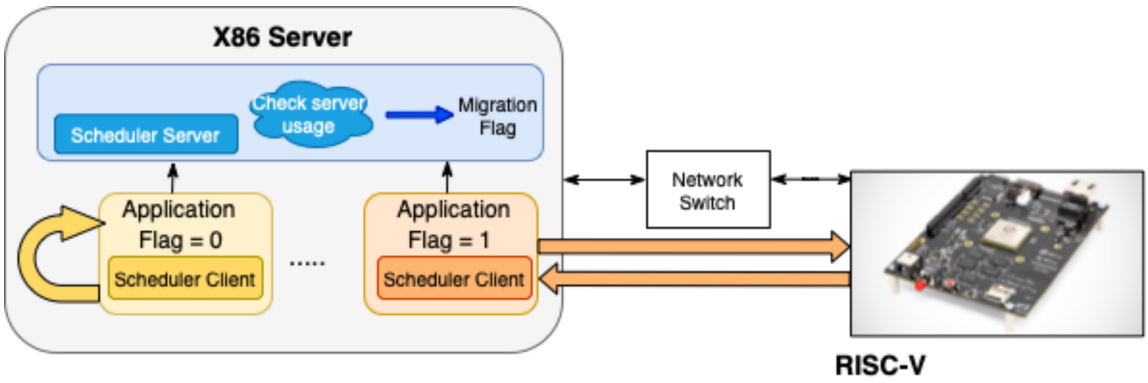

Figure 5.1: Least Slowdown First Scheduler's Client Server model to migrate jobs in a heterogeneous ISA system with x86 and RISC-V. Flag represents the target machine where the application is scheduled to execute.

The scheduler is a single-threaded user-space application written in C. It runs on one of the cores of the server and aims at scheduling jobs so that neither of the machines is overloaded at any point in time. All the jobs are continuously profiled for the number of instructions per second. This live profiling is done using Linux's *perf-stat* tool.

We use Popcorn Linux's signal-based migration API to migrate applications from the server to the RISC-V machine.

Intel's RAPL [10] provides a set of counters that can be used to read energy and power consumption information of the processor. As no such libraries exist for RISC-V to read the energy consumption of the processor, we need to measure the entire power of the system. To this cause we use a pair of HOBO Plug Load Data Logger [26] to measure the power consumption of the server and RISC-V board.

We also measured the overhead of **LSF**. **LSF** brings an overhead of 0.8 seconds - 1 second. This includes the overhead of using the *perf-stat* tool. Since our scheduler is aimed at long-running compute-intensive workloads, this overhead is negligible.

Chapter 6

Evaluation

In this chapter, we describe the evaluation of **LSF**. We aim to answer the following two questions

- What is the **LSF**'s throughput gain?
- What is the **LSF**'s energy efficiency?

This chapter is organized as follows: Section 6.1 describes the experimental setup we used to evaluate **LSF**. Section 6.2 provides a summary of our evaluations.

6.1 Experimental Setup

6.1.1 Hardware Setup and Baseline

Several vendors offer RISC-V embedded boards with varying price points depending on their frequency, memory, cache size, and extensions. For this project's scope, any RISC-V board that was to be considered needed to satisfy two requirements - 1. the price of the board needed to be a fraction of the x86 server, and 2. the RISC-V board needs to have support to run Linux applications. Although the first requirement was satisfied by most of the vendors, SiFive's HiFive Unleashed [32] and, HiFive Unmatched [33] are the only two boards that

Machine	Xeon	HiFive Unleashed
CPU model	Xeon E5-2637	U54 RISC-V
ISA	x86-64	RV64IMAFDC
CPU Frequency	3 GHz	1.5 GHz
Cores	4 (8 HT)	4
RAM	64 GB	8 GB
Power (idle)	60 W	4.15 W
Price	$3049	$999

Table 6.1: Considered servers & board characteristics.

support running Linux applications. We use the HiFive Unleashed RISC-V board to evaluate our scheduler.

Table 6.1 describes the key specifications of the sever and the RISC-V machine. Our heterogeneous setup consists of an Intel Xeon E5 and a SiFive HiFive Unleashed. These are linked via Ethernet that is capped at 1 Gbps. The OS being used on Xeon E5 is Ubuntu 16.04, while the RISC-V is running a custom Linux distribution built using the Yocto Project. Both the machines are running Popcorn Kernel 5.2.21. To calculate the idle power consumption of both the systems, we measure the power consumed by the server and the RISC-V board when idle for a period of 10 minutes.

Hyper-threading is disabled on the x86 server and, we use 3/4 of the available cores on both machines. This is done as it is common practice in data centers not to overload any of the servers. Scheduler is launched on one of the cores of the server. In the evaluation experiments, we demonstrate the benefits of **LSF** in consolidated scenarios. We show that connecting a low cost, low power board to the server can yield throughput than a single server while being energy efficient in the case of applications that face lower slowdown on RISC-V board.

The evaluation workload script is a multi-threaded Python script running on another machine on the same network as our servers and the RISC-V machine. We do this to closely mimic the scenarios in data centers where the jobs are assigned to the worker servers by another

machine. The server is only responsible for executing the tasks. The scheduler is usually coupled with the server assigning jobs in a cloud computing scenario and acts as a load balancer. We implement our scheduler on the server to reduce the latency between the decision making process and the time to issue the migration command to the application. The workload scripts take in as input a batch of applications that needs to be scheduled. This batch of jobs is shuffled and added to a work queue. This is repeated multiple times to create an infinite queue of jobs. The rationale behind this is to randomly distribute jobs while ensuring all the jobs are uniformly distributed.

The evaluation script always assigns jobs just to the server. It does this by launching a thread that establishes an SSH session with the server and launches the job. This evaluation script also contains a piece of client code to communicate with the scheduler server the number of instructions per second performance metric of the applications. Each thread communicates the metric of their respective application to the **LSF** and then selects another job from the work queue to launch on the server. We repeat this process until the end of the evaluation period. We ran each of the experiments for 75 minutes because the most time taken by an application to execute on RISC-V ranges between 47-50 minutes and, we wanted to make sure multiple instances of all the applications are scheduled on the RISC-V machine.

6.1.2 Benchmarks

To evaluate the **LSF** we use a set of macro and micro benchmarks. For the macro benchmarks, we use the NASA Parallel Benchmark suite[3]. This suite contains benchmarks which are representative of HPC compute intensive workloads. For our evaluation we use the serially compiled class A workloads of NPB suite. The class value is representative of the workload of the benchmark.

Another set of applications deployed in data centers are memory-intensive applications; hence we developed a micro-benchmark to evaluate our **LSF**. These micro-benchmarks allocate a fixed amount of memory that is higher than the cache size. It then accesses this allocated memory in a randomized pattern such that each operation results in a memory fetch. We access the allocated memory with step sizes multiples of the cache line size to create different execution loads. As the step size increases, the number of memory fetches and cache-loads decreases, decreasing 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 loads. Even though the cache size of the server and the RISC-V machine are different, this micro-benchmark is suitable for our system as the cache line size remains constant. Due to similar memory access patterns between the server and the RISC-V machine, we find a direct correlation between the number of LLC-cache misses per second and slow down. Thus, the slowdown decreases as the cache fetches decrease for increasing step size. The memory allocation is done using integer arrays, hence we consider step sizes of 16, 32 and, 64. Varying degrees of slowdown in this micro-benchmark is achieved due to the size of the cache and the amount spent in fetching data from the memory. Depending on their step sizes, we name our three micro-benchmarks as *cache_step_16, cache_step_32 and, cache_step64.*

6.2 Throughput Experiments

We run two sets of experiments to measure the throughput of **LSF** scheduler. In the first experiment, we launch 8 sets of jobs. Each of these sets contains an infinite queue of a single instance of the micro/macro benchmarks we considered. We ran each of these sets for 75 minutes. These sets represents an ideal case when only of single jobs being launched

on the server that experience same amount of slowdown. We consider our baseline to be a homogeneous setup consisting of an x86 setup.

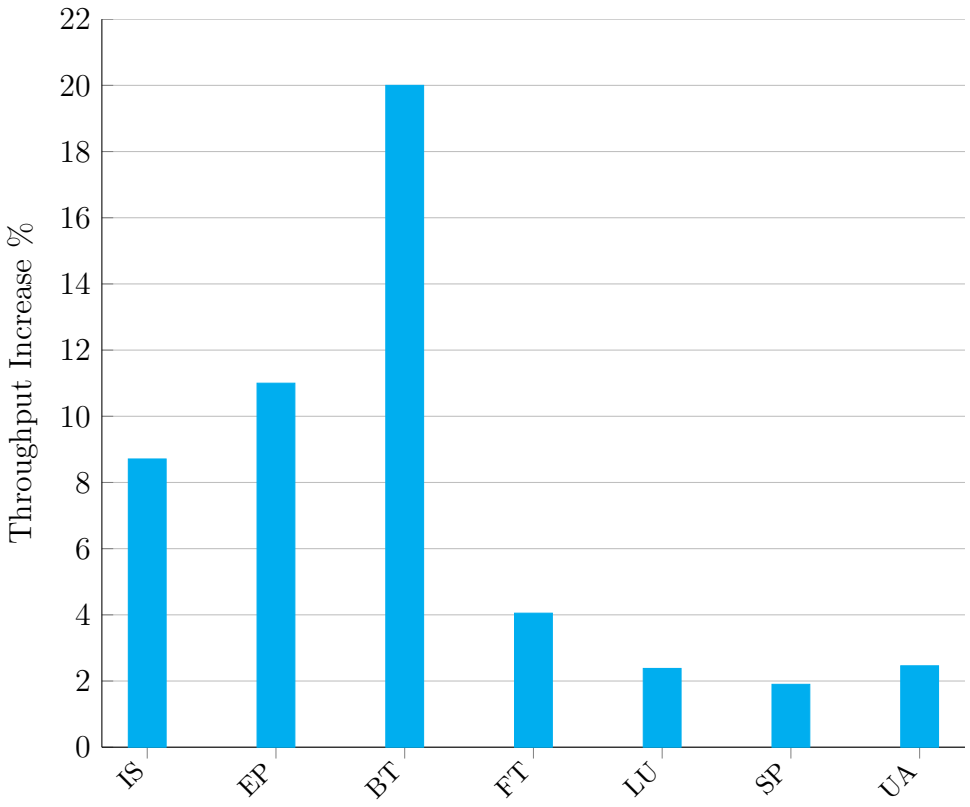

Figure 6.1: **LSF**'s throughput gain over a single Xeon server

Figure 6.1 shows the throughput increase achieved by the set of applications considered. Here the x-axis shows the applications in the set and the y-axis represents the throughput increase. Unlike our assumption that the application with the least slowdown would yield higher throughput, application BT is the best case with a throughput yield of 20%. Although IS gets an increase in throughput it is lesser than that of BT and EP that have higher slowdowns when running on RISC-V board. This because IS has a very low execution time

and the overhead of migrating this application to the RISC-V board outweighs the benefits gained by migrating it when compared to BT and IS. Currently **LSF** scheduler does not take into consideration the execution time of the application and migrates applications only depending on their slowdown information. SP represents the worst case for our scheduler with a throughput yield of 1.9%. The remaining benchmarks follow the trend where the throughput of the applications are proportional to their slowdown.

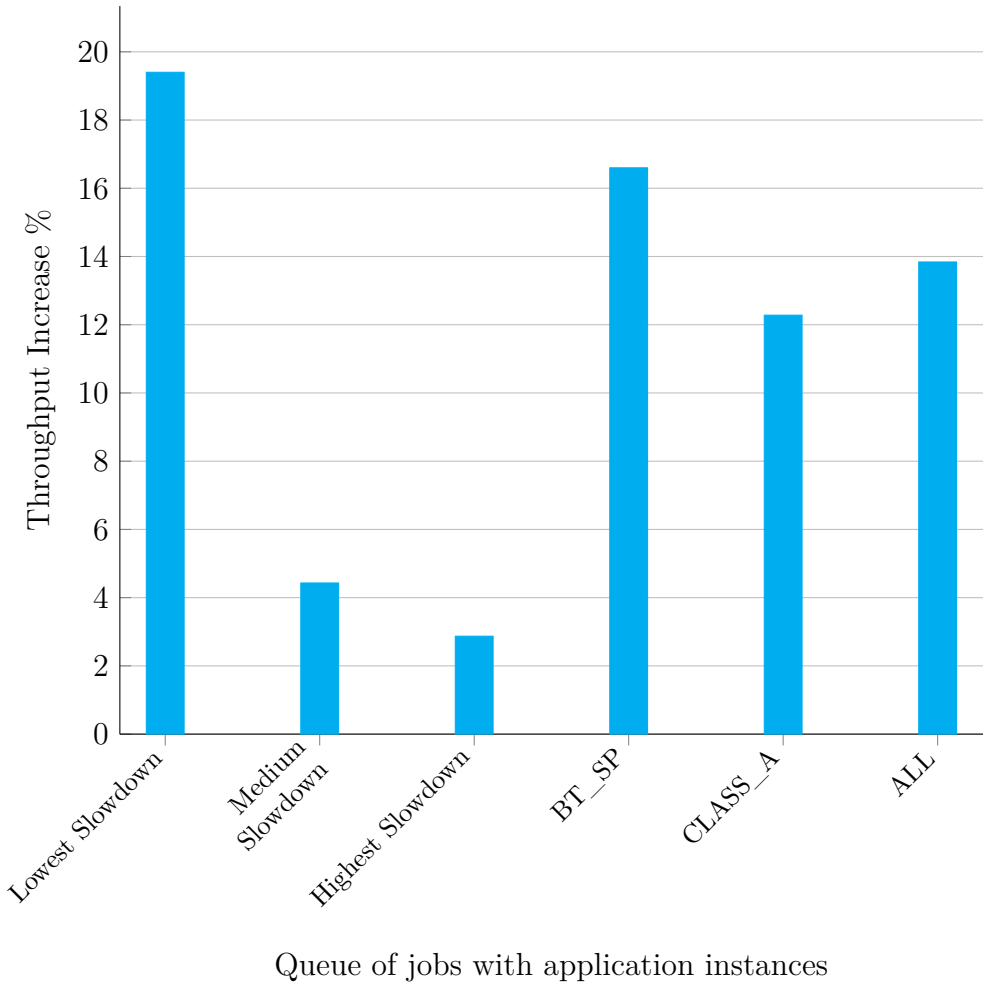

Figure 6.2: **LSF**'s throughput gain over a single Xeon server (consolidated case)

We also test **LSF** for consolidated case. We run six sets of consolidated applications A=BT, EP and IS, B=FT, cache_step_32, cache_step_64, C=CG, LU, MG, SP, UA, cache_step_16,

D=BT_SP, E=BT,CG,EP,FT,IS,LU,MG,SP and UA, F=NPB Apps + memory intensive micro-benchmark. Set A contains the appliations that experience slowdown less than 10% hence represents best case for **LSF**. Set B contains the applications that experience slowdowns in the range of 11%-20%, hence it represents a middle case for **LSF**. Set D contains applications that face slowdown greater than 21% hence represents the worst case for **LSF**. Set C contains applications with the best and worst slowdown. Set E contains NPB applications with a variety of workloads and Set F consists of all the micro and macro benchmarks we considered for evaluation. Set E and F represent the real world scenarios where a combination of jobs are launched on a server. These experiments were run for 75 minutes.

Figure 6.2 shows the throughput gain for the consolidated sets. Set A we see a throughput yield of 19% which is similar to the throughput gain in the case of an infinite queue of single instance of BT. When combining the best and worst case (BT and SP) we get a throughput increase of 16.6% this is less than the throughput increase brought by BT because when the jobs are first launched and no profiling information is available, some of the SP applications are migrated to RISC-V board. Set B and C with the middle and the worst case gain a throughput of 4.4% and 2.8% respectively. The set with highest slowdown gains higher throughput than SP as it also contains applications with slowdowns less than that of SP that are more likely to be offloaded on the RISC-V board. Set E and F experience throughput gains of 12.28% and 13.84%. The overall throughput is lesser than the combination of best and worst case. This is because BT takes the most amount of time for execution, during which no information about it's instructions per second metric would be available, hence the scheduler migrates applications with the lowest instructions per second metric available at that instance. In the case of Class A applications with higher slowdown dominate the set and hence would be migrated more often. While in the case of Set F, the set also contains applications with medium slowdowns, hence the probability of migrating applications with

medium slowdown is higher and therefore we see higher throughput in the case of set when all applications are considered.

6.3 Energy Efficiency Experiments

We also collect the energy consumed by the SiFive RISC-V board and the Intel Xeon x86 server when running each of the queues. We calculated the energy efficiency for each sets by dividing the total applications that completed their execution during the evaluation period by the total energy consumed by both the machines during this period.

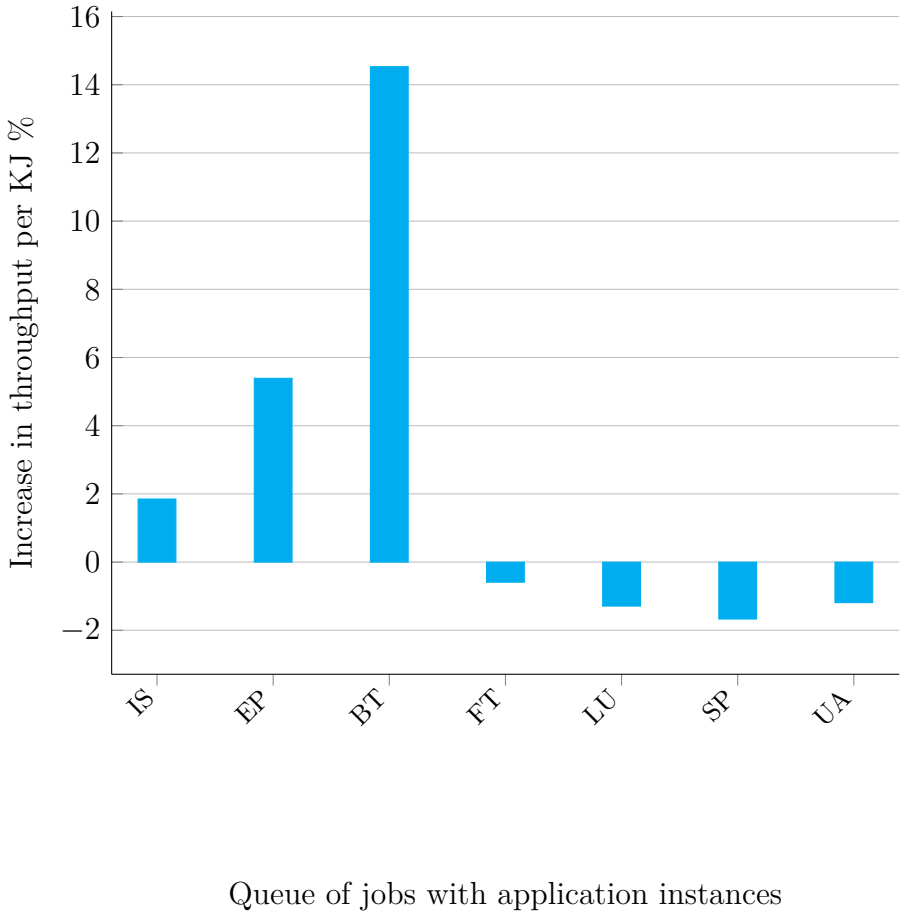

Figure 6.3: LSF's energy efficiency gain over a single Xeon server

Figure 6.3 shows the throughput increase in terms of jobs completed per kilo-joule. The x-axis shows the applications in the set and the y-axis represents the throughput in terms of applications executed per kilo joule. Here 1 on the y-axis represents the energy efficiency of the x86 server.

We can observe that the increase in energy efficiency brought by **LSF** scheduler is less than the increase in throughput. The applications that face slowdown higher than 10X when executing on RISC-V board do not see a higher energy efficiency instead they end up consuming more energy than when running on a x86 server. Hence for applications with higher slowdown FT, LU, SP, UA and cache_step_16 we see that the power efficiency is lower than the baseline.

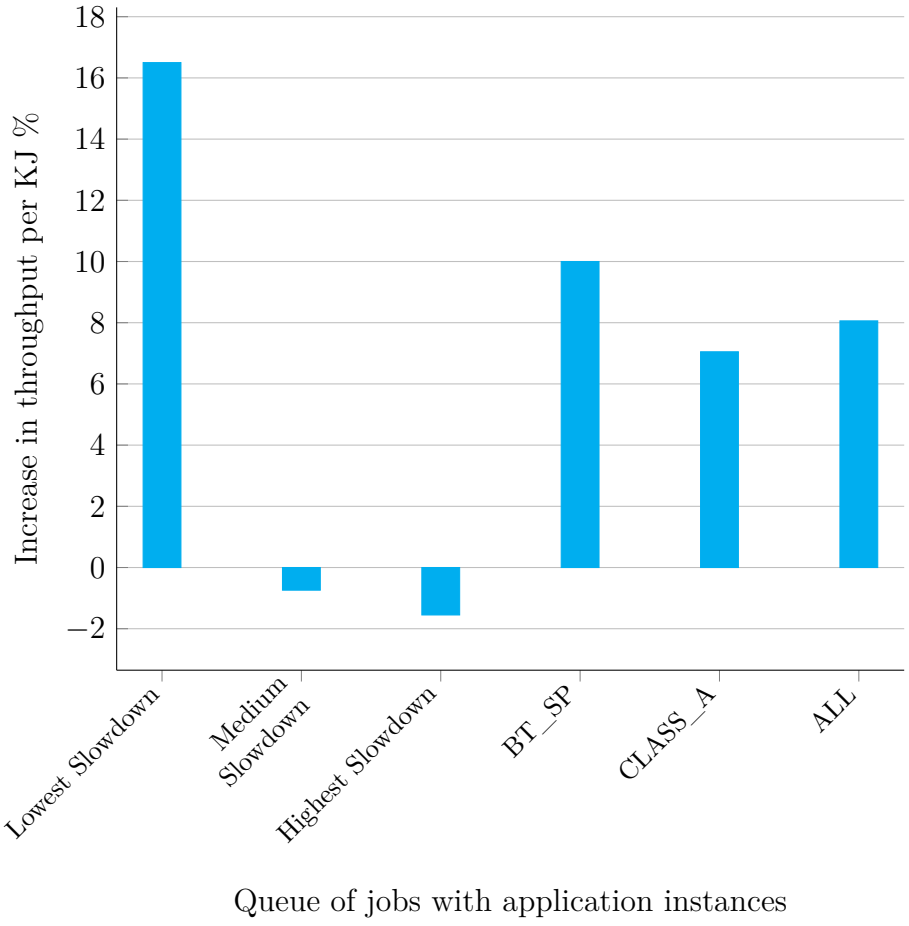

Figure 6.4: **LSF**'s energy efficiency gain over a single Xeon server (consolidated case)

Figure 6.4 shows the energy efficiency in case of consolidated sets. We can observe that the set containing applications with the least amount of slowdown has an energy efficiency of 16.5%. This is higher than that of infinite queue of BT because this set also consists of other applications. These applications consume comparatively less energy on the x86 server than BT. In the case of set with highest slowdown, the energy efficiency is better than SP as the energy efficiency is offset by other applications in the set. The set containing BT_SP sees higher energy efficiency than other sets, this is because this set higher executes higher ratio of BT that results in higher energy efficiency. Set containing class A of NPB Benchmarks and set containing all the considered micro-macro benchmarks follow a similar trend where set containing all micro-macro benchmarks is more energy efficient than class A as it executes higher ratio of applications with better energy efficiency.

6.4 Summary

Running workloads with different sets of applications, we see a throughput increase ranging from 1.9% to 20% depending on the applications in the workload set. The evaluation results prove that **LSF** can identify the right tasks that are to be migrated to the RISC-V board. Energy efficiency evaluation shows that while **LSF** scheduler does provide us energy efficiency in most of the cases with the best being 16.5%, the **LSF** scheduler doesn't help to cause of getting better energy efficiency while also getting better throughput in the case of sets containing applications with higher slowdowns.

Chapter 7

Conclusions & Future Work

7.1 Conclusion

In this project we develop Lowest-Slowdown-offloaded-First Heterogeneous-ISA scheduler (**LSF**), that can migrate applications during run time between RISC-V and x86 server. We were able to demonstrate that in the case when **LSF** receives a combination of workloads, it was able to rightly predict the job performance on the board and migrate the applications which face least amount of slowdown when executing on the RISC-V board. We use a SiFive RISC-V board and an Intel Xeon E5-2367 x86 server to evaluate **LSF** scheduler. Our evaluations we show that **LSF** out performed a single x86 server setup. We were able to achieve throughput gains of up to 20% while getting energy efficiency of 16%. We consider these gains significant as the cost of a SiFive board is 3X less than that of the server. While **LSF** was able to get throughput gains in all the cases, it is important to note that **LSF** scheduler is better suited for long running compute-intensive HPC workloads.

In the end, we strongly support the use of RISC-V boards as a way of offloading jobs during run time in data centers. RISC-V boards can be considered as an alternative to ARM embedded boards that have seen a constant increase in prices. Our results show that dynamically migrating applications in a heterogeneous system can bring significant throughput gains while consuming lesser energy and also being highly cost efficient.

7.2 Future Work

The Future work that can be carried out to improve the current capabilities of **LSF** is highlighted in this section.

The RISC-V architecture has recently gained significant commercial interest, and OS-capable embedded boards with RISC-V cores are increasingly available at the commodity-scale. **LSF** scheduler uses one of the first generation RISC-V boards to offload jobs. With SiFive and other vendors releasing new RISC-V boards and servers, it would be worth investigating the throughput gain and energy efficiency that can be achieved by using the newer and improved generations of these OS-capable embedded boards.

Currently **LSF** does not take into consideration specific set of codes that are more favourable to execute on the RISCV. **LSF** scheduler can be modified to learn and offload code that is more favourable to execute on the RISC-V board, similar to the work done by Pang et al. [27].

Bibliography

[1] Newsha Ardalani, Clint Lestourgeon, Karthikeyan Sankaralingam, and Xiaojin Zhu. Cross-Architecture Performance Prediction (XAPP) Using CPU Code to Predict GPU Performance. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, page 725–737, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450340342. doi: 10.1145/2830772.2830780. URL https://doi.org/10.1145/2830772.2830780.

[2] ARM. ARM Big Little. URL https://www.arm.com/technologies/big-little.

[3] 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. The Nas Parallel Benchmarks. *International Journal of High Performance Computing Applications*, 5:63–73, 09 1991. doi: 10.1177/109434209100500306.

[4] Antonio Barbalace, Robert Lyerly, Christopher Jelesnianski, Anthony Carno, Ho-Ren Chuang, Vincent Legout, and Binoy Ravindran. Breaking the Boundaries in Heterogeneous-ISA Datacenters. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, page 645–659, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450344654. doi: 10.1145/3037697.3037738. URL https://doi.org/10.1145/3037697.3037738.

[5] Michela Becchi and Patrick Crowley. Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures. In *Proceedings of the 3rd Conference on Com-*

puting Frontiers, CF '06, page 29–40, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933026. doi: 10.1145/1128022.1128029. URL https://doi.org/10.1145/1128022.1128029.

[6] Tobias Beisel, Tobias Wiersema, Christian Plessl, and André Brinkmann. Cooperative multitasking for heterogeneous accelerators in the Linux Completely Fair Scheduler. In *ASAP 2011 - 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 223–226, 2011. doi: 10.1109/ASAP.2011.6043273.

[7] Ricardo Bianchini. Improving Datacenter Efficiency. *SIGARCH Comput. Archit. News*, 45(1):327, apr 2017. ISSN 0163-5964. doi: 10.1145/3093337.3046426. URL https://doi.org/10.1145/3093337.3046426.

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

[9] Donald J. Daly and Donald J. Daly. AWS Graviton, 2022. URL https://aws.amazon.com/ec2/graviton/?utm_source=thenewstack&utm_medium=website&utm_campaign=platform.

[10] Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. RAPL: Memory Power Estimation and Capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '10, page 189–194, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450301466. doi: 10.1145/1840845.1840883. URL https://doi.org/10.1145/1840845.1840883.

[11] Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beránek, Luca Benini, and Torsten Hoefler. A RISC-V in-network ac-

celerator for flexible high-performance low-power packet processing. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 958–971, 2021. doi: 10.1109/ISCA52012.2021.00079.

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

[13] Juan Fang, Jiaxing Zhang, Shuaibing Lu, and Hui Zhao. Exploration on Task Scheduling Strategy for CPU-GPU Heterogeneous Computing System. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 306–311, 2020. doi: 10.1109/ISVLSI49217.2020.00063.

[14] Chris Gregg, Jeff S. Brantley, and Kim Hazelwood. Contention-Aware Scheduling of Parallel Code for Heterogeneous Systems. Berkeley, CA, June 2010. USENIX Association.

[15] Keith R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J. Wasserman, and Nicholas J. Wright. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 159–168, 2010. doi: 10.1109/CloudCom.2010.69.

[16] Ivan Jibaja, Ting Cao, Stephen M. Blackburn, and Kathryn S. McKinley. Portable Performance on Asymmetric Multicore Processors. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, CGO '16, page 24–35, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450337786. doi: 10.1145/2854038.2854047. URL https://doi.org/10.1145/2854038.2854047.

[17] Mohamed L. Karaoui, Anthony Carno, Rob Lyerly, Sang-Hoon Kim, Pierre Olivier, Changwoo Min, and Binoy Ravindran. Scheduling HPC Workloads on Heterogeneous-ISA Architectures: Poster. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, PPoPP '19, page 409–410, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362252. doi: 10.1145/3293883.3295717. URL https://doi.org/10.1145/3293883.3295717.

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

[19] David Koufaty, Dheeraj Reddy, and Scott Hahn. Bias Scheduling in Heterogeneous Multi-Core Architectures. EuroSys '10, page 125–138, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605585772. doi: 10.1145/1755913.1755928. URL https://doi.org/10.1145/1755913.1755928.

[20] Rakesh Kumar, Dean M. Tullsen, Parthasarathy Ranganathan, Norman P. Jouppi, and Keith I. Farkas. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, ISCA '04, page 64, USA, 2004. IEEE Computer Society. ISBN 0769521436.

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

[22] Robert Lyerly, Changwoo Min, Christopher J. Rossbach, and Binoy Ravindran. An OpenMP Runtime for Transparent Work Sharing Across Cache-Incoherent Hetero-

geneous Nodes. In *Proceedings of the 21st International Middleware Conference*, Middleware '20, page 415–429, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381536. doi: 10.1145/3423211.3425679. URL https://doi.org/10.1145/3423211.3425679.

[23] Sparsh Mittal and Jeffrey S. Vetter. A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Comput. Surv.*, 47(4), jul 2015. ISSN 0360-0300. doi: 10.1145/2788396. URL https://doi.org/10.1145/2788396.

[24] Nvidia. Variable SMP a multi-core CPU. URL https://www.nvidia.com/content/PDF/tegra_white_papers/tegra-whitepaper-0911b.pdf.

[25] Pierre Olivier, A. K. M. Fazla Mehrab, Stefan Lankes, Mohamed Lamine Karaoui, Robert Lyerly, and Binoy Ravindran. 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*, HPDC '19, page 85–96, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366700. doi: 10.1145/3307681.3325408. URL https://doi.org/10.1145/3307681.3325408.

[26] Onset. Hobo Plug Load Data logger. URL https://www.onsetcomp.com/products/data-loggers/ux120-018/.

[27] Yihan Pang, Robert Lyerly, and Binoy Ravindran. Cross-ISA Execution of SIMD Regions for Improved Performance. In *Proceedings of the 12th ACM International Conference on Systems and Storage*, SYSTOR '19, page 55–67, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367493. doi: 10.1145/3319647.3325832. URL https://doi.org/10.1145/3319647.3325832.

[28] Vinivius Petrucci, Orlando Loques, and Daniel Mosse. Lucky Scheduling for Energy-Efficient Heterogeneous Multi-Core Systems. In *2012 Workshop on Power-Aware Computing and Systems (HotPower 12)*, Hollywood, CA, October 2012. USENIX Association. URL https://www.usenix.org/conference/hotpower12/workshop-program/presentation/Petrucci.

[29] Cesar J Phillipidis. Popcorn Linux Port for X86 - RISCV Architecture, Jun 2020. URL http://popcornlinux.org.

[30] Greg Sadowski. Design Challenges Facing CPU-GPU-Accelerator Integrated Heterogeneous Systems. 06 2014.

[31] Samsung. Samsung SmartSSD - Samsungsemiconductor-us.com, Mar 2019. URL https://samsungsemiconductor-us.com/smartssd-archive/pdf/SmartSSD_ProductBrief_13.pdf.

[32] SiFive. SiFive HiFive Unleashed, . URL https://www.sifive.com/boards/hifive-unleashed.

[33] SiFive. Sifive hifive unmatched, . URL https://www.sifive.com/boards/hifive-unmatched.

[34] Dean Takahashi. Sifive moves into high-end RISC-V processors with P650 design, Dec 2021. URL https://venturebeat.com/2021/12/02/sifive-moves-into-high-end-risc-v-processors-with-p650-design/.

[35] Kenzo Van Craeynest, Shoaib Akram, Wim Heirman, Aamer Jaleel, and Lieven Eeckhout. Fairness-Aware Scheduling on Single-ISA Heterogeneous Multi-Cores. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, PACT '13, page 177–188. IEEE Press, 2013. ISBN 9781479910212.

[36] Ashish Venkat and Dean M. Tullsen. Harnessing ISA Diversity: Design of a Heterogeneous-ISA Chip Multiprocessor. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, page 121–132. IEEE Press, 2014. ISBN 9781479943944.

[37] David G. von Bank, Charles M. Shub, and Robert W. Sebesta. A Unified Model of Pointwise Equivalence of Procedural Computations. 16(6):1842–1874, nov 1994. ISSN 0164-0925. doi: 10.1145/197320.197402. URL https://doi.org/10.1145/197320.197402.

[38] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanović. The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA. Technical Report UCB/EECS-2011-62, EECS Department, University of California, Berkeley, May 2011. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-62.html.

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