Leveraging Processor-diversity for Improved Performance in Heterogeneous-ISA Systems

Yihan Pang

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

Master of Science
in
Computer Engineering

Binoy Ravindran, Chair
Changwoo Min
Cameron D. Patterson

Sept 27, 2019
Blacksburg, Virginia

Keywords: System Software, Heterogeneous Architectures, SIMD, Scheduling, ISA

Leveraging Processor-diversity for Improved Performance in Heterogeneous-ISA Systems

Yihan Pang

(ABSTRACT)

The purpose of this thesis is to investigate the effectiveness of executing High Performance Computing (HPC) workloads on multiprocessors with heterogeneous Instruction Set Architecture (ISA) cores. ISA-heterogeneity in processor designs provides a unique dimension for researchers to explore performance benefits through diversity in design choices. Additionally, each application has a natural preference to one processor in a selected group of processors (we defined this term as processor-preference), and processor-preference is highly affected by processor design choices. Thus, a system with heterogeneous-ISA cores offers an intriguing design perspective, packing heterogeneous-ISA cores in the same processor or system that compensate each other in dynamic workload scenarios. This thesis considers dynamic migrating applications with different processor-preferences across ISA-different cores to exploit the potential of this idea. With SIMD instructions getting more attention from chip designers, this thesis also presents the necessary modifications for a general compiler/run-time infrastructure to transform the dynamic program state of SIMD regions at run-time from one ISA format to another for cross-ISA migration and execution. Lastly, this thesis presents a processor-preference-aware scheduling policy that makes dynamic cross-ISA migration decisions that improve overall system throughput compared to homogeneous-ISA systems. This thesis prototypes a heterogeneous-ISA system using an Intel Xeon Gold 5118 x86-64 server and a Cavium ThunderX ARMv8 server and evaluates the effectiveness of our infrastructure and scheduling policy. Our results reveal that heterogeneous-ISA systems that are processor-preference-aware and with cross-ISA execution migration capability can yield throughput gains up to 36% compared to traditional homogeneous ISA systems.

Leveraging Processor-diversity for Improved Performance in Heterogeneous-ISA Systems

Yihan Pang

(GENERAL AUDIENCE ABSTRACT)

The author of this thesis has a family full of non-engineers. To persuade family members that the work of this thesis is meaningful, aka the author is not procrastinating in school, the author decided to draw an analogy between processors and cars.

Suppose in an alternative universe, cars (systems) can be powered by engines (processors) that uses two different fuel-sources (ISAs): gasoline or electric (single-ISA) processors but not both (heterogeneous-ISA). Car manufacturers (chip designers) can build engines with different design choices (processors with varying design options): engines combined with turbochargers for gasoline-powered cars, high-performance batteries combined with energy-efficient batteries for electric-powered cars (added extended instruction sets, CPU designs that target vastly different use cases, etc.). However, each design choice is limited to improving performance for a specific type of fuel-source based engine. For example, having battery alternatives has no performance impact on gasoline-powered engines. As time passes by, car manufacturers have exhausted options to make a drastic improvement to their existing engine designs (limited performance gains in recent chips).

To tackle this problem, in this thesis, the author first examined the usage of cars: driving on the road (running applications). The author's study found that no single engine is suitable for all routes (no single processor is good for all workloads), and cars powered by different fuel-source based engines showed a significant diversity in performance (application performance varies drastically between systems with processors built on different ISAs).

Gasoline-powered cars perform well on high-speed roads, whereas electric-powered cars perform well on low-speed roads. Unfortunately, in real life, a person's commute (a workload of applications) consists of a mixture of high-speed roads and low-speed roads, and one cannot know the exact percentage of each kind of path they travel (exact application composition in a workload) beforehand. Therefore it is challenging for a person to make the correct car selection for the incoming commute (choose the right system for a workload).

This thesis tries to solve this commuting problem by building a car that has multiple engines fitted to suit different road needs (systems with processors that have vastly different use cases). This thesis looks at a particular dimension of combining various fuel-powered engines in the same car (a system with heterogeneous-ISA processors). The author believes that adding diversity in fuel-powered engine selections provide an exciting dimension in car design choices (adding ISA-heterogeneity in processors provide a unique dimension in system design). Thus, this thesis focuses on estimating a theoretical multi fuel-powered car's performance by combining two different fuel-powered cars into a single mega-car using some framework (Popcorn Linux). This framework allows this mega-car to be driven by a combined fuel source with fuel intake freely transfer between fuel-sources (cross-ISA migration and execution) based on road conditions (application encountered). Based on the evaluation of this new prototype, the author finds that in a real-life scenario (workload with mixed application combination), cars with multiple fuel-source based engines have better performance than two single fuel-source based cars (systems with heterogeneous-ISAs processors perform better than systems with homogeneous-ISAs processors). The author hopes that this study can help build the foundation for the development of hybrid cars (system with heterogeneous-ISAs in the same processor) in the future as well as the consideration of modifying existing car into a mega-car with multiple engines suited for different road needs for improved commute performance for now.

Ultimately, this thesis is not about cars. The author hopes that by explaining the research done in this paper through cars, general audiences can understand what this work is trying to investigate and what solution they have provided. In this work, we investigate the potential of a system with heterogeneous-ISA processors. This thesis prototypes one such system and finds that heterogeneous-ISA systems have performance benefits than traditional homogeneous-ISA systems over a series of experiment evaluations.

Dedication

To my family and friends.

Acknowledgments

First and foremost, I owe my deepest gratitude to my advisor Dr. Binoy Ravindran. Dr. Ravindran opened the research door for me and helped me transition from a student to a researcher. You have shown me patience and encouraged me to pursue a diverse range of research topics. I learned so much from these experiences. I am forever grateful. I want to also thank the members of my M.S advisory committee, Dr. Changwoo Min and Cameron D. Patterson. Their advice on research-related topics helped me to formulate my thesis in the correct direction.

My fellow lab mates at the System Software Research Group (SSRG) had a significant influence on the success of my research. Their encouragement and knowledge helped me identify many mistakes in my earlier designs. In particular, without Dr. Robert Lyerly's help on the Popcorn Linux framework, this research will likely never succeed in time. I wish Dr. Lyerly's the best. In addition, I also want to thank my friends and my mentor (Dr. Xun Jian) in the High-performance, Energy-efficient, Assured Processing (HEAP) Lab at the CS department. My research collaboration with them helped me to understand how to formulate a research idea into a conference paper efficiently. The lessons I learned in their group helped me significantly in publishing this work.

Furthermore, I want to thank all the friends who I met during my eight-year stay in Blacksburg. Without your presence in my life, I have no idea how I will survive in an isolated location like Blacksburg.

And Finally, I would like to thank my parents. They deserve so many more credits and thanks than they will ever receive. Their decision to sent me to America for better-suited education twelve years ago started this incredible knowledge-pursuing journey. They took care of everything back home so that I can fully pursue my studies abroad.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 1 helps the reader to get a better understanding of the necessity of the researches conducted in this thesis. This chapter discusses the motivation of investigating heterogeneity at the ISA level in processor designs. In addition, the chapter states the target problems and lays out the thesis's contribution. Lastly, for readers to better navigate this thesis, the thesis organization is also provided.

This chapter starts with Section 1.1, discussing the motivations behind our research. The problem statement is explained in Section 1.2, followed by a list of contributions in Section 1.3. This chapter ends with Section 1.4 illustrates the thesis organization.

1.1 Motivation

For the last half of the 20th century, Intel Co-founder Gordon Moore's prediction of the number of transistors on a chip, also known as Moore's Law, has set a loose guideline for predicting future CPU performance [120]. This simple, yet influential forecast has been solidified over the years as chip manufacturers consistently produce more and more powerful CPUs through packing an increasing amount of transistors.

Regrettably, this trend cannot last forever. With more and more transistors crammed into a small area, problems such as heat [2, 118], technology barriers for shrinking transistors size [104], and increasing manufacturing costs [121] are some of the main factors that have contributed to the limited performance gains of recent chips [17, 116]. Figure 1.1 is a plot of microprocessor trends from 1975 to 2015 done by several scientists [86]. One of the main conclusions drawn from this graph is that adding more transistors is not the panacea for improving chip performance anymore. This diminishing return in chip performance is causing many to claim that the "end of Moore's Law" is coming [17, 30, 104, 108, 116, 121]. These phenomena have, therefore, forced chip designers to advance performance and energy efficiency boundaries elsewhere.

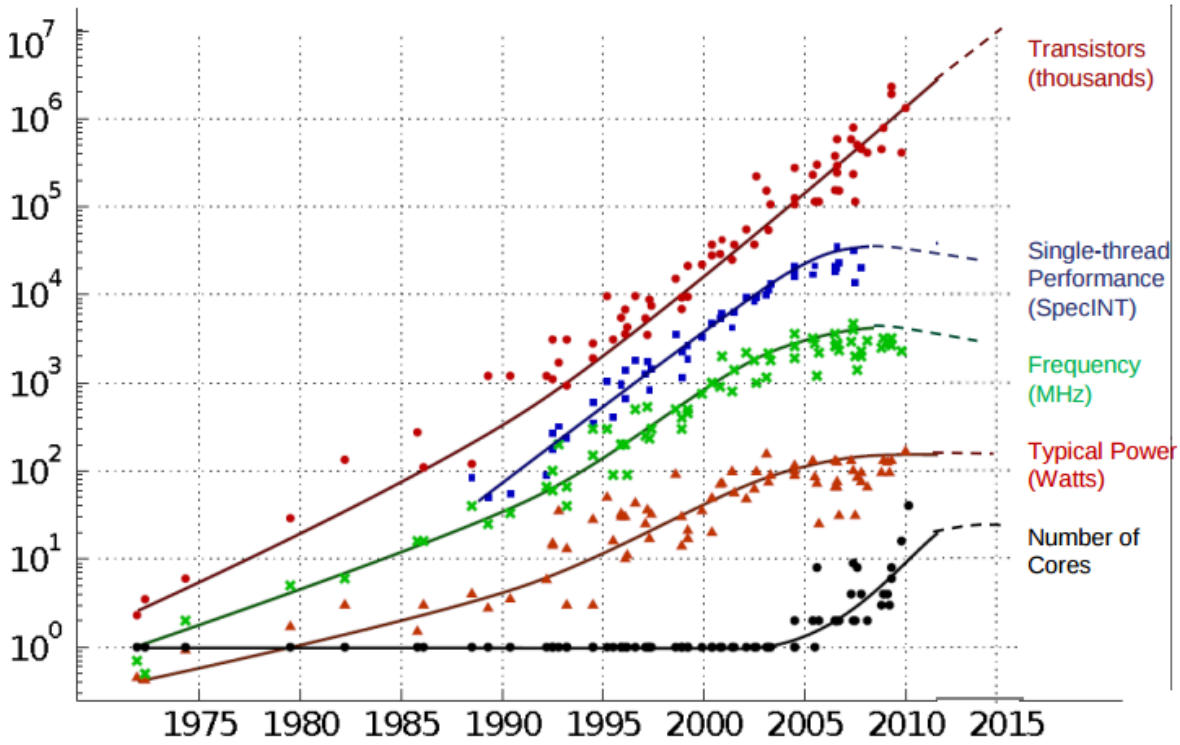


Figure 1.1: CPU trends for the last 35 years. Original Data shown in [86].

Thus, in recent years, the computer architecture landscape has seen the rise of systems integrating heterogeneous architectures as a possible solution to deal with the "end of Moore's Law" [17, 30, 104, 108, 116, 121]. Chip designers have explored the pairing of CPU designs that target vastly different use cases. For example, ARM's "big.LITTLE" technology [84] and its successor, "DynamIQ" [78], couple cache-coherent "big" cores (high clock speeds, advanced micro-architecture) for latency-sensitive workloads with "little" cores (high energy efficiency) for background and low-priority tasks. In the server space, Intel "Xeon-Xeon Phi" systems [31] integrate a small number of high-performance cores with many low-power cores to accelerate different workloads. Intel has also released plans for a heterogeneous x86 architecture with one big Sunny Cove core and multiple small Atom cores, which use a new 3D stacking technology [45, 93, 103, 126]. Some designers even expanded their exploration space further by exploring performance gain through other components in a computer system. For example, increasing performance through in or near memory computing has been a rising topic in the scientific community [73, 111].

Unfortunately, existing CPU designs do not have heterogeneity at the Instruction Set Architecture (ISA) level for general-purpose CPUs. At best, some cores used in these designs support extended instruction sets, but at their base, all cores share the same ISA (e.g., x86-64 ISA for "Xeon-Xeon Phi" [31], AArch64 for ARM "big.LITTLE" [84] and ARM "DynamIQ" [78]). To date, there have not been any commodity-scale systems that have processors with heterogeneity at the ISA level (a notable exception is MPSoCs for embedded systems [48]). The lack of heterogeneity in today's systems is also reinforced by the fact that most data centers consist only of servers with x86-based processors [53, 80, 101, 105]. Restricting processors to a single ISA eliminates a vital dimension of processor design. Fortunately, the research community has been exploring experimental heterogeneous-ISA processor designs, showing that they provide better performance and energy efficiency

than single-ISA heterogeneity. New innovative designs – shared-memory chip multiprocessors [12, 114, 116], independent cache-coherent domains processors [57, 72], and cores using a superset ISA [115] – span across many settings, ranging from cluster architectures [91] to mobile environments [68].

Recently, the industry has become more and more welcoming to the idea of heterogeneity in server systems. With the advent of ARM-based high-end servers [25, 109, 110], capable of powering high-performance computing (HPC) applications, third-party organizations, such as data-center providers and cloud providers, are increasingly integrating machines of different ISA families in their computing installations [5]. Chip vendors are also integrating CPUs of different ISA families in the same SoC or on the same platform. For example, the Intel Skylake processor with in-package FPGA [32, 52] is capable of synthesizing RISC-V and x86 soft cores; smart NICs integrate ARM [40, 88] or MIPS64 [82] on the same platform.

Heterogeneity brings an additional dimension in processor-diversity. This work studies one set of processor-diversity, processors built with different ISAs, to determine whether processors with heterogeneity at ISA-level can be leveraged for improved performance. To our knowledge, in the server space, no one has extensively investigated the impact of heterogeneity at the ISA-level on CPUs in terms of performance and cost.

To reveal the performance difference between server systems with different ISAs, we ran applications from four popular HPC benchmark suites and calculated the slowdown of each single-threaded benchmark in four different system settings. The selected benchmarks include the NAS Parallel Benchmark (NPB) suite [11], Phoenix Benchmark suite (PHX) [100], Livermore Loops suite (LL) [83], and Test Suite for Vectorizing Compilers (TSVC) [19]. We ran two experiments that compared servers with the same type of ISA as a baseline: Cavium ThunderX v.s. Marvell ThunderX2, and Intel Xeon Silver 4110 v.s. Intel Xeon Gold 5118. We then ran two additional experiments that compared servers with different ISAs: Cavium

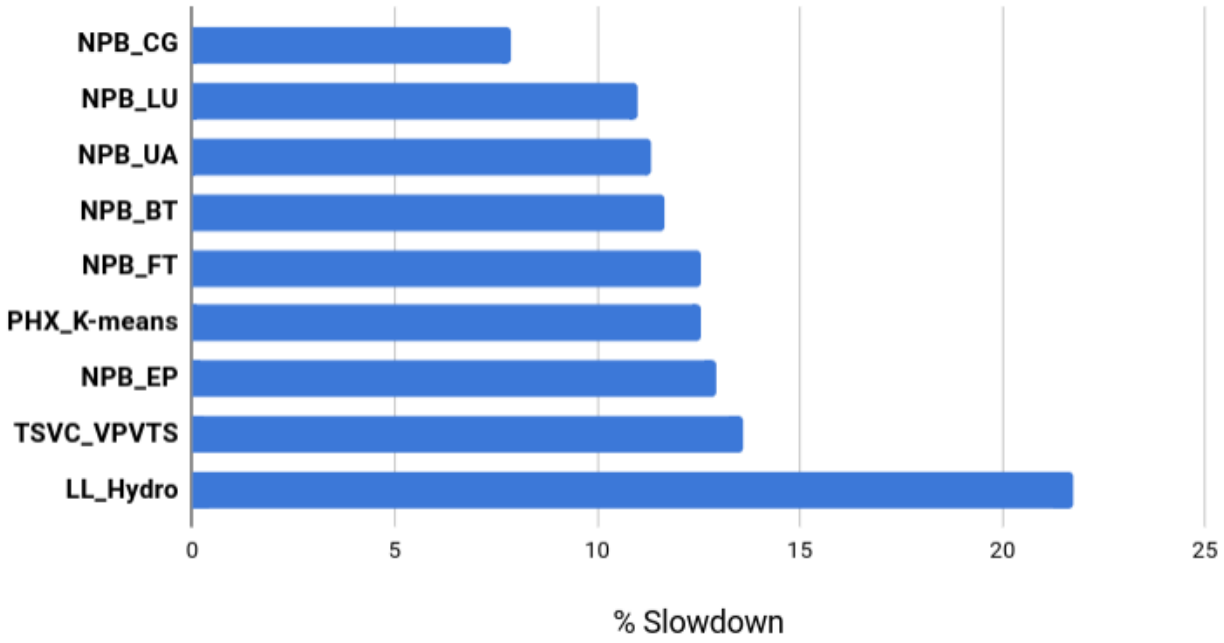


Figure 1.2: Percentage slowdown of NAS Parallel Benchmarks [11], Livermoore Loops [83], Phoenix [100], and Test Suite for Vectorizing Compilers [19] when running on a Intel Xeon Silver 4110 core v.s. a Intel Xeon Gold 5118 core.

ThunderX v.s. Intel Xeon-Gold 5118 and Marvell ThunderX2 v.s. Intel Xeon-Gold 5118. For all servers, we used factory-specified settings.

Figure 1.2 and Figure 1.3 show the performance differences between servers with the same type of ISAs. For servers belonging to the same generation (Intel Xeon Silver 4110 and Intel Xeon Gold 5118), application performance varied within 20%. For servers that were released between different generations (Cavium ThunderX and Marvell ThunderX2), most applications' performance varied within 100%. This result is not surprising, considering that ARM-based servers just entered the market and have not optimized thoroughly compared to their x86 counterparts. Thus, different generations of ARM chips likely yield more significant performance improvement.

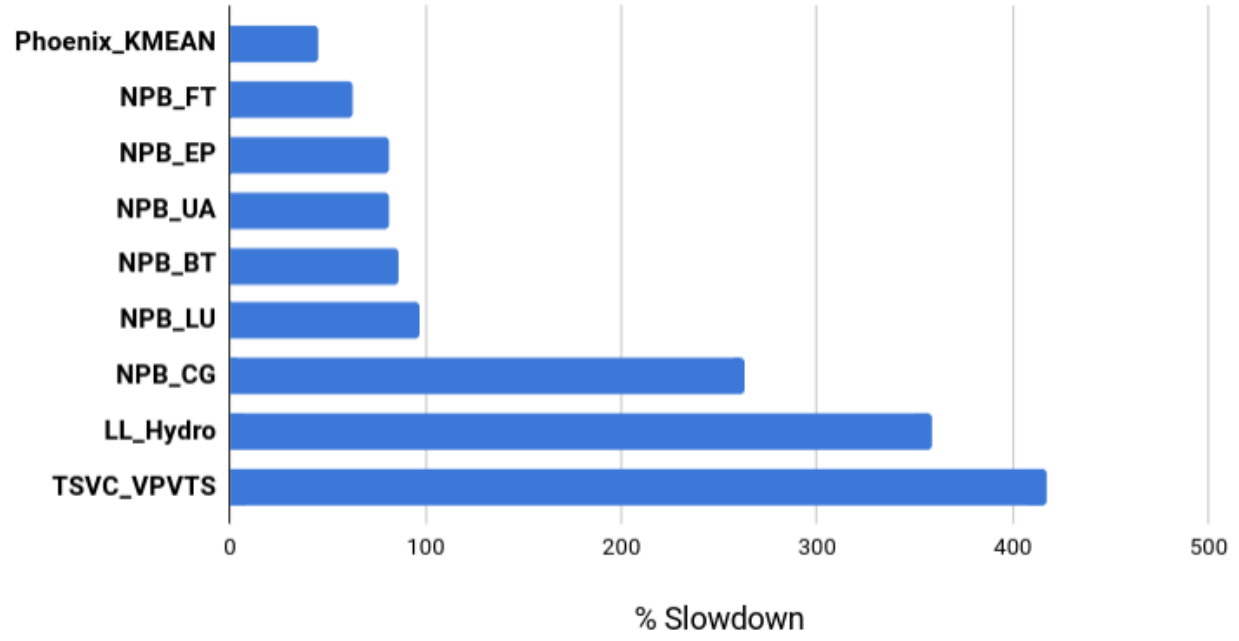


Figure 1.3: Percentage slowdown of NAS Parallel Benchmarks [11], Livermoore Loops [83], Phoenix [100], and Test Suite for Vectorizing Compilers [19] when running on a Cavium ThunderX core v.s. a Marvell ThunderX2 core.

However, the slowdown difference between servers with the same type of ISA is negligible when compared to servers with different types of ISAs. Figure 1.4 shows the relative performance of single-threaded applications using one core of the Cavium ThunderX machine (96 cores, ARMv8 ISA) [109], in comparison to one core of the Intel Xeon Gold 5118 machine (12-core/24-thread, x86-64 ISA) [52]. Figure 1.5 shows the same performance compared to the same Intel core with a newer ARM model, Cavium Thunder X2. In both figures, the minimum application slowdown is more than 200%, with some reaching more than 2000% slowdown (VPVTS in Cavium ThunderX v.s.Intel Xeon Gold 5118). Based on these preliminary results, heterogeneity at ISA-level does bring significant processor diversity as application performance varies drastically on processors built with different ISAs.

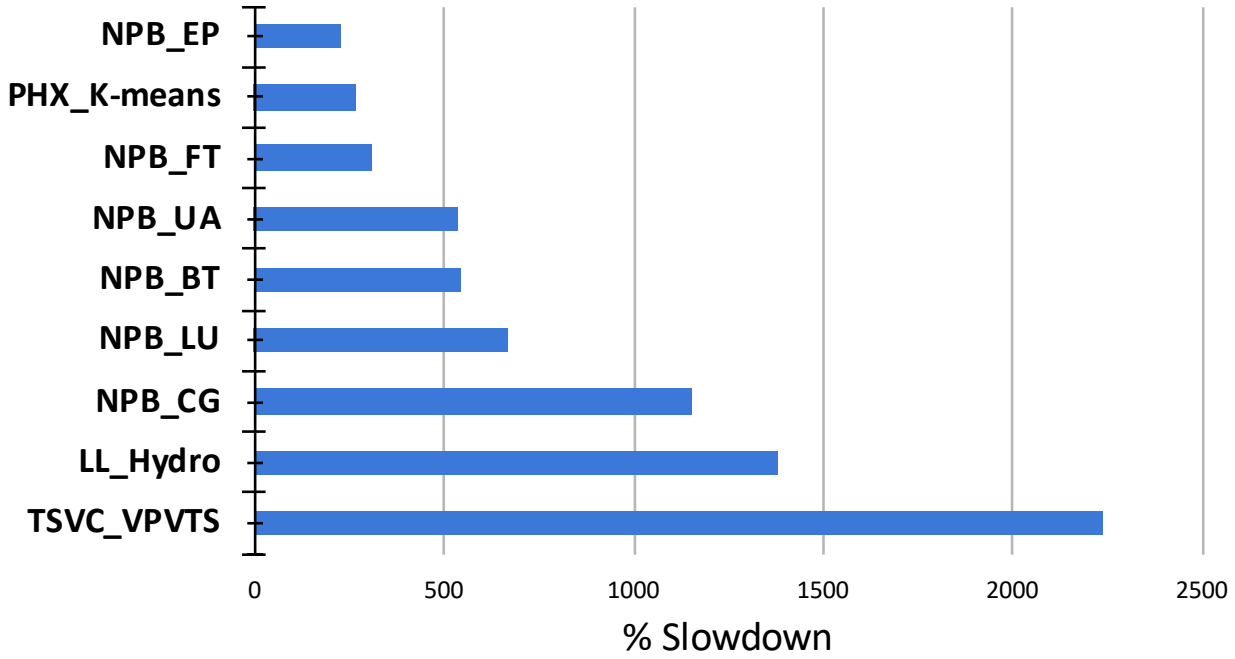


Figure 1.4: Percentage slowdown of NAS Parallel Benchmarks [11], Livermoore Loops [83], Phoenix [100], and Test Suite for Vectorizing Compilers [19] when running on a Cavium ThunderX core v.s. an Intel Xeon Gold 5118 core.

Figure 1.4 and Figure 1.5 also reveal that all applications are slower on either ARM servers but differ in the degree of slowdown. It is worth noting that the ARM core slowdown is not surprising, given that each Xeon core is clocked faster, and both ThunderX and ThunderX2 cores have different processor design goals, trading off single-core performance for massive parallelism. As a result, there are significantly more ARM cores (48 cores in ThunderX and 28 cores in ThunderX2) in the ThunderX CPU family.

Based on the previous four experiments, if processor design choices solely determined based on application performance, processors with heterogeneity at ISA level have no performance advantage over homogeneous-ISA processors. System architects with unlimited resources should always choose the faster x86-based core. Unfortunately, the system cost is a servere constraint in system designs. Processor design choices heavily depend on the profit of the final product, and most system architects start with limited budgets. Thus, to convince

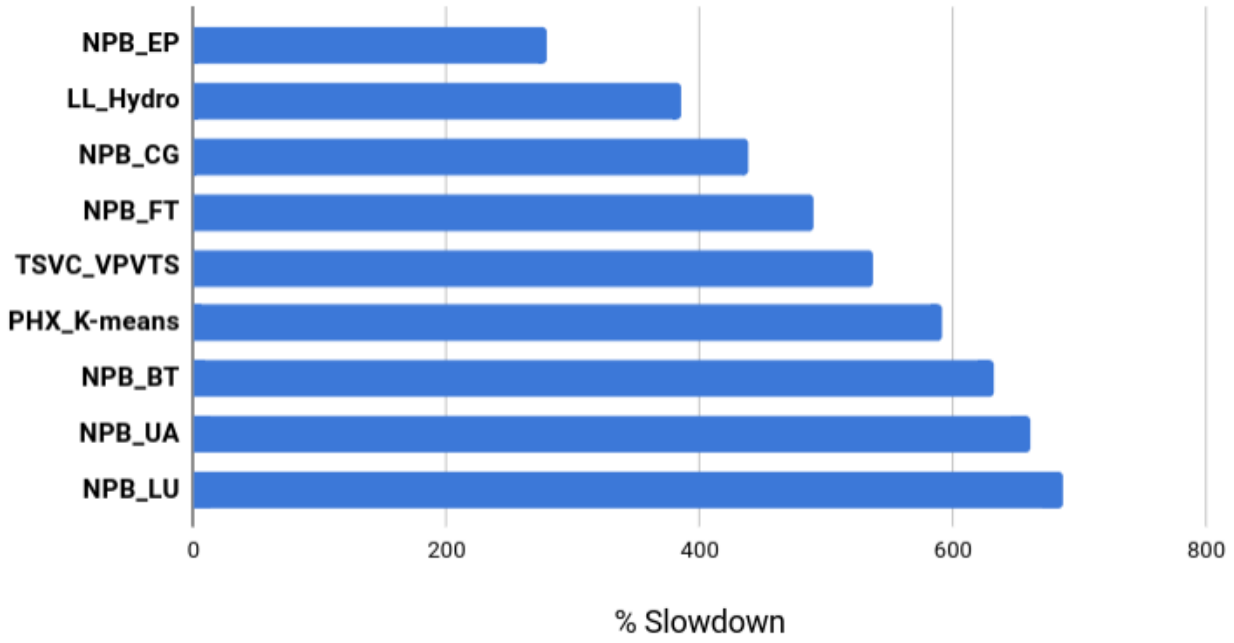


Figure 1.5: Percentage slowdown of NAS Parallel Benchmarks [11], Livermoore Loops [83], Phoenix [100], and Test Suite for Vectorizing Compilers [19] when running on a Marvell ThunderX2 core v.s. an Intel Xeon Gold 5118 core.

industry leaders to consider using ISA-heterogeneity in their future processor designs, one must convince that packing heterogeneous-ISA processor designs can have better a profit margin than existing homogeneous-ISA based processor designs. In this paper, the profit margin is mathematical formula defined as :

$$\frac{\Delta of Performance for New Design}{\Delta of Cost for New Design} > 1$$

Fortunately, the price of x86-based cores and ARM-based cores are very different. Table 1.1 shows the cost of each CPU used in this experiment. The unit price of x86-based cores is usually higher than their ARM counterparts. With the combination of varying chip costs and performance slowdown, it is hard not to ask the question: is there an advantage of using a heterogeneous-ISA system to improve performance without sacrificing cost?

Chip	ISA	Core Count	Year	Unit Price	Adjusted	$/HT
Intel Xeon Silver 4110	X86	8/16 HT	2017	$501 [50]	$501	$31.31
Intel Xeon Gold 5118	X86	12/24 HT	2017	$1273 [51]	$1273	$53.04
Cavium ThunderX CN8890	ARMv8	48	2016	$795 [20]	$607	$16.56
Marvell ThunderX2 CN9980	ARMv8	32/128 HT	2018	$1795 [58]	$1662	$14.02

Table 1.1: Chip Unit Price Comparison.

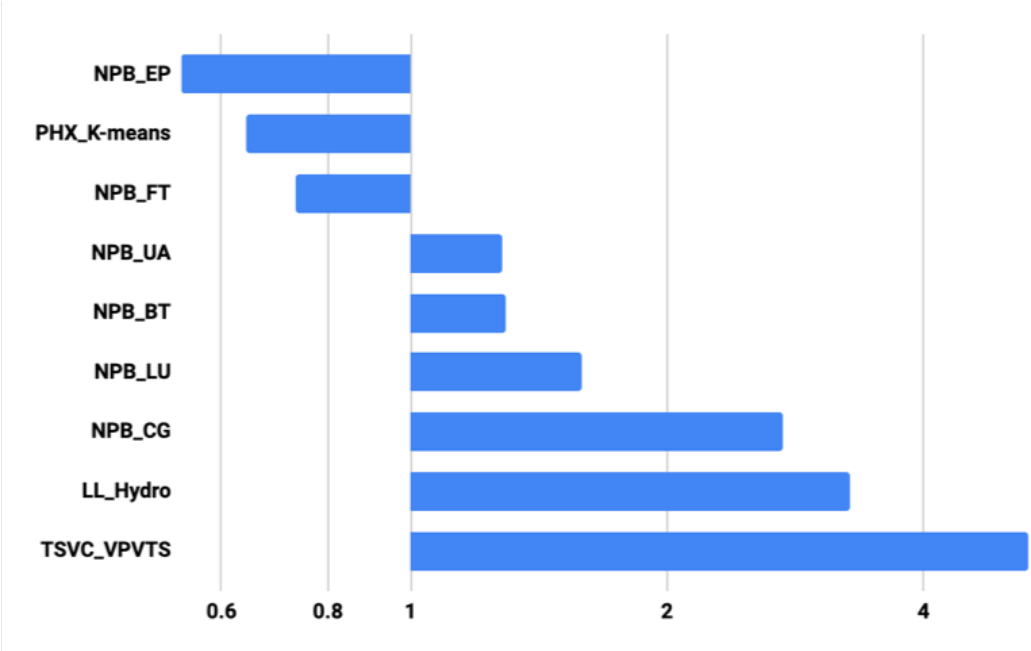


Figure 1.6: Profit margin of NAS Parallel Benchmarks [11], Livermoore Loops [83], Phoenix [100], and Test Suite for Vectorizing Compilers [19] when running on a Cavium ThunderX core v.s. an Intel Xeon Gold 5118 core.

Figure 1.6 shows the profit margin of the Cavium ThunderX vs. Intel Xeon Gold 5118 single-core performance comparison. This time, heterogeneity at ISA-level shows promising signs. Based on the mathematical formula, a profit margin of fewer than one means for every dollar spent, you are getting less than one dollar worth of production. In short, it is not profitable to run on the faster Xeon Gold core. In this figure, three benchmarks experiences profit margin lower than one with three more have a profit margin slightly over one. This evaluation shows that the processor-diversity brought by the use of heterogeneous-ISAs can potentially improve performance within the same budget.

1.2 Problem Statement

Venkat et al. [114] and Akram et al. [3, 4] are some of the first pioneers studying the impact of ISAs on performance. Their research shows that ISAs can affect application performance, and applications have ISA-affinities: each application has a natural ISA preference. However, each processor is designed differently. It is almost impossible to calculate the ISA's impact on application performance in a real system. Many factors (number of stages in the pipeline, cache size, etc.) can be attributed to the performance slowdown. Thus, to be more precise, this thesis decided to define and use the phrase processor-preference instead of ISA-affinity. Processor-preference means that each application has a natural processor-preference in a selected group of processors. The performance impact of each processor design decision (e.g., what ISA to use) is ignored; this thesis instead focuses on the overall processor performance that includes all design decisions. In this case, all applications tested earlier have processor-preference to the x86-based Xeon Gold 5118 processor. This research investigates the fluctuation of processor-preference between applications, and this fluctuation of processor-preferences matters in heterogeneous system designs. An application experiencing a 2000% slowdown and another application experiencing a 200% slowdown both have processor-preference for the faster core. Still, the benefit they get by executing on the faster core is widely different (20X speedup v.s. 2X speedup). The lower speed up gain from applications with low processor-preference preference can sometimes be mitigated by running them on cheaper and slower cores in parallel. In our case, ARM-based processors are designed with more cores in a system and with lower unit prices. This diversity in processor designs means that, for applications that experience low processor-preference (applications that are experiencing low slowdowns when running on non-optimal processor cores), it is not immediately clear which system provides the best throughput within the similar price budget.

Given the fact that x86-based servers dominate data centers today [12], this begs the question, are x86-based servers indeed the clear winner for running diverse workloads within a similar price budget?

To find out if there is a clear winner in throughput when running diverse workloads, we performed another study. We selected five benchmarks, EP, FT, BT, LU, and Hydro, based on their respective slowdown results from Figure 1.4. We ran them with a variety of different composition ratios for a given workload on a pair of Cavium ThunderX and Intel Xeon Gold 5118 servers used for the earlier slowdown experiments. Our Cavium ThunderX server has two processors; thus, the total cost amounts to $1214, which is within a similar price range to Xeon 5118 ($1273). These five benchmarks are chosen because they represent benchmarks that experience various degrees of slowdown, hence, benchmarks with different degrees of processor-preference. EP and FT suffered relatively low slowdown (2-3X) on the ARM servers; BT and LU suffered a slightly higher slowdown (5-6X); whereas, Hydro suffered the most substantial slowdown among all with more than 20X slowdown in performance. For ease of comparison, we treated EP and FT as the same type of benchmark (low processor-preference benchmarks); the same also applies to BT and LU (medium processor-preference benchmarks). Therefore, there will be a single ratio for both types of benchmarks.

Figure 1.7 illustrates how our experiments are performed. For each scenario, a workload batch with a predefined ratio of each benchmark is generated first in a vanilla Linux Kernel. Then some randomly selected benchmarks from this newly created workload batch are executed until all available physical threads are occupied without overloading the CPU. Afterward, whenever a running benchmark finished execution, another random benchmark from the remaining workload batch was selected and executed. If the workload batch in waiting was empty, a new workload batch with the same predefined ratio was generated again. This process repeated until the evaluation period ended. To ensure a fair comparison, we

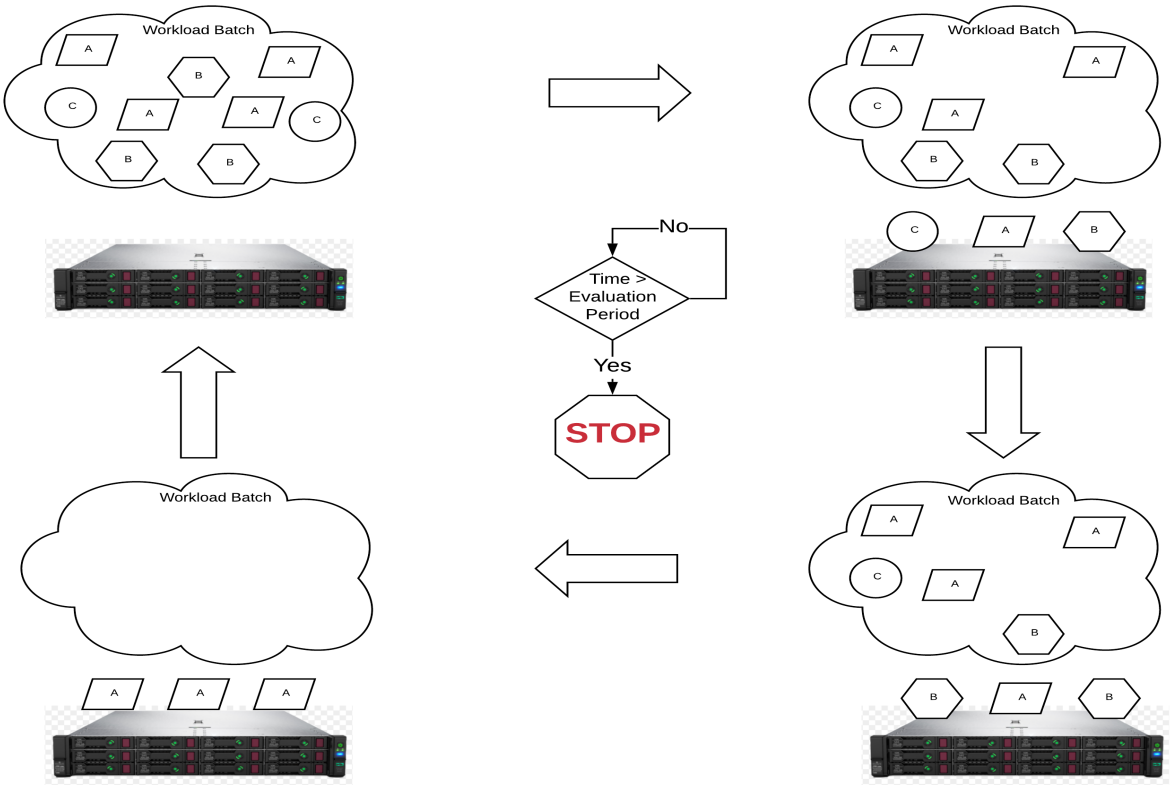


Figure 1.7: Experiment procedures.

used the same random seed so that each configuration had the same benchmark selection outcome for every run. Each experiment was run for 75 minutes. The rationale behind this evaluation time selection was that most of the benchmarks execute in about 3 to 5 minutes when running on the server with Intel Xeon Gold 5118 CPU; 75 minutes was long enough to mitigate the impact of noise. In this experiment, we compared the system throughput of each scenario, i.e., *the number of benchmarks completed in 75 minutes.*

Figure 1.8 shows the result of our study. Neither the x86-based nor the ARM-based servers had a clear advantage in all tested scenarios. x86-based servers have a higher throughput when there is an increasing number of relatively high processor-preference benchmarks (BT/LU/Hydro). In contrast, ARM-based servers perform better when the workload is

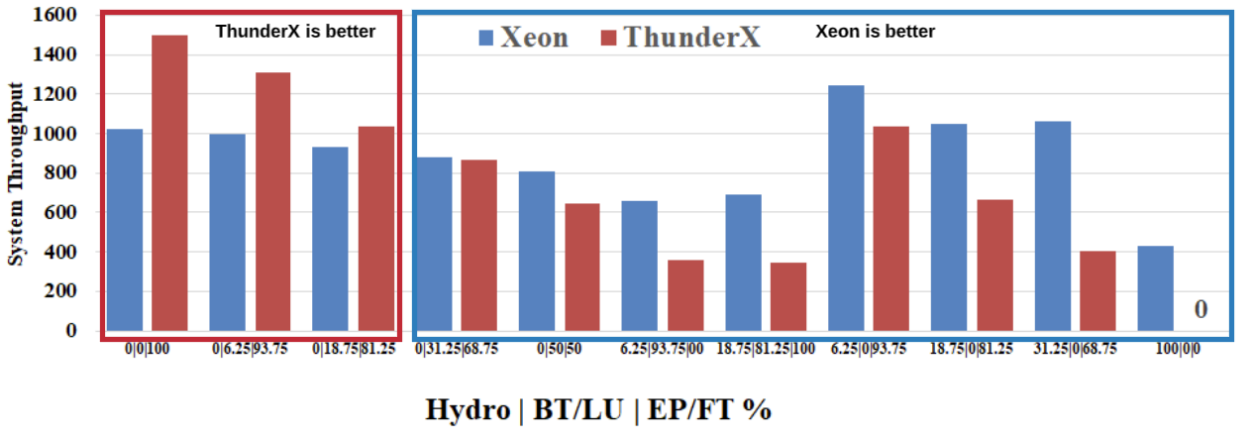


Figure 1.8: System throughput under different ratios of EP/FT [11], BT/LU [11], and Hydro [83] on two Xeon servers and two ThunderX servers.

mainly composed of low processor-preference benchmarks (EP/FT).

These results again reinforce the idea that each application has a natural processor-preference. Still, their processor-preference may not be all that significant to have a de facto processor for all workloads with costs accounted for. Thus, how to maximize harnessing processor-preference may be an exciting research question and may provide a quality solution to chip designers in the future. Due to the fact there is no commercial heterogeneous-ISA chip multiprocessor with cache-coherent shared memory available in today's market, we used the Popcorn Linux Framework [12] to provide the basis for us to explore processor-diverse heterogeneous-ISA servers. Popcorn Linux allowed us to emulate a heterogeneous-ISA server through high speed interconnect. In Chapter 2, we will give a detailed example of how the Popcorn Linux Framework achieved cross-ISA migration. Because of the engineering work required and the lack of human resources, this framework has not been able to port to all machines available to us. Currently, Cavium ThunderX is the only ARM server that is capable of using the Popcorn Linux Framework. Thus, to show that harnessing processor-preference in processor diverse heterogeneous-ISA systems is beneficial, we want to focus on investigating the performance slowdown for Cavium ThunderX vs. Intel Xeon Gold 5118, as

shown in Figure 1.4 and identify bottlenecks and provide a solution that leverages processor-preference. Due to the similar price range between the two servers, the performance increase will indicate a better profit margin. Lastly, We hope that our research will provide a valuable case study for future researchers to improve system performance through leveraging processor-diversity in heterogeneous-ISA systems.

Further inspection of the highest slowdown benchmarks of Cavium ThunderX in Figure 1.4 revealed that benchmarks such as Hydro and VPVTS contain significant portions of single-instruction-multiple-data (SIMD) instructions, accounting for more than 80% of benchmark execution time. A more detailed explanation of why SIMD instruction-dominated programs have a processor-preference to x86-based processors than ARM-based processors will be provided in the background section of this thesis. Unfortunately, based on our findings, the original Popcorn Linux Framework does not support the migration of SIMD instructions.

With SIMD instructions gaining more attention from chip designers as a means to extract additional data parallelism in various application domains (e.g., HPC [10], ML [98, 122], computer vision [29, 92], cryptography [6, 61, 119], and other domains [37, 125]), coupling together heterogeneous machines that have significant differences in processor designs and SIMD extensions provides an exciting platform for running diverse applications. This unique system pairing also reinforces the need for a commodity heterogeneous-ISA chip multiprocessor with cache-coherent shared memory in the future. Thus, to fully enable research in leveraging processor-preference in processor-diverse heterogeneous-ISA systems, we need first to allow the Popcorn Linux Framework to support cross-ISA SIMD migration.

For designers aiming to harness processor-preference to optimize the performance in a processor-diverse heterogeneous-ISA system with a controlled budget, this raises interesting questions:

1. *How should a workload consisting of applications with a mixture of processor-preferences be scheduled to maximize throughput?*
2. *What is the impact of an application's processor-preference on the execution of co-executing workloads?*
3. *Are there throughput advantages in migrating applications across ISAs based on their processor-preference?*
4. *Are there advantages of adding the dimension of ISA-diversity to future processor designs?*

1.3 Thesis Contributions

In this thesis, we investigated a specific dimension of system design choices, a system with heterogeneous-ISA chip multiprocessors. Such processor design diversity allowed us to explore scheduling batch workloads consisting of applications with various degrees of processor-preference on heterogeneous-ISA systems. We analyzed the effects of how applications with different processor-preference interact on both x86 and ARM systems. Additionally, we analyzed the impact of processor-diversity on system utilization, including where applications should be scheduled to maximize throughput. To conduct this investigation, we extended Popcorn Linux [12], an OS/compiler/run-time system framework, for executing and migrating shared-memory applications across non-cache-coherent heterogeneous-ISA CPUs.

This thesis makes the following contributions:

- We developed a cross-ISA SIMD migration compiler/run-time framework that enables applications containing SIMD instructions to be migrated between heterogeneous-ISA CPUs with different SIMD register widths. The framework is built as an extension of Popcorn Linux's [12] compiler/run-time system infrastructure.
- We analyzed the effects of co-executing applications with different processor-preferences on a heterogeneous-ISA system to understand the impact of processor-preference and workload composition on system throughput.
- We used insights gained from our analysis and developed a processor-preference-aware scheduler that monitors system workload and migrates applications dynamically. When used on our prototype heterogeneous-ISA system, our evaluations revealed up to 36% throughput gains over the next best homogeneous-ISA system within a similar price budget.
- We showed that harnessing processor-diversity in heterogeneous-ISA systems have performance improvement through a case study of pairing an Intel Xeon Gold 5118 to a Cavium ThunderX server using the Popcorn Linux Framework.
- We demonstrated that by having diversity in system processor choices, there exists a possibility of leveraging processor-preference for improved performance.
- We reinforced the support for developing a commodity heterogeneous-ISA chip multiprocessor with cache-coherent shared memory.

1.4 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 presents the necessary background on SIMD instruction and LLVM Infrastructure. For completeness, we also summarize Popcorn Linux's [12] cross-ISA execution migration infrastructure. A Survey of related work is discussed in Chapter 3. Chapter 4 describes our method to enable basic cross-ISA SIMD migration, and Chapter 5 illustrates a more optimized approach. Chapter 6 details our processor-preference-aware scheduler. Chapter 7 illustrates our experimental setup and presents our results. Conclusion and future research direction is discussed last in Chapter 8.

Chapter 2

Background

This chapter presents relevant background information to understand this thesis. This chapter contains information on how SIMD instruction is used in today's system as a means to extracting additional data parallelism and why SIMD applications in our current system setups show extreme high preference to x86 ISA-based processors in our existing experiments. Chapter 2 also provides a brief overview of major components in LLVM compiler infrastructure to give the audience a clear picture of where our modifications are done. Because our design built on top of the Popcorn Linux Framework [12], we also explain how the Popcorn Linux Framework achieved cross-ISA migration.

This chapter starts with information on SIMD instructions in Section 3.4, followed by a brief overview of LLVM components in Section 2.2 and ends with the explanation of cross-ISA execution migration on Popcorn Linux in Section 2.3.

2.1 SIMD Instruction

Single instruction multiple data(SIMD) is one of the four classifications in Flynn's taxonomy. A processor that supports SIMD instructions can issue a single SIMD operation that operates on multiple data elements. Figure 2.1 illustrates the difference between traditional scalar processing v.s. SIMD processing. SIMD instructions help both execute multiple operations in parallel and amortize CPU core front-end costs (instruction cache pressure, decoding/issue

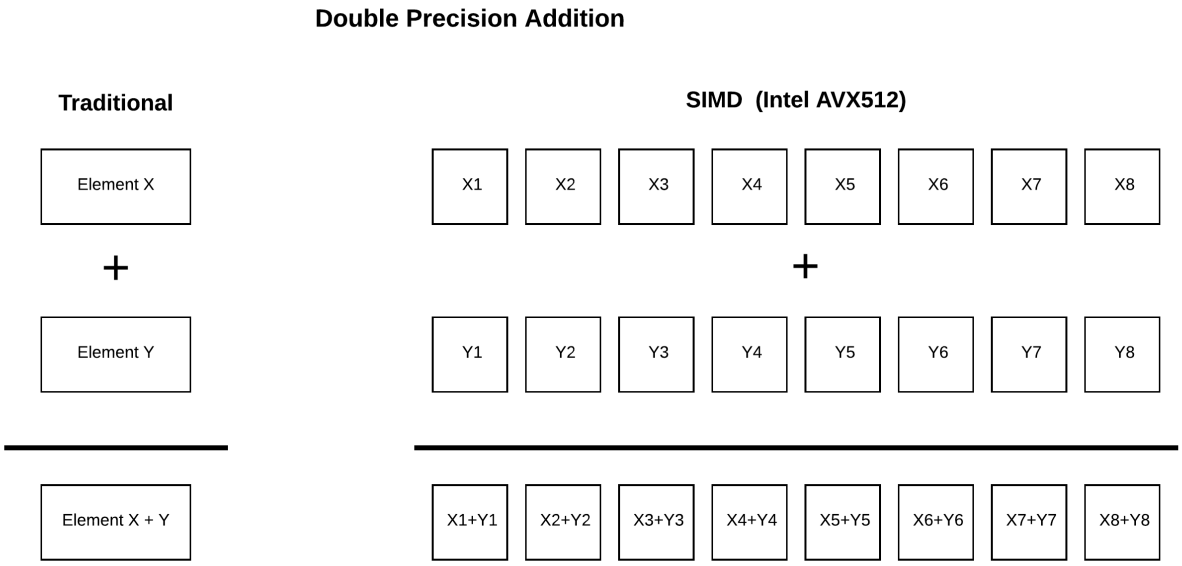


Figure 2.1: Scalar Add v.s. SIMD Add.

latency) and is thus an attractive method of extending the ISA to the chip designers. For example, Intel continues to widen its SIMD variant, AVX vector extensions, to 512 bits [49], and add new capabilities such as neural network instructions [18].

Similarly, ARM has introduced its SIMD support, Scalable Vector Extension [8], with width-agnostic instructions to complement its existing NEON SIMD extension [7]. ARM NEON SIMD extension with has a SIMD register width of 128 bit. The four times difference in SIMD register width (512 bits v.s. 128 bits) allows x86-based CPU with AVX512 extension to calculate four-times more element per SIMD instruction than its ARM counterparts. This difference in calculation power explains why SIMD-instruction dominated applications such as Hydro and VPVTS have a higher preference toward x86 ISA-based processor.

Extending SIMD register width does not come for free – since the processor executes multiple instructions at once, cores consume significantly more power. Thus the processor must reduce clock speeds to avoid overheating the chip. Intel's processor is a perfect example

Mode	Base	Turbo Frequency/Active Cores			
		1-2	3-4	5-8	9-12
Normal	2.3 GHz	3.2 GHz	3.0 GHz	2.9 GHz	2.7GHz
AVX2	1.9 GHz	3.1 GHz	2.9 GHz	2.6 GHz	2.3 GHz
AVX512	1.2 GHz	2.9 GHz	2.4 GHz	1.8 GHz	1.6 GHz

Figure 2.2: CPU Frequency Table for Intel Xeon Gold 5118 [1].

of this trade-off. Intel CPUs use dynamic voltage frequency scaling (DVFS) to change CPU frequency during runtime. When the CPU encounters SIMD-intensive code, it scales down the CPU frequency (known as AVX frequency) dramatically in order to limit power consumption and reduce heat this lower frequency. Table 2.2 shows the CPU frequency under different SIMD workloads for the Intel Xeon Gold 5118 CPU used in this thesis. Despite the extra heat generated, the benefits of using SIMD instructions (up to 16 times more calculations per SIMD instruction than scalar instructions) outweighs the benefit of using traditional scalar instructions(having approximately 69% more CPU frequency under full workload). Therefore, extending the Popcorn Linux Framework to support SIMD extension is still necessary as we need the capability to migrate high x86 processor-preference SIMD applications away from ARM-based servers.

2.2 Brief LLVM Overview

LLVM is an industrial-grade compiler that is widely embraced by the academic community. Although there are other compiler options available [36, 106], we decided to stick with LLVM in order to reduce unnecessary engineering efforts(the original Popcorn Linux compiler is modified on-top of LLVM). Due to LLVM's complexity, this thesis will only explain two aspects of LLVM: an overview of major components and built-in Profile Guided Optimization (PGO) capability.

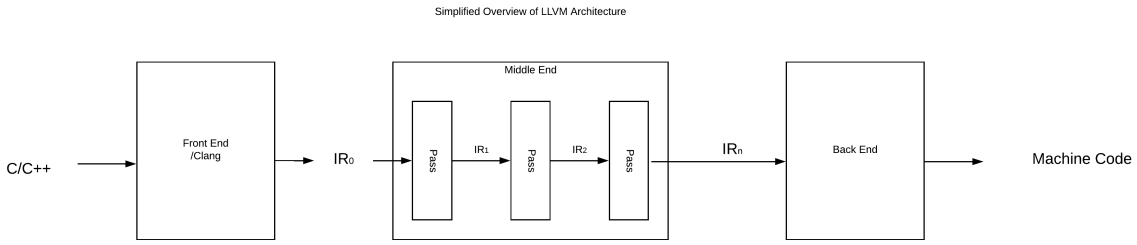


Figure 2.3: An Overview of LLVM Major Components.

Figure 2.3 shows an overview of LLVM's major components. Like most modern compiler architectures, LLVM contains three parts, a front end (Clang), middle-end, and back end. The front end, also known as clang, takes source code and transform it to LLVM's intermediate representation (IR), which is platform-independent. LLVM's middle-end consists of multiple passes that aimed to analyze, optimize, and otherwise transform LLVM IR [66] in a user-defined sequence. In this thesis, our supports for extending cross-ISA SIMD migration for both approaches are done in the middle-end. Lastly, the back end lowers LLVM IR to ISA-specific machine code.

Profile Guided Optimization (PGO) is an advanced optimization approach deployed in LLVM. The goal of this approach is to optimization application by providing the compiler profiled runtime information at compile time to aid code generation [16]. Clang currently

supports two profile modes: sampling profiling and instrumented profiling [77]. Sample profiling has a low runtime profile but also less efficient in building efficient programs. Instrument profiling has a higher runtime profile but is more efficient in optimizing programs. After either profiling stage completes, LLVM transforms profiled data into a usable messaging format, and recompile the application with this additional information, LLVM recompile the code with better optimizations [75]. In our proposed PGO-based approach, our design shared the same concept as the instrumented profiling approach.

2.3 Popcorn Linux

Currently, there does not exist a commodity heterogeneous-ISA chip multiprocessor with cache-coherent shared memory. To approximate such a machine, we connected an Intel Xeon server (x86-64) to a Cavium ThunderX server (ARMv8) via Infiniband. To support cross-ISA execution, the system software must provide two capabilities: i) the ability to migrate threads and ii) the ability to migrate an application's data between the servers. We build on Popcorn Linux [12], an operating system, compiler, and runtime that provides system software support for cross-ISA execution migration.

Popcorn Linux uses a replicated-kernel OS design, where kernels on separate machines communicate via message passing to provide user applications the illusion of a single machine. The OS provides a thread migration service, whereby threads call into the kernel and transparently resume execution on the destination architecture. Underneath, the originating kernel (origin) transfers the thread's context to the destination kernel (remote) to be reinstantiated and returned to user-space. Additionally, the OS provides a page migration service that transfers an application's data between machines on demand. When initially migrated to a new machine, the application has no pages mapped into memory – all mem-

ory accesses result in page faults. The OS's page fault handler is modified to intercept the faulting accesses, allowing the kernels to observe data accessed by threads and coordinate to migrate page data between machines. Pages are unmapped from the origin and mapped into the thread's address space on the remote.

While these capabilities are sufficient for migrating threads and application data pages between machines, they are not sufficient for cross-ISA execution. Popcorn Linux's compiler generates multi-ISA binaries that are suitable for cross-ISA execution. In multi-ISA binaries, the application's virtual address space is aligned so that references to symbols (global data, function addresses) are identical across all architectures. For execution state that is tailored to each ISA's capabilities (stack, register set), the compiler generates metadata describing the function activation layout.

When migrating between architectures, Popcorn Linux's run-time parses the metadata for all function activations currently on the stack and transforms them between ISA-specific formats. After this state transformation, the runtime hands the new stack and register set to the OS's thread migration service to be restarted on the destination architecture. By aligning as much of the virtual address space as possible and only transforming the pieces that are tailored to each ISA, most of the application's state is valid across architectures, incurring minimal thread and data migration time.

In order to generate multi-ISA binaries, Popcorn Linux's compiler builds on LLVM's modularity to generate object files for all target architectures. The compiler's front-ends and middle-ends generate optimized LLVM bitcode from the application source. Threads can only migrate between architectures at equivalence points [117], i.e., points where execution has reached a semantically equivalent location and there exists a valid mapping between each ISA's execution state. A pass inserts call-outs to a migration library at equivalence points to enable thread migration between architectures. Because threads can only migrate at the

inserted call-outs, there is a tradeoff between how many migration points are inserted into the code (and the associated call overheads) versus how long it takes a thread to respond to a migration request. Finally, a pass tags each call site with metadata describing all the live values at that location, as the run-time must be able to recreate the sequence of all function activations in the destination ISA's format.

The LLVM bitcode instrumented with migration points and live value metadata is passed to all target ISA backends for code generation. As the bitcode is lowered to machine code for each ISA, the backend records the locations of the live values specified by the middle-end, i.e., stored in a register or a stack slot. Note that the same set of live values is passed to each backend, and thus the same values are alive at each call site, meaning the runtime only needs to determine where to copy each live value for each ISA. In addition to live value locations, the compiler records per-function information such as callee-saved register locations and frame sizes. Each call site is also tagged with a unique ID to correlate call sites across architectures at run-time. The linker takes the object files for each architecture as input and emits a multi-ISA binary with an aligned virtual address space and the transformation metadata.

At run-time, threads execute like normal threads. When threads reach a migration point, they call out to the migration library to check if migration was requested, and if so, migrate to the requested destination. Threads check for migration requests via syscall – the kernel maintains a per-thread flag that can be set within the application or by external processes. If a migration is requested, the thread takes a snapshot of its current register set and begins transforming its stack and register set. First, the thread unwinds its stack to determine which activations are currently alive and to load each function's metadata. Next, the thread goes frame-by-frame from the most recently called function inwards, copying live values to the correct destination-ISA location. After transformation, the runtime passes the trans-

formed register set for the outermost function to the OS's thread migration service. The kernel transfers the register set to the destination, which instantiates a new thread with the transformed register set and returns the thread to userspace. The thread exits the runtime on the destination and resumes normal execution as if it were still executing on the same machine.

Chapter 3

Related Work

This chapter discusses research in related fields. The problem space for this thesis mainly lies in heterogeneous computing. However, for a related work survey, this thesis classifies heterogeneous computing into three different areas: CPU/GPU Computing, single-ISA heterogeneous computing, and heterogeneous-ISA computing. Besides heterogeneous computing, our work also involves extending support for SIMD instructions in a particular (Popcorn Linux) framework and our purposed scheduling techniques touch techniques in workload scheduling domain as well. Surveys of workload scheduling techniques in this thesis are also classified into three different areas: workload scheduling in heterogeneous-ISA systems, contention-aware scheduling, and NUMA-aware scheduling.

This chapter starts with three research areas in heterogeneous computing: Section 3.1 for CPU/GPU computing, Section 3.2 for single-ISA heterogeneous computing, and Section 3.3 for heterogeneous-ISA computing. Then this chapter describes related work done in the SIMD field in Section 3.4. Finally, this chapter ends up with Section 3.5 discussing related work in the workload scheduling domain.

3.1 CPU/GPU Computing

The vast majority of past efforts in heterogeneous computing focus on CPU/GPU systems [38, 85] in which a GPU accelerator device is attached to a host CPU [69, 89, 90, 102]. One of the major problems for process migration between CPU and GPU is that migration requires writing target-specific code. This step usually requires code modifications from the programmer's end. Much research aimed to solve this automatic offloading to GPU through some checkpointing technique. Gad et al. [38] used static analysis and combined it with runtime heap information gathering to achieve automatic context migration. Karablieh et al. [56] proposed a checkpoint technique using POSIX threads. Xu et al. [124] proposed a checkpoint technique for fault-tolerant CPU/GPU systems. Our work provided process migration through stack rewriting and should be able to extend support for offloading to GPU with minor modifications.

3.2 Single-ISA Heterogeneous Computing

Single-ISA heterogeneous computing uses cores of the same ISA but with different ISA extensions or micro-architecture. Kumar et al. [62] first proposed the single-ISA heterogeneous architecture in 2003 as a solution to reduce processor energy usage. Kumar continued exploring this field by investigating multi-threaded workload [63] and architecture optimization [64] on single-ISA heterogeneous cores. These researches successfully contributed to their commercial adaptations, such as ARM's "big.LITTLE" [39], "DynamIQ" [78], and Nvidia's "Kal-El" [90]. Researchers use this new architecture to explore areas such as resource partitioning [46], program scheduling [22, 26, 113], and in/near memory computing [59].

3.3 Heterogeneous-ISA Computing

More recently, there have been several studies on heterogeneous-ISA systems [12, 68, 72, 114, 115]. Venkat et al. [114] investigate the design space of heterogeneous ISAs using general-purpose processors (e.g., x86, Thumb, Alpha) to evaluate their effectiveness for improving performance and energy. Their work reveals that many applications, especially in the HPC domain, exhibit better performance, and energy efficiency, often in different program phases. Akram further studied the impact of ISAs on processor performance in 2017 [3, 4]. Exploiting processor-preference for performance and energy gains on heterogeneous-ISA systems requires a cross-ISA execution migration infrastructure, which can transform the program state from one ISA format to another and migrate execution to the optimal-ISA core. Barbalace et al. [12] present a complete software stack – Popcorn Linux – that supports a cross-ISA execution migration infrastructure, which we summarize in Chapter 2. These efforts do not seem to consider cross-ISA migration inside SIMD regions – precisely the problem that we study. We extended Barbalace's [12] compiler and run-time for cross-ISA SIMD migration.

Lee et al. [68] investigated the offloading of compute-heavy workloads from mobile platforms, which often use RISC-style ISAs such as ARM, to server platforms that use CISC-style ISAs such as x86. They presented a compiler infrastructure that generates binaries that can execute on multiple ISAs, such as those with uniform memory layouts, address conversion code, endianness translation/conversion code, and a run-time system for orchestrating the offloading of computations across ISA-different platforms. We do not consider offloading tasks because our goal is to maximize throughput, which requires us to utilize both servers as much as possible, rather than having one platform wait for the completion of the offloaded tasks. Offloading and migrating processes are also different in terms of process execution sequence. Offloaded processes require synchronization at the end of the offloaded computa-

tion with their parent processes, and exit at the original platform. In contrast, a migrated process can exit on the migrated architecture since its call stack is transformed.

The paper of Lin et al. [72] is very similar to Barbalace's Popcorn Linux work [12], but focuses on the mobile incoherent domain SoCs, whereas our work focuses on the server space with general-purpose CPUs. Venkat et al. [115] focuses on leveraging a single large superset ISA composed of fully custom ISAs but scopes out cross-ISA migration and lacks a real prototype.

3.4 SIMD

With SIMD gaining more attention from chip vendors [7, 49, 65] as a means to extract additional data parallelism, research interest in this space has also surged. Most efforts focus on redesigning algorithms to leverage SIMD instructions [21, 41, 44, 47, 95], exploring SIMD usage in new application domains [23, 27, 42, 123], and improving SIMD code generation at the compiler level [9, 34]. SIMD instructions have also been considered in dynamic binary translation (DBT) efforts [24, 33, 35, 43, 71, 74, 94], which focus on the efficient translation of SIMD registers between ISAs. In addition to cross-ISA SIMD translations at migration points, our work also conducts SIMD-aware scheduling to maximize system throughput of workloads composed of applications with or without SIMD usages – entirely out of scope for DBT efforts.

3.5 Workload Scheduling

Workload scheduling has been studied extensively under many scenarios. Our work focuses on workload scheduling in heterogeneous systems and, more specifically, heterogeneous-ISA systems. However, our design is also influenced by scheduling techniques in other domains. Thus, in this section, we will discuss the related works in several scheduling disciplines that range from workload scheduling for heterogeneous systems to NUMA-aware scheduling.

3.5.1 Workload Scheduling in heterogeneous systems

Workload scheduling in heterogeneous systems has been consistently receiving attention from the research community. However, most of these efforts focus on single-ISA heterogeneous systems, where the heterogeneity is in terms of execution frequency [79, 97], micro-architecture [60], cache sizes [55], asymmetric interconnect [70], or performance goals [96, 112]. These schedulers follow the same principle of allocating resources to applications based on their resource demand – e.g., if an application requires a higher usage of a resource (frequency, micro-architectural features, cache line), an attempt is made to grant that resource.

Workload Scheduling in heterogeneous-ISA systems has received less attention. Beisel et al. [13] extend the Linux Complete Fair Scheduler (CFS) to support cooperative multitasking for heterogeneous accelerators (CPU/GPU). The scheduler in the paper of Barbalace et al. [12] only balances thread counts across ISA-different cores. More recently, Prodromou et al. [99] present a machine learning-based program performance predictor that drives an ML-based heterogeneous-ISA job scheduler. These works have ignored migration costs as well as processor-preference's impact on system performance. Karaoui et al. [57] present schedulers for heterogeneous-ISA systems, Although these schedulers consider migration costs but ignore SIMD extension.

Our work is one of the few that accounts for actual system migration costs, ISA-different SIMD extension differences, and application's processor-preferences.

When designing our schedulers, we need to not only account for the heterogeneity in the system aspect but also how to efficiently schedule applications within the same system. With this in mind, our design is also influenced by scheduling techniques in other domains, such as contention-aware scheduling and NUMA-aware scheduling. These well-researched fields provide us with more insights on how to improve our processor-preference-aware scheduling decisions.

3.5.2 Contention-aware Scheduling

A contention-aware scheduler [54, 81, 127] focuses on minimizing contention in a system through classifying and co-locating compatible applications in the same memory domain [111]. Jaleel et al. [54] schedule single-thread applications dynamically based on the application impact on each other. Based on their LLC behavior, their design classifies applications into four different categories and applies fixed scheduling policies accordingly. In our work, we also want to identify suitable application pairings if the underlying systems support multithreading. Thus, for our scheduling policy, we classify applications into three different categories. However, our selection metric is based on the application's processor-preference (degree of application execution slowdown between running on different processors).

3.5.3 NUMA-aware Scheduling

A NUMA-aware scheduler focuses on how to schedule applications across NUMA nodes. A system consists of multiple NUMA nodes is becoming more and more important in the scheduling field as multiple NUMA sockets are common in today's servers. Because our work aims for generic heterogeneous-ISA systems, we also need to account for servers that have multiple NUMA-nodes. Tam et al. [107] place threads with the same frequency of sharing and places them in the same socket. Blagodurov et al. [15] group single-thread applications, so that each group has almost equivalent memory usage, and places these newly formed groups into different sockets, and migrate pages along with their threads. Lepers et al. [70] schedule applications based on maximizing bandwidth for communicating threads.

Chapter 4

Enabling Cross-ISA SIMD Migration

This chapter is the first of three design chapters of this thesis. Chapter 4 clarifies the meaning of several frequently used SIMD-related terms used in this thesis. In this chapter, a basic approach to extending SIMD support for the Popcorn Linux Framework is also provided.

Chapter 4 begins in Section 4.1 with an introduction of several SIMD-related terms that will be used throughout this thesis. Section 4.2 talks about the decision making behind inserting migration points inside each SIMD region and Section 4.3 explains how each SIMD region migration correctness is ensured.

4.1 Definitions

In this section, the meaning of *SIMD region*, *SIMD workload*, and *SIMD-intensive* are defined.

We define a *SIMD region* as a piece of code within a program that includes the usage of SIMD instructions, such as a vectorized matrix computation. Figure 4.1 is an example of a SIMD region under the LLVM intermediate representation (IR) level. A SIMD region's size can vary in execution time, and a program can contain any number of SIMD regions. If multiple SIMD regions are nested together, these regions are considered as a single SIMD region. SIMD instructions show high processor-preference toward cores with ISAs that provide wider SIMD register width support (See Section 3.4 for detail). To demonstrate the performance

benefit of leveraging processor-preference in heterogeneous-ISA systems, this thesis develops a framework that has cross-ISA migration within SIMD regions.

A *SIMD workload* is defined as a set of applications in which every application has at least one SIMD region. In contrast, a non-SIMD workload is a set of applications that have no SIMD regions. Because SIMD regions experience higher processor-preference toward ISAs with wider SIMD register width support, this thesis uses SIMD workloads to investigate the impact of high processor-preference applications on the execution of co-executing workloads.

We define a program to be *SIMD-intensive* if 50% of program execution time is in SIMD regions. Investigating migrations in SIMD-intensive programs allows us to optimize our SIMD extension support for the Popcorn Linux Framework. Our Profile Guided Optimization-based (PGO) approach (see Chapter 5 for detail) supports SIMD region migrations in the Popcorn Linux Framework. Applications instrumented using the PGO approach respond to migration commands efficiently and incur low runtime overheads.

4.2 Selecting Migration Points

The main obstacle for enabling SIMD region migration was identifying a suitable location inside each SIMD region for migration. The LLVM's intermediate representation (IR) provides a means to overcome this. The intermediate representation is one of the LLVM's unique features. LLVM IR is both easy to read and platform-independent. Upon inspection of multiple SIMD IRs generated by the LLVM compiler [67], we observed that SIMD computation at the LLVM IR level followed a very predictable code flow, as shown in Figure 4.1.

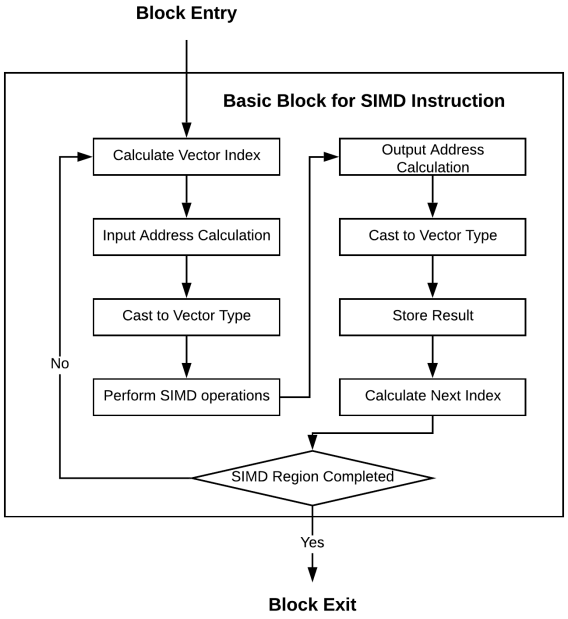


Figure 4.1: LLVM Basic Block Layout for SIMD instructions.

The majority of SIMD computations occurred within a single basic block. In cases where a SIMD region spans multiple basic blocks, at least one of those basic blocks contained this code flow. In Popcorn Linux [12], equivalence points [117] are identified as "migration points" – i.e., program points where execution can be migrated across ISAs. Function boundaries are one of the naturally occurring equivalence points. During compilation, migration points are automatically inserted at function entries and exits [12]. Our design aims to enable migration within a SIMD region with minimum modifications to the underlying framework. Thus, this approach is extended to our design. We created a new LLVM Pass that inserted during the LLVM middle-end stage. This pass was designed to execute before the Popcorn Linux modifications in LLVM took place. In this pass, we swept through the generated IRs and identified all basic blocks containing SIMD instruction code flow, as shown in Figure 4.1. New equivalence points were provided through adding new function boundaries inside a SIMD region via "dummy" function call. This approach reduced the need to find non-function-

boundary equivalence points inside each SIMD region manually. We created a simple library that provided the dummy function call and added the new library to the link-time libraries.

With the migration instrumentation method set, one crucial question remained unanswered: where should the dummy function call be placed inside a SIMD region? In theory, the Popcorn Linux Framework supports the arbitrary placement of this dummy function call. As long as the framework encountered a function boundary (newly inserted dummy function) inside a SIMD region, a migration point can be instrumented. Unfortunately, after attempting to place one function call after each instruction inside a SIMD region, we found that arbitrarily selecting a dummy function placement location is not feasible as we encountered multiple compilation errors.

Further investigations into the Popcorn Linux framework's source code revealed why the SIMD migration was not initially supported in the original Popcorn Linux Framework. To ensure a correct migration, Popcorn Linux Framework collects live values at each migration call site using the LLVM stackmap feature. Live values are programmer-defined variables, or LLVM generated intermediates at statically-known locations within a code. In essence, live values are just values that runtime requires to be live at a migration point. Popcorn Linux Framework stores these live values before migration allows a program to resume execution when migrated to another ISA successfully. Then the framework generated a single set of optimized LLVM bit code and lowered it through target-specific back ends. Because SIMD extensions are architecture-specific features with different SIMD register width supports, it is difficult to find a single legal live value argument across different ISAs and store it in a single stackmap (512 bits live value on x86 requires 4 128 bits live values on ARM). Attempts were made to try to split large SIMD operands for x86 ISAs in each stackmap into ARM-compatible ones. However, our attempts were unsuccessful as we ran into multiple compilation or assertion errors. After consulting with the LLVM community, we found that

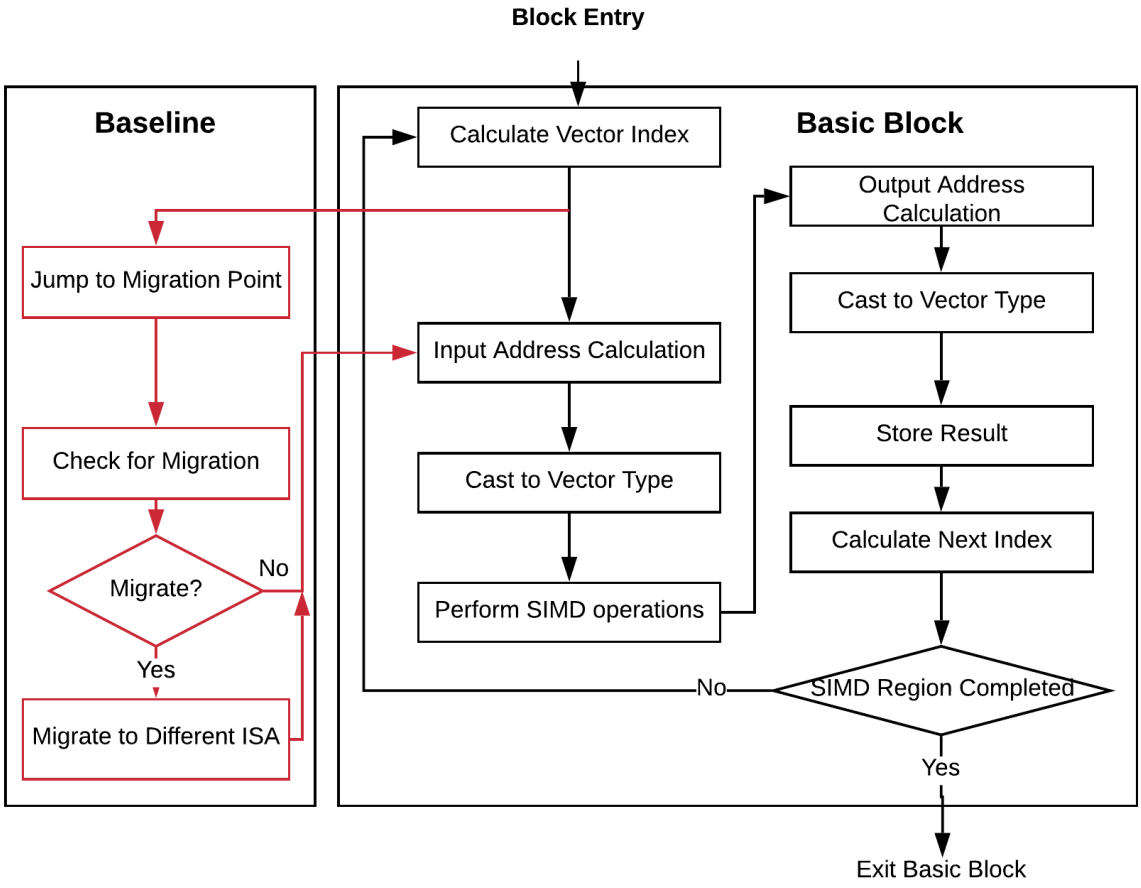


Figure 4.2: Baseline Approach for Supporting Cross-ISA SIMD Migration.

the current LLVM compiler has no support in performing vector splitting for stackmap operands, and community experts strongly suggest us to avoid solving this vector operands splitting during legalization stage while lowering IR. Stackmaps in LLVM are designed to describe the locations of elements such that they can be easily found at run time. Stackmaps are therefore designed to expect values stored in stackmap to be always legal arguments. Therefore, we directed our attention to find the right instrumentation point that avoids the presence of SIMD live values.

After inspecting the basic block code flow several times, the design decision was to place the dummy function after the vector index is calculated. This approach both reduces the number of live values need to be recorded by stackmap compared to other suitable insertion location ("Input Address Calculation" and "Calculate Next Index"), as well as prevents the hassle of dealing with legalizing SIMD live values across different ISAs explained in the prior paragraph. Because our design considers nested SIMD regions as a single SIMD region, we did not encounter any SIMD live values outside SIMD basic block that would require further modifications. Inserting after the vector index eliminates SIMD live value presence inside the region as SIMD registers are not extracted until "Cast to Vector Type" is performed. Figure 4.2 illustrates this approach. For reference purposes, this approach is called the "Baseline Approach".

4.3 Vector Unrolling

After inserting migration points, we must ensure that the program executes correctly after (potential) migration. Because each ISA implements SIMD operations with varying widths, the number of loop iterations needed for each SIMD region varies. Our extension for the Popcorn Linux Framework needs to account for combining different numbers of loop iterations for x86 ISA and ARM ISA due to their different SIMD register width support.

Consider the example in Figure 4.3, in which one SIMD region is vectorized for ISA A and one for ISA B, whose SIMD register width vary by a factor of two. Suppose there are 1000 elements needed to be computed; thus, ISA A will take half as many iterations (250) as ISA B (500) to complete the task. Corner cases can violate program correctness if the Popcorn Linux Framework tries to migrate from ISA B to ISA A at the start of the last iteration (iteration 499) on ISA B and exit on ISA A. In this case, the result will perform

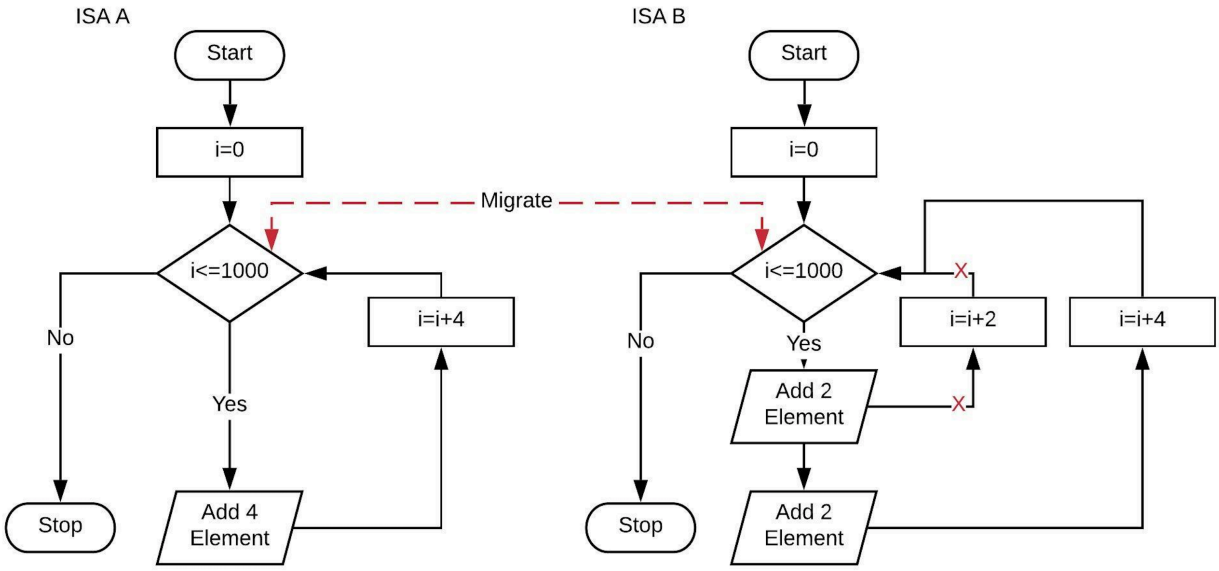


Figure 4.3: Vector Unroll Example.

two unnecessary calculations at the end (ISA A will attempt to perform a calculation on two additional elements of 1001 and 1002 or unknown program behaviors if element 1001 and 1002 does not exist).

One way to prevent this error is by unrolling the loop as many times as the least common multiple (LCM) of both ISAs' SIMD register width so that the same number of calculations are done in a single loop iteration. In LLVM, the loop vectorizer uses a cost model to decide on the unroll factor, and users can force the vectorizer to use specific values [76]. The LCM approach is compatible with any compiler optimization. The LCM is generated based on the final number of elements that are being processed in each iteration after compiler optimization (a SIMD register width of 2 unrolled three times is equivalent to a SIMD register width of 6). Hence, for Figure 4.3's example, the problem is solved by unrolling twice on ISA B. Therefore, each time ISA B performs operations on four elements, it is identical to the number of elements performed in a single loop iteration for ISA A.

As it turns out, this technique also helps to reduce the runtime overhead (discussed in detail in Chapter 5) by increasing the migration check interval.

In order to ensure programs were migrated correctly if migrations were initiated in the SIMD region, a series of micro-benchmarks were used. We designed several micro-benchmarks contained mainly of matrix computations (vector add, vector multiply, vector add and multiply , etc.). At the end of each test, we checked if each migrated program had the same result as the non-migrated version. We used the same random seed to ensure the element of the matrix was the same across each run , and we tested multiple times. For all our runs, our migrated micro-benchmarks return the same value as the non-migrated ones. Thus, the correctness of migration within a SIMD region is guaranteed in our baseline approach design.

Chapter 5

Optimizing Cross-ISA SIMD Migration

This chapter is the second of three design chapters for this thesis. In this chapter, we identify the weakness of the baseline approach discussed in Chapter 4. We refine our design with a new profile guided optimization (PGO) based approach similar to LLVM's built-in PGO. Using this new approach, we can lower runtime overhead while ensuring applications are still responsive to migration commands.

Chapter 5 begins with Section 5.1 identifying the weakness of the previous baseline approach, and ends with Section 5.2 describing the new profiled guided optimization-based approach.

5.1 Baseline Approach Overhead

SIMD operations are meant to speed up computation, and in most cases, each SIMD loop iteration executes relatively quickly. However, if the compiler blindly inserts migration points at the beginning of each SIMD loop iteration (after "Calculate Vector Index") as discussed in Chapter 4, the program will suffer significant runtime overhead. Figure 5.1 illustrates the cause of this extra runtime overhead. This overhead is mainly due to the

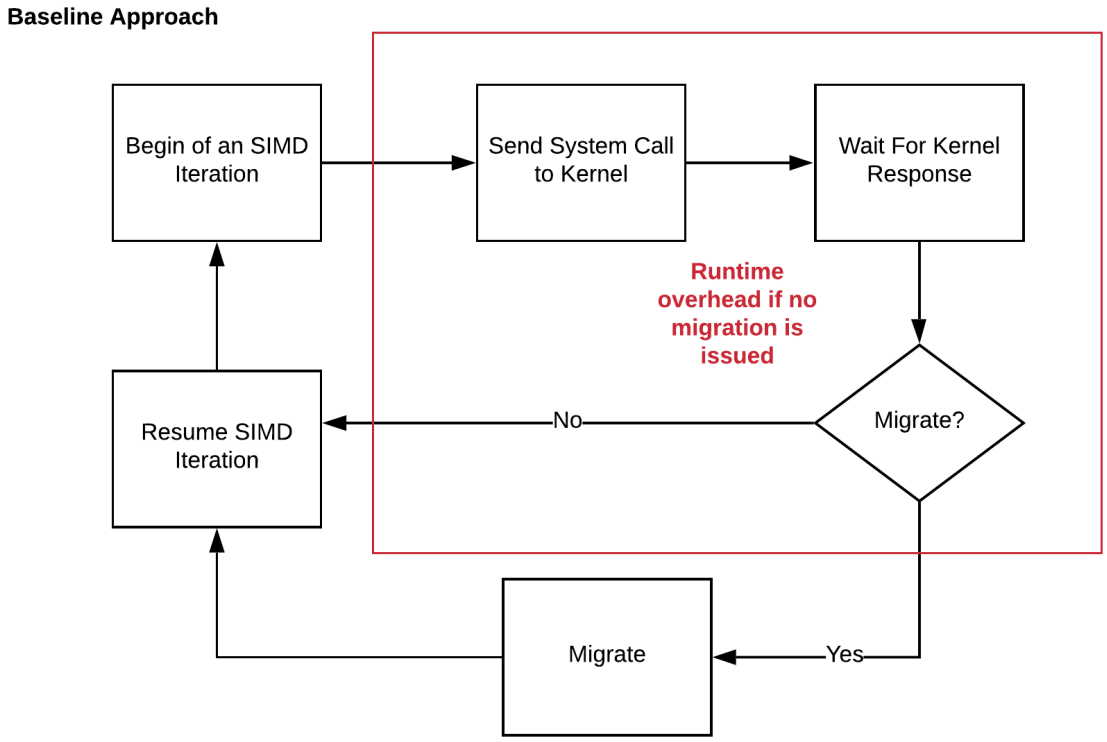


Figure 5.1: Baseline Approach Overhead.

additional system calls to check for a migration decision (i.e., whether or not to migrate). For SIMD-intensive programs with a large number of loop iterations that never actually migrate, naïvely executing system calls to check at every loop iteration can harm performance. Nonetheless, applications should be able to respond to migration requests to efficiently leverage heterogeneous-ISA systems quickly.

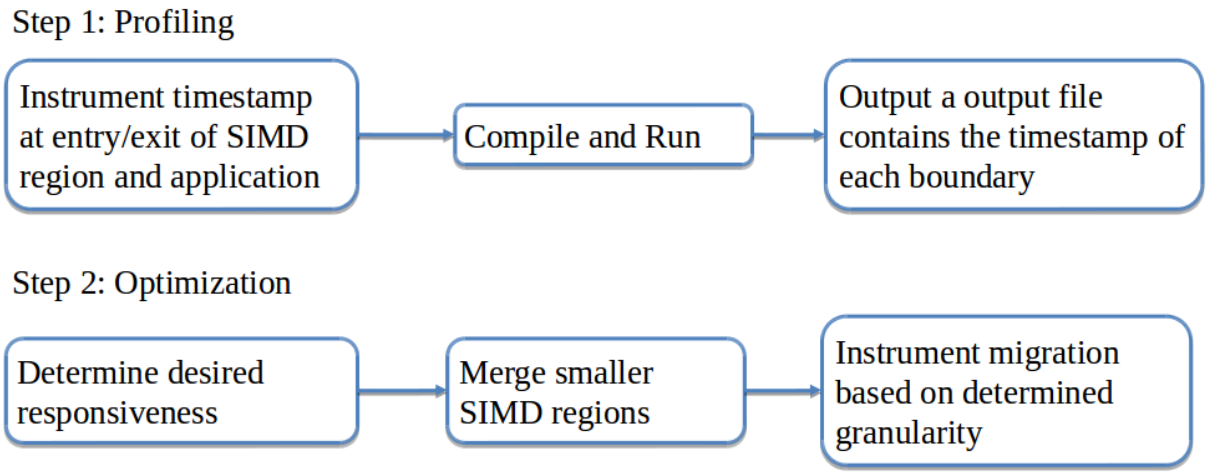


Figure 5.2: PGO Approach for Supporting Cross-ISA SIMD Migration.

5.2 Profiled Guided Optimization Approach

In order for programs to quickly respond to migration requests as well as incur low run-time overhead when migrated, we propose a two-stage profile-guided optimization(PGO) approach, similar to LLVM's built-in PGO [16, 75, 77], to guide the insertion of optimized migration points. We built this approach on top of the existing baseline approach. Figure 5.2 shows an overview of our new approach. The newly proposed PGO approach has two stages: a profiling stage and an optimizing stage.

5.2.1 Profiling Stage

In the profiling stage, we want to compile a program with instrumentation. The instrumented program executes and prints out timestamps at the entry/exit of each SIMD region. These timestamps later will be used to calculate each SIMD region's execution time. Inserting timestamp function calls can be done using a similar approach as inserting migration points discussed in Chapter 4. However, timestamp information itself does not provide enough

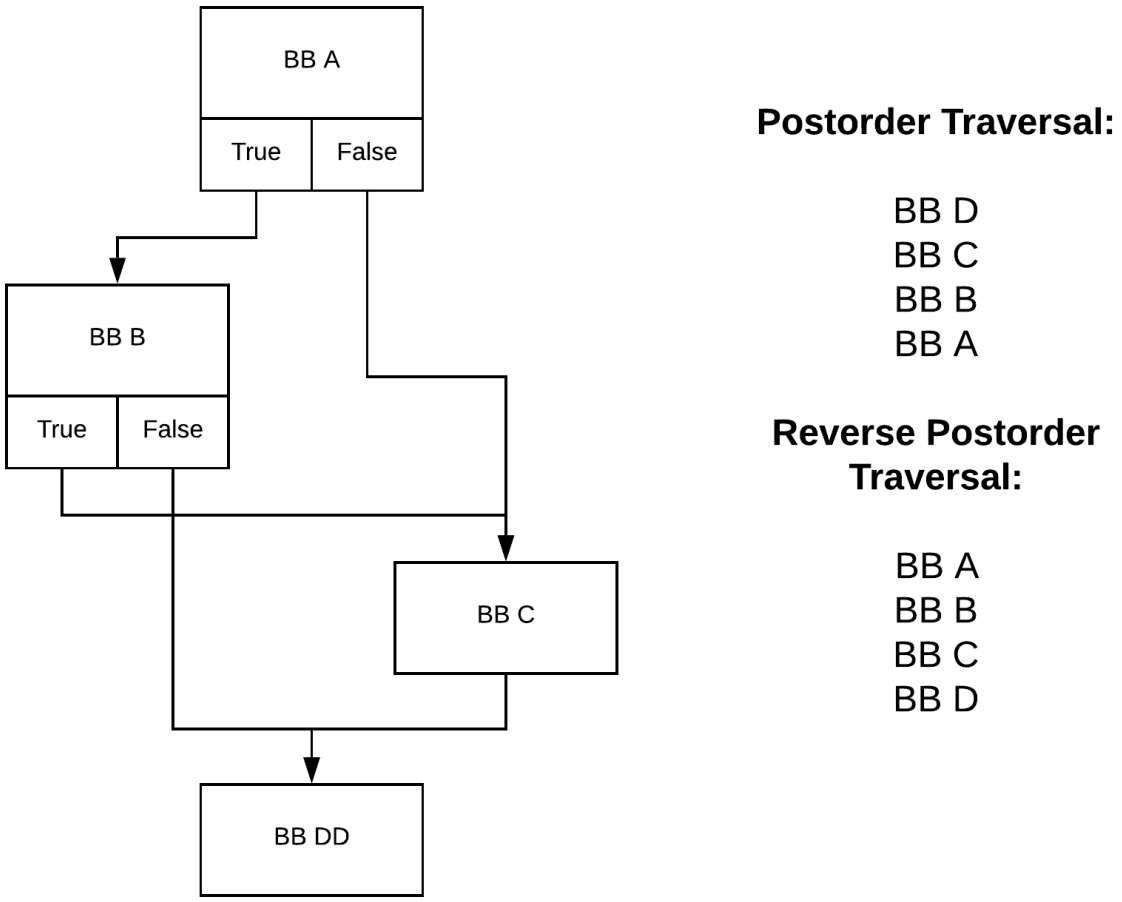


Figure 5.3: Reverse Post-order Traversal Example [14].

information for a program to make intelligent optimization decisions. To supply the compiler with more information, we also need to print out more information so that each timestamp can map to a specific SIMD region as well as can identify whether a particular timestamp corresponds to entry or exit point of a SIMD region.

Many techniques have been used by the compiler to uniquely identify a basic block, such as name mangling for a unique name [28], numbering, etc. Among the numbering techniques, we choose the reverse post-order (RPO) over the control flow graph (CFG) to assign a unique number to each basic block, even though there are other methods we can use, such as pre-

order traversal and in-order traversal. The benefit of using reverse post-order is to leverage the existing infrastructure in the LLVM [67] because the LLVM itself highly relies on the reverse post-order to run transformations and analysis [28]. Figure 5.3 shows that reverse post-order traversal is just the reverse of post-order traversal. RPO guarantees that a basic block appears before any of its successors and reaches each basic block in the same order if the program is compiled with the same optimization flags. This guarantee allows us to use profiled data to optimize designated basic blocks.

In our final modified instrumentation for the profiling stage, we created an LLVM IR pass that contains a counter that increases every time we reach a new basic block. If we encountered the entry/exit point of a SIMD region, we would insert a timestamp function call that contains two additional pieces of information: Basic Block RPO ID (counter) and a flag to indicate whether it is an entry or exit of a SIMD region. This inserted function call is linked to an external library that prints out timestamp as well as two additional pieces of information to the profiled data file.

In the end, the instrumented program is executed with outputs logged into a text file, which ends the profiling stage.

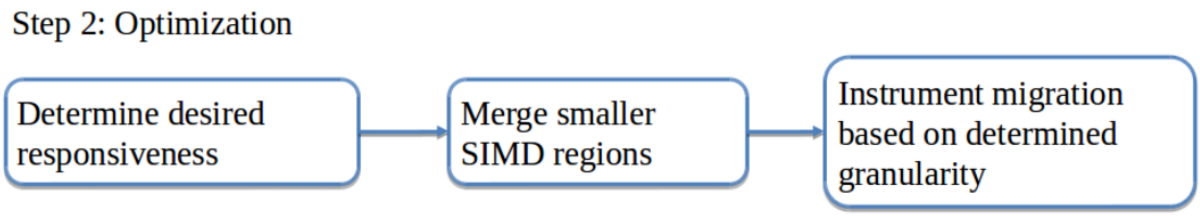


Figure 5.4: PGO Optimizing Stage.

5.2.2 Optimizing Stage

The optimizing stage, shown in Figure 5.4, contains three steps. After a profiled data file is generated, we first need to use the generated data to determine how often migration should be checked inside each SIMD region. The granularity at which each application checks for migration decisions can be modified through executing migration checks only after certain iteration intervals. System designers using this framework can choose an arbitrary value as granularity with an upper bound of checking at every interval (Baseline Approach) or not checking at all (vanilla). System designers must know that migration responsiveness and runtime overhead are directly correlated in our framework; increasing responsiveness will result in more runtime overheads. In our design, we assume that the granularity is set at checking for migration at a one-second interval in a heterogeneous-ISA system. Based on our preliminary testing and our workload evaluation results (shown in Chapter 7), this granularity is both responsive and incurs low runtime overheads.

With migration check granularity set and the fact that granularity can be adjusted by checking for migration-only after certain SIMD iteration intervals, the PGO approach needs to determine how many SIMD iterations migration checks be skipped to match this desired granularity. The number of SIMD iterations can be challenging to estimate and automate due to different compiling options and system configurations. To produce an accurate estimation for an application, we created a microbenchmark that contains the mainly same type

of SIMD instructions and the same configuration as the compiling application and adjusts the loop iteration in the source code until the microbenchmark has an execution time of 1 min. We then calculated the number of SIMD iterations the microbenchmark calculated to approximate the number of iterations the compiling server can execute in 1 sec through fundamental mathematics division. Currently, this estimation process has to be done once per application. For our experimental evaluations, we individually profiled each benchmark containing SIMD regions. However, it is possible to use a single metric to estimate target iterations based on the unrolling factor and SIMD extension given at the command line. For example, if a program with AVX-512 extension that unrolled once can perform four SIMD loop iterations in a second, then it is likely another program with the same SIMD extension can only perform two SIMD loop iterations when unrolled twice.

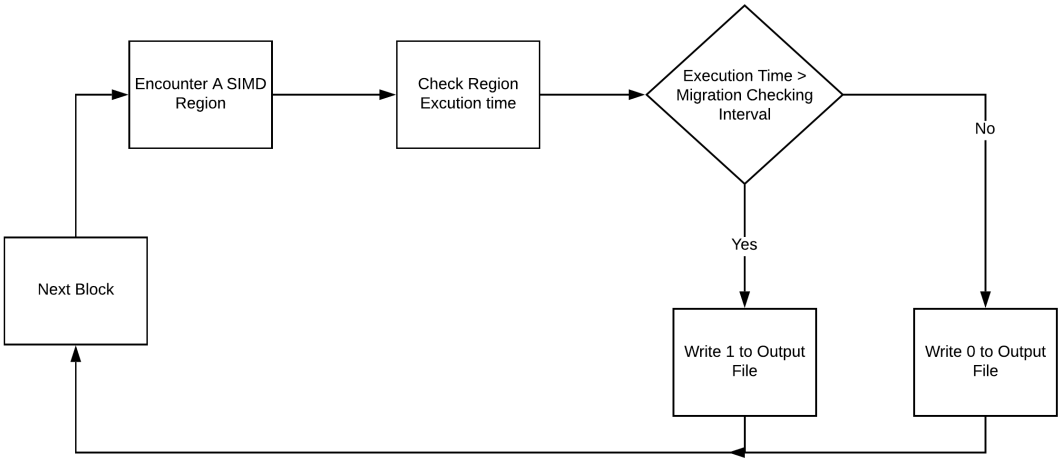


Figure 5.5: Optimization Program Decision Tree.

The second step involves merging smaller SIMD regions. Smaller SIMD regions hurt the effectiveness of the heterogeneous-ISA systems. A program with smaller SIMD regions likely exhibit less processor-preference than larger SIMD regions and incorrectly categorize them as SIMD-intensive program can hinder system performance by not utilizing resources efficiently. PGO approach eliminates smaller SIMD regions in two smaller steps. We created

an independent analysis program that parses through the profiled data and determines the execution time of each region. This independent analysis program also requires the user to provide the desired responsiveness they have determined in a unit of seconds. Based on responsiveness input, the program will output a file with a flag corresponding to each SIMD region. This flag determines whether this SIMD region is considered big enough. A flag that is not set means the compiler will not instrument that SIMD region with the Popcorn Linux Framework instrumentation required for migration, essentially eliminating smaller SIMD regions. The output file will produce a result per line containing the RPO ID, as well as the flag. Figure 5.5 illustrates the parsing program's decision tree.

Up to this point, this framework has gathered information on migration interval checks as well as identified smaller SIMD regions. In the last step, a migration point has to be instrumented in selected SIMD regions based on this new information. We first need to load this information into the LLVM infrastructure so that they can be used later by LLVM IRs.

We propose to have the loaded file available to all functions and stores in a single compilation unit (aka, Module in LLVM). This external profile information is loaded in the initialization stage of the LLVM execution stance rather than loading it every time for each function, which can significantly reduce the run time for parsing external profile. We provide an instance of the class, named ProfileDataManager, to be initialized with the passed in profile when an LLVM execution environment is going to be created. The class ProfileDataManager contains an internal data structure for the query from instrument pass to check whether a basic block should be considered as an instrument candidate. The internal data structure of class ProfileDataManager is organized as a Boolean array. Implementation of the Boolean array allows LLVM to quickly determine if the basic block should be treated by indexing the specified array slot with the basic block number.

In the middle-end, we modified on top of the existing instrumentation pass used in the

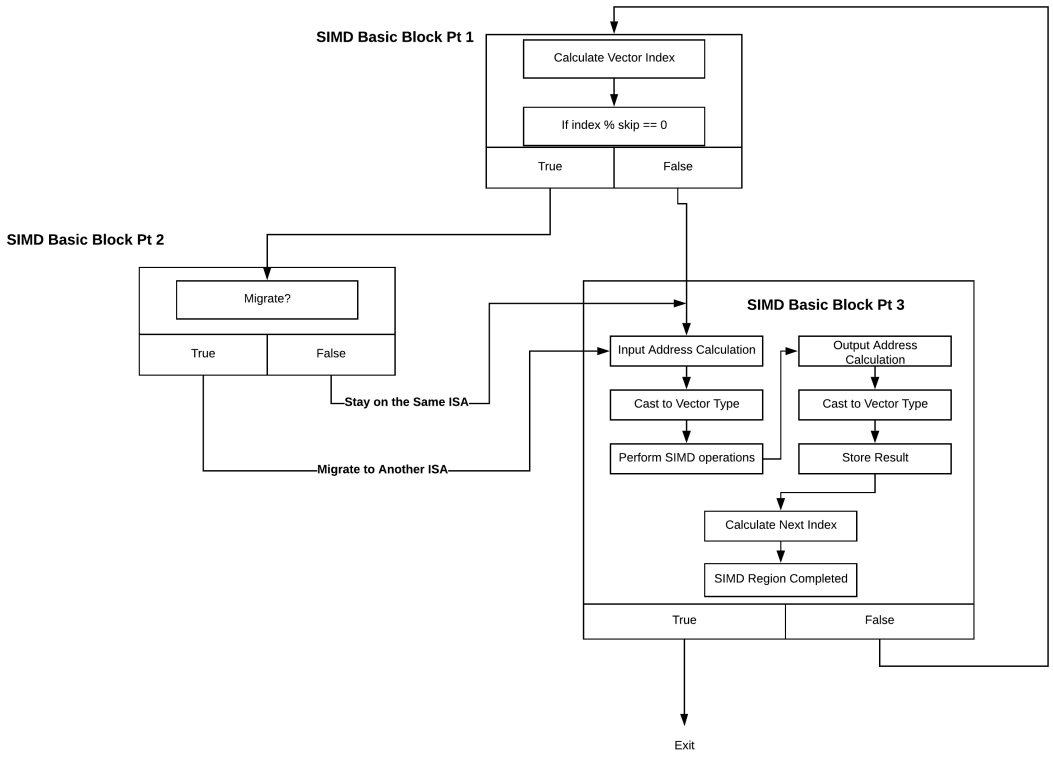


Figure 5.6: SIMD Basic Block Split for PGO Approach.

Baseline Approach (Chapter 4). Like the profiling stage code, we also have a counter that increases when it reaches a new basic block. Now for every SIMD basic block, we encountered while traversing in reversed post-order, we check if the global data for that particular ID, if that basic block ID has flag not set in the corresponding slot in the Boolean array, we move onto scanning the next basic block. If the basic block has a flag set, we split the original SIMD basic block into three smaller basic blocks shown in Figure 5.6, one extracts vector indexes, one checks for migration, and one contains the remainder of the original SIMD basic block. After each split, we skip the next two newly created basic blocks as they are not present in the profiling stage. Figure 5.7 illustrates the final instrumentation of a large SIMD region at the IR level.

Our evaluation revealed that by using the PGO approach, the final instrumented SIMD

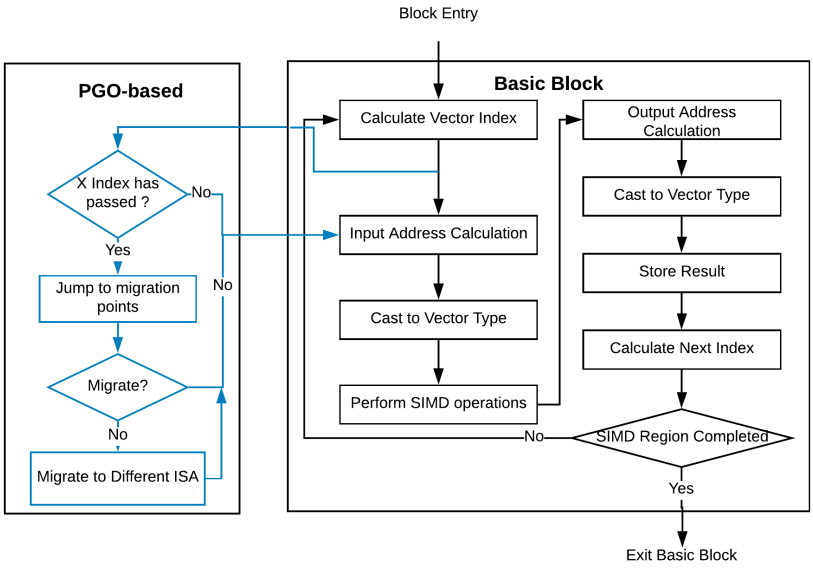


Figure 5.7: PGO Approach for Supporting Cross-ISA SIMD Migration.

application with a one-second migration checking interval would only suffer a 5% execution time overhead on average.

Chapter 6

Leverage Processor-preference for Performance Gain

This chapter is the last of three design chapters for this thesis. Because processor choices in heterogeneous-ISA systems are diverse, applications are more likely to experience different degrees of processor-preference. In this chapter, we leverage applications of different processor-preferences for increased system performance in migration-capable heterogeneous-ISA systems with SIMD extension support. This chapter also explains how our processor-preference-aware scheduler is built and illustrates the scheduler decision-making process.

Chapter 6 begins in Section 6.1, with an explanation of the core idea of our scheduling policy. This chapter ends in Section 6.2 with an illustration of the processor-preference-aware scheduler setup and its decision-making process.

6.1 Scheduling Policy

Compiler-level infrastructural supports for cross-ISA execution migration within SIMD regions allow us to explore further the possibilities of leveraging processor-preference for a greater range of applications [114] to increase system throughput. The system, however, requires a processor-preference-aware scheduler – i.e., one that can decide when to migrate an application from one processor to another based on the application's processor-preference

to increase the overall system throughput.

We propose a scheduling policy to achieve this goal. Our policy assumes that the final system slowdown (taking into consideration all factors, such as clock speed and micro-architectural differences) for each application on each different processor-based platform is known through the profiling stage in our two-step PGO approach. Our policy is centered around one central idea:

the speedup gained from executing an application on the optimal core should outweigh the slowdown other applications suffer from not running on that core. In other words,

$$\frac{speedup\_of\_app\_X}{slowdown\_of\_app\_Y} > 1$$

For example, consider two applications A and B, where A runs $10X$ slower on an ARM-based core than on an x86-based core, and B runs $5X$ slower on an ARM-based core than on an x86-based core. Since the speedup gained from running A on the optimal x86-based core ($10X$) is greater than the slowdown B experiences from running on the non-optimal ARM-based core ($5X$), it is likely effective to schedule A on the x86-based core and B on the ARM-based core (assuming there is only one ARM-based core and one x86-based core available) to improve the system throughput.

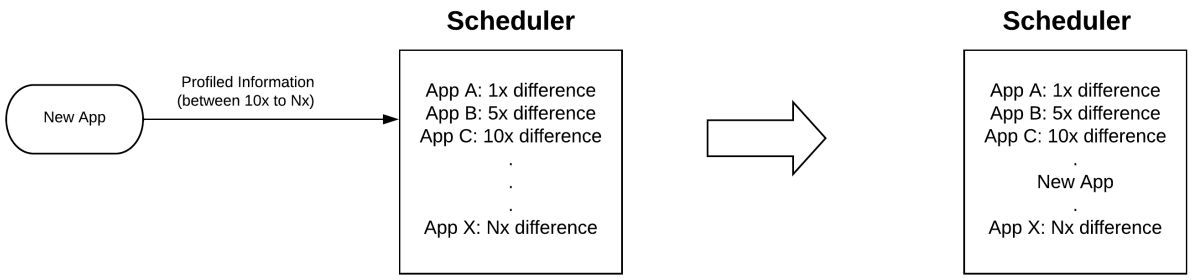


Figure 6.1: Comparing Application by Application is Inefficient.

Our scheduling policy should also be fast in decision making. Any scheduling decision that takes too long to decide can force systems to utilize their resources inefficiently. Figure 6.1 shows that comparing every application-based design on its slowdown is accurate; however, a general linked list style comparison has a time complexity O(Number of Unique Applications). This application-tracking method can be troublesome for a system running diverse workloads for a very long time.

The scheduler categorizes applications into smaller groups using some logically defined metrics to reduce the complexity of comparing every application based on their processor-preferences. In this thesis, we categorize all applications into three smaller groups based on their processor-preference and the hardware thread count (ht) difference between the two ISA-different servers, as follows:

$$App = \begin{cases} Group\_High\_Processor\_Preference, & \text{if } Slowdown \geq \Delta 2ht \\ Group\_Medium\_Processor\_Preference, & \text{if } \Delta ht < Slowdown < \Delta 2ht \\ Group\_Low\_Processor\_Preference, & \text{if } Slowdown \leq \Delta ht \end{cases}$$

In the above equation, Δht represents the hardware thread count differences between the two ISA-different servers.

Our ideology behind using hardware thread count difference as a classification metric is that application processor-preference's impact on system throughput can be directly compensated by executing more programs in parallel in a perfect system-resource-abundant world. Thus, these two factors are closely related. However, system designers can freely choose any other classification metric to form any amount of smaller groups as long as they can group applications with the relatively same degree of processor-preference applications together. Wrong group classification can harm system performance. By classifying all applications into three

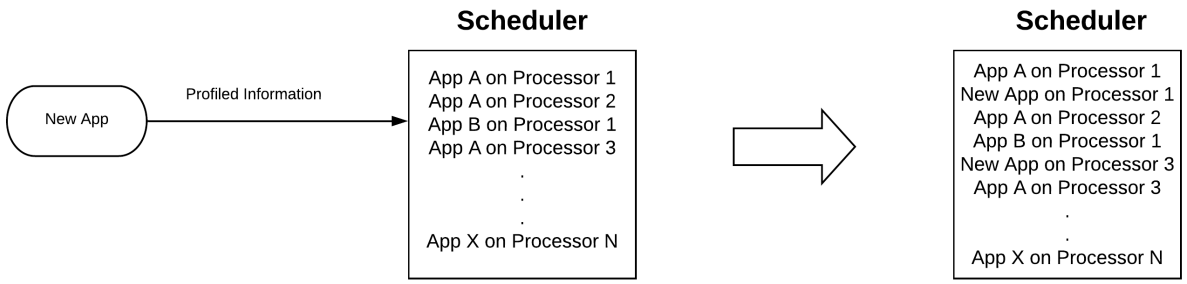


Figure 6.2: Scheduler Information Storing Example in a Multi-processor (>=3 Processors) Scenario.

groups, the system has to compare new incoming applications three times, which have a time complexity of O(3) ≈ O(1).

This scheduling model is further scalable for future multi-processor systems (i.e., more than two processors). System designers in those cases need to identify a "baseline" platform to compare. Applications performance on different processors will be compared to the base platform and will be stored individually in the scheduler, as shown in Figure 6.2.

In our two ISA setup, applications belong to the low processor-preference group (*Group_Low_Processor_Preference*) have small performance differences to either processor and are most likely to benefit from the extra cores. Medium processor-preference group members (*Group_Medium_Processor_Preference*) have a higher preference for the faster processor. However, the slowdown by running on non-optimal cores is likely to equal to or close to the hardware thread count differences between the two servers. Therefore, they are likely to experience a smaller degree of throughput degradation. Lastly, high processor-preference applications (*Group_High_Processor_Preference*) have an extremely high preference for the faster processor, and thus, the speedup gain of running on the optimal cores easily outweighs the maximum hardware thread count differences.

In short, our scheduling policy prioritizes executing high processor-preference group members on the servers with a faster processor as often as possible, followed by medium processor-preference group members, and finally, the low processor-preference group members.

6.2 Scheduler Setup

Our scheduler is implemented using an event-driven client-server model using internet sockets. Each application communicates with the scheduler using function calls upon three events: (1) arrival into the system queue, (2) upon application completion, and (3) after a migration is completed. For the first and second event, function insertions were done at LLVM middle-end using the LLVM pass feature. For the third event, modifications were made in the Popcorn Linux migration library.

Figure 6.3 illustrates the decision tree of our scheduler. The scheduler is idle by default and continues waiting for the arrival of one of three possible events (a program just started, a program just migrated, and a program just finished). Then the scheduler extracts the server usage for both servers it internally tracks. Based on the server usage information, the scheduler uses the scheduling policy discussed in Section 6.1 to make a scheduling decision on the incoming event. For events that are not worth migrating (program already running on an optimal core, no optimal cores are available, or program exits), the scheduler updates its server usage info and continue waiting for the next event. For every migration-worthy event, the scheduler can issue up to two migration commands. If the migration destination server is underutilized, the scheduler issues one migration command to the incoming program. If the migration destination server is fully utilized, one additional migration commands will be issued to migrate away from one of the running programs. Programs that are selected to vacate cores for the incoming events are always selected based on their arrival time. Thus,

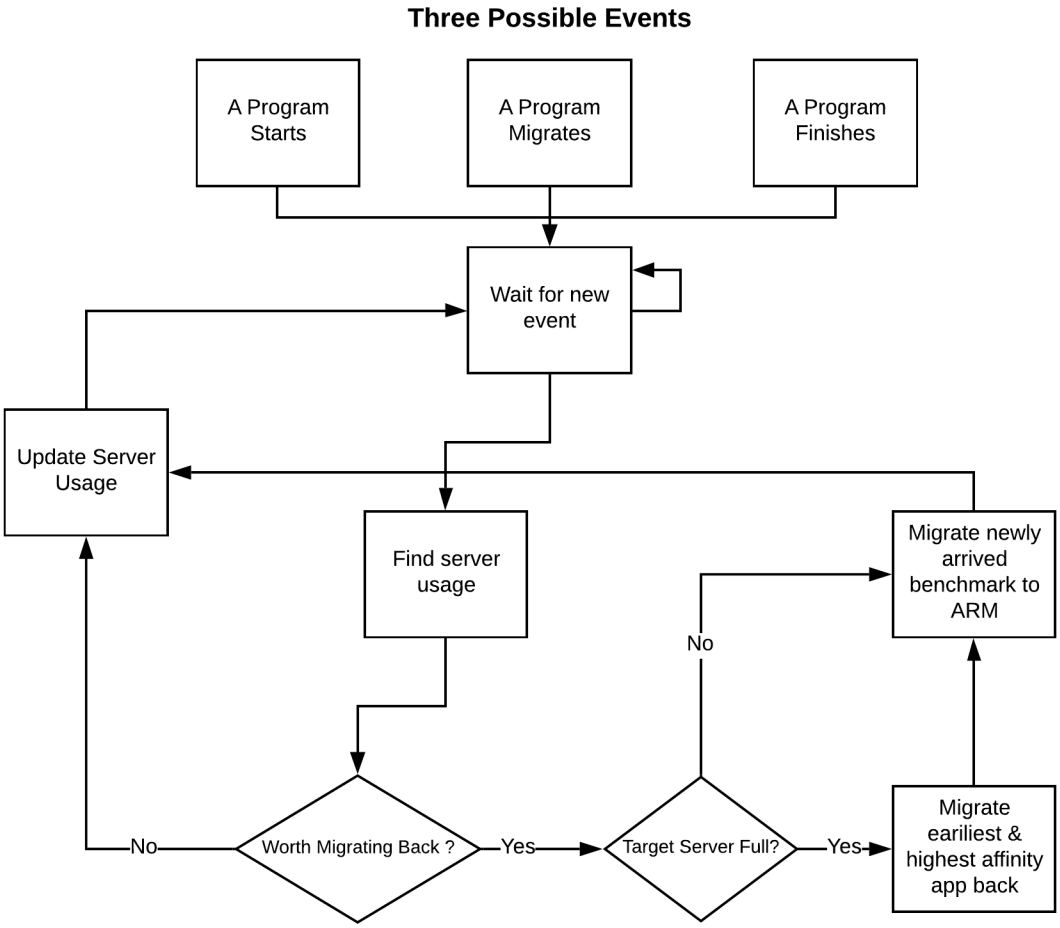


Figure 6.3: Scheduler Decision Making Process.

programs that started recently will be selected first among applications belonging to the same classification group.

The scheduler runs on one of the server cores (Cavium ThunderX in our experimental setup) and always tries to schedule applications without overloading any servers.

Chapter 7

Experimental Setup & Experiment Results

This chapter presents how we evaluated our purposed design. Chapter 7 contains relevant information on system setups, compilation procedures, and testing script descriptions. Impacts of processor-preference on system performance (our design in particular) are first examined in a controlled environment. The effectiveness of our design is further tested in more real diverse workloads at the end of this chapter.

This chapter starts with information on the experimental setup in Section 7.1, followed by Section 7.2 on explaining experiment results. More specifically, Subsection 7.2.1 examines two-application workload scenario results, and Subsection 7.2.2 examines a more real multi-application workload scenario results .

7.1 Experimental Setup

Table 7.1 describes the key characteristics of the heterogeneous-ISA servers that we used in our experiments. We considered four server configurations for comparison: (1) a homogeneous system composed of two Xeon servers, called “x86-static”; (2) a homogeneous system composed of two Cavium ThunderX servers, called “ARM-static”; (3) a heterogeneous system composed of one Cavium ThunderX server and one Intel Xeon Gold 5118 server, called

Table 7.1: Server Configurations.

Name	Intel Xeon	Cavium ThunderX
Generation	Gold 5118	1
ISA	x86-64	ARMv8
Micro-architecture	OoO	IO
Number of Cores	12	96
Number of Threads	24	96
RAM Size	48 GB	128 GB
SIMD Register Width	512 bit	128 bit

"het-static", wherein applications are statically pinned to the next available core with the highest processor-preference and run to completion on that core (i.e., no migration); and (4) a heterogeneous system composed of one Cavium ThunderX server and one Intel Xeon Gold 5118 server with cross-ISA SIMD migration enabled using the aforementioned techniques, called "het-dynamic". We hope to show the potential of systems with heterogeneous-ISA multiprocessors through emulating such systems using servers with different ISA-based processors combined with the Popcorn Linux Framework. As shown in Table 7.2, all four setups have identical processor costs; thus, performance is the only evaluation metric to show if the design is profitable. x86-static, ARM-static, and het-static serve as a baseline to evaluate

Server configuration	Cost
x86-static	$2546
arm-static	$2432
het-static | het-dynamic	$2489

Table 7.2: System Configuration Cost.

the effectiveness of het-dynamic setup and do not migrate applications between servers. For het-dynamic, the two servers are interconnected via RDMA over Infiniband (56Gbps). We used factory-specified settings for all setups because we are trying to evaluate the maximum performance of each machine despite their micro/macro-architectural differences and are not interested in trying to isolate individual differences.

We used the same benchmarks as in Figure 1.4, compiled in two different ways for our experimental studies. For the het-dynamic setup, the benchmarks are compiled with the migration instrumentation described in Chapter 5. For the three baseline configurations, the benchmarks are compiled without any instrumentation to avoid unnecessary overhead and to ensure a fair comparison (applications do not migrate in these cases). Currently, our PGO approach is done manually. We profiled the SIMD benchmarks and instrumented the SIMD regions to check for migration once every second. Because each benchmark was profiled beforehand, the relative slowdown of all applications is known upfront when setting up classification groups for schedulers.

Our evaluation workload script is similar to the testing script used in Section 1.2. The evaluation workloads were generated by a script that starts a workload batch with a predefined benchmark composition. To ensure fairness, in the first iteration, the workload script assigned benchmarks based on the application's processor-preference for three baseline setups. When all cores are fully occupied, the script randomly assigned benchmarks remaining in the workload batch to the next available free core to best mimic a dynamic workload scenario. In a dynamic workload scenario, the incoming benchmarks can not be predicted, but the overall ratio can be estimated. If a workload batch is finished, the script regenerates an identical batch from which to select. This process continues until the evaluation period ends. To ensure a fair comparison, we used the same random seed so that each configuration has the same benchmark selection outcome for every run. Each experiment was run for 75 minutes. The rationale behind this is that most of the benchmarks execute in about 3 to 5 minutes when running on the x86-based processor; 75 minutes is large enough to mitigate the impact of noise.

7.2 Experiment Results

We evaluate our proposed framework to understand the effectiveness of our purposed design (het-dynamic) on improving system throughput on a diverse range of workloads. To this end, we conducted two sets of experiments. The first set considered workloads composed of two applications: one high processor-preference (SIMD) benchmark and one non-high processor-preference benchmark. The goal of this two-application workload experiment is to determine what workload compositions yield throughput gains for our het-dynamic design. By focusing on only two benchmarks, we can more easily narrow down the impact of individual benchmarks with different processor-preferences.

Using the insights gained from these experiments, our second set of experiments considered a more realistic workload consisting of multiple benchmarks with various degrees of processor-preference. To exhaustively test our extended SIMD migration support for the Popcorn Linux Framework, we only designated benchmarks with large SIMD regions as high processor-preference benchmarks. For easier result comparison, we used the Hydro benchmark from the Livermoore Loops suite [83] as the designated SIMD benchmark for both sets of experiments.

7.2.1 Two-application Workloads

Figures 7.1 and 7.2 show the system throughput of a two-application workload under two different high processor-preference (SIMD)/non-SIMD ratios: 12.5/87.5% and 25/75% respectively. The x-axis represents the benchmarks tested as the non-SIMD benchmark, arranged in increasing slowdown order according to Figure 1.4, with the leftmost being EP from the NPB suite [11], which has the lowest slowdown, and the rightmost being CG from NPB, which has the highest slowdown. The y-axis is the total number of benchmarks completed in a testing period of 75 minutes, where higher is better. The upper pink portion of each

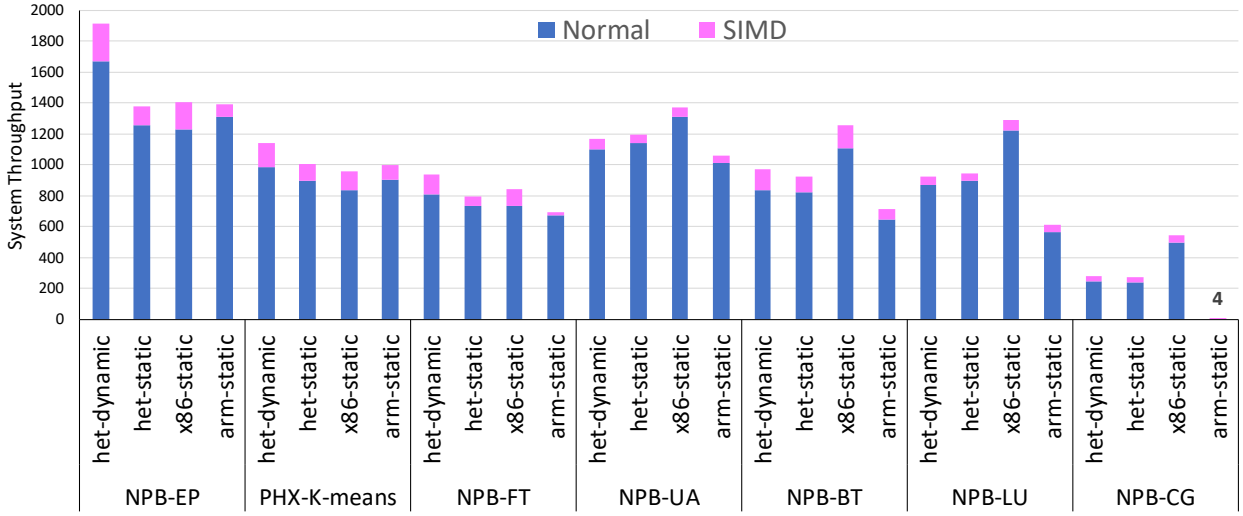
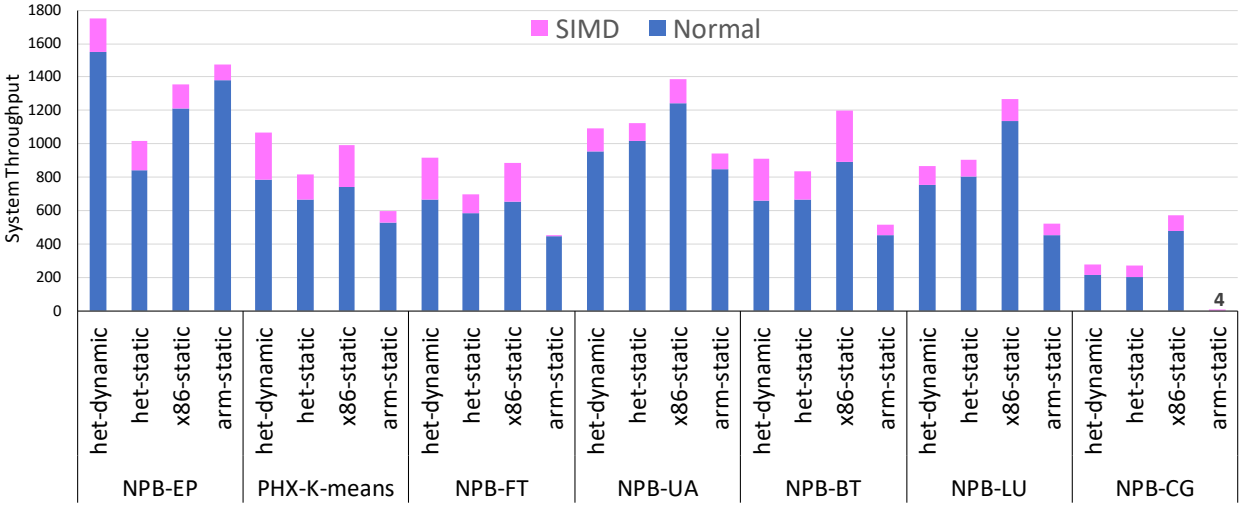


Figure 7.1: Throughput of two application workloads with 1/8 SIMD/non-SIMD ratio.

bar represents the number of SIMD benchmarks completed. and the bottom blue portion represents the non-SIMD benchmarks completed in a testing period.

These figures reveal that het-dynamic consistently outperforms het-static. For the 12.5% SIMD scenario, het-dynamic outperforms all baselines in three cases: EP, K-means, and FT. EP has the best performance gain of ∼36% over the next best baseline. However, the performance gain shrinks to ∼14% for K-means and ∼11% for FT. For other benchmarks with increasing processor-preference toward x86-based cores, x86-static consistently outperforms het-dynamic and also het-static and ARM-static. An average of 6.3% of non-SIMD benchmarks migrated to ARM in order to vacate x86-based cores for SIMD benchmarks, and no SIMD benchmark is spilled onto ARM.

Similar trends occur in the 25% SIMD scenario – het-dynamic still has better performance for EP, K-means, and FT. However, the performance gain drops down to ∼19%, ∼8%, and ∼3%, respectively. This performance decrease can be attributed to an average of 5.1% SIMD benchmarks spilled onto ARM. However, every spilled SIMD benchmark eventually migrates back to x86-based cores, which mitigated some throughput reduction. The average



h

Figure 7.2: Throughput of two application workloads with 1/4 SIMD/non-SIMD ratio.

percentage of non-SIMD benchmarks migrated to vacate x86-based cores for SIMD benchmarks is around 12.8%. This almost doubled percentage is likely due to the 2x increase in the SIMD ratio. We also tested larger SIMD benchmark ratios of 50% (i.e., 50% SIMD benchmark ratio) and 100% (i.e., all SIMD benchmarks). In both cases, het-dynamic shows no performance gain over x86-static.

From these results, we can draw several conclusions. First, het-dynamic performs well with workloads consist of a low ratio of high processor-preference (SIMD) applications. For workloads with larger SIMD benchmark ratios, het-dynamic lacks enough application-preferred x86-based cores to execute these high processor-preference benchmarks; thus, spilling SIMD benchmarks onto ARM hurts throughput. The spilled SIMD benchmarks are forced to execute on a significantly slower (lower processor-preference) architecture until one of the existing SIMD benchmarks on an x86-based core finishes. These spilled benchmarks degrade throughput. Reduction in application diversity will reduce the need for processor-diversity in systems.

Second, het-dynamic yields better throughput for low processor-preference applications. In

both Figures 7.1 and 7.2, all three benchmarks either belong to or close enough to the low processor-preference group. Low processor-preference applications allow het-dynamic to compensate for the smaller performance slowdown with the higher number of Cavium ThunderX cores. These additional cores allow the system to execute more applications in parallel, thereby outperforming x86-static. Despite a low SIMD ratio, het-dynamic is also better than ARM-static because there are still a few SIMD benchmarks in the mix. In the ARM-static case, the SIMD benchmarks become stragglers due to extreme slowdowns, which ultimately harms throughput.

Lastly, the impact of het-dynamic's scheduler on system throughput cannot be ignored. The scheduler allows het-dynamic to better allocate resources based on the incoming application's processor-preference. The impact of the scheduler can also be reflected by the fact that the EP-Hydro (SIMD) workload combination has the best performance gain in both 1/4 and 1/8 scenarios. The EP-Hydro workload combination contains two benchmarks that have the most diverse processor-preferences. Thus, migrating processes across ISAs to match their processor-preferences enables het-dynamic to obtain the most significant system throughput gain (36%) over the next best baseline.

7.2.2 Multi-application Workloads

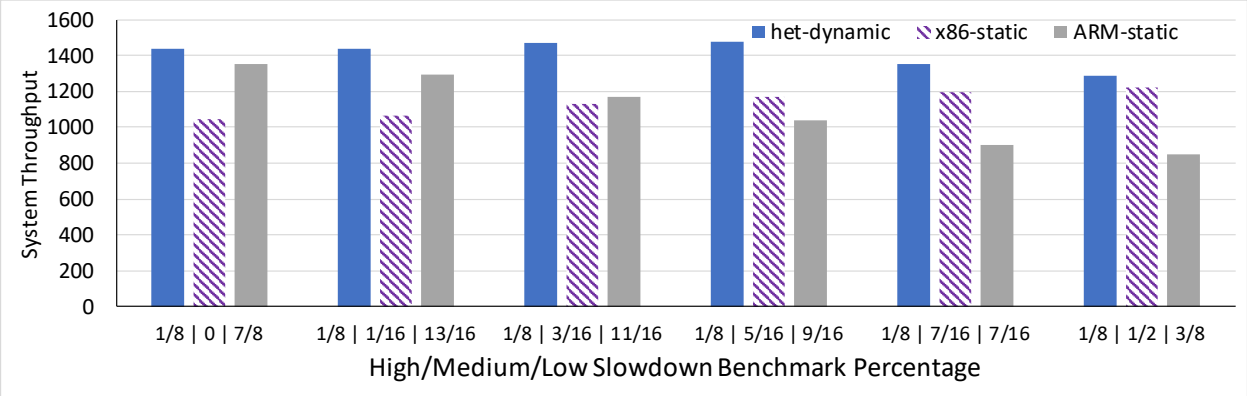


Figure 7.3: Throughput of multi-application workloads with 1/8 high processor-preference (high slowdown) application ratio.

Figures 7.3 and 7.4 show the system throughput of workloads composed of more than two benchmarks with the high processor-preference benchmarks (SIMD only) fixed at 12.5% and 25% of total workloads, respectively. For both scenarios, we fixed the ratio of the high processor-preference group benchmarks. The high processor-preference group only contains benchmarks with a large SIMD region to test the effectiveness of our SIMD migration capability. For each scenario, we varied the ratio of benchmarks belonging to low and medium processor-preference groups. The x-axis represents each group's ratio, and the y-axis shows the system throughput.

In the 12.5% high processor-preference group (SIMD) ratio scenario, het-dynamic outperforms both x86-static and ARM-static in all tested cases with an average gain of 14.6% and a maximum gain of ∼26% over the next best homogeneous baseline with workloads consisting of 12.5%, 31.25%, and 56.25% of high, medium, and low processor-preference group members respectively. However, similar to the two-application workload experiments, the performance gain shrinks as the percentage of medium processor-preference benchmarks increases. This decrease in performance is due to our scheduling policies reduce effectiveness

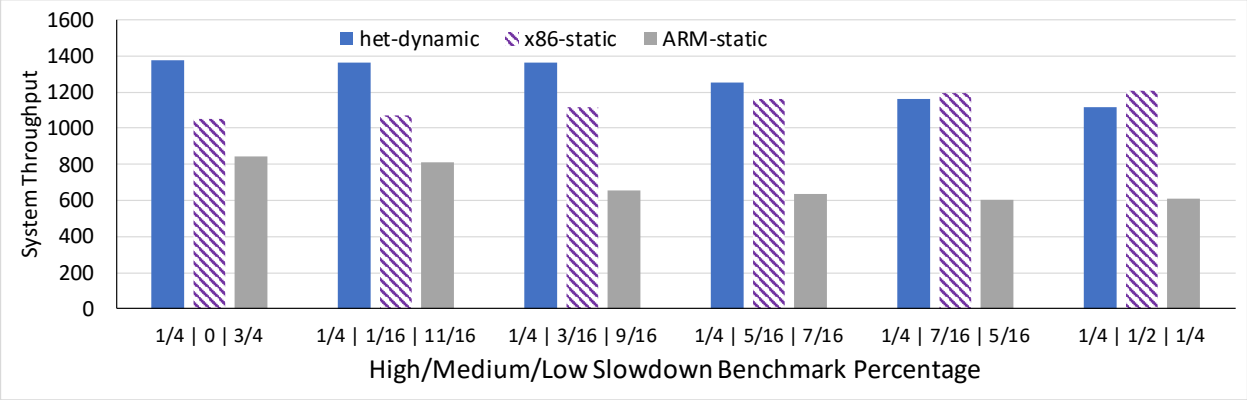


Figure 7.4: Throughput of multi-application workloads with 1/4 high processor-preference (high slowdown) application ratio.

when workloads consist of more benchmarks biased towards a single processor. An average of 23.1% of total benchmarks is migrated to ARM during the experiment to vacate x86 cores for benchmarks with higher preference. 24.7% of total benchmarks on average are eventually migrated back to an x86 core. This larger migrate back percentage indicates that our scheduler can utilize the new framework to allow benchmarks to finish in chronological order if there is no difference in processor-preferences.

Similar trends also occur in the 25% scenario – het-dynamic still has the best performance in four out of six test cases, achieving a maximum performance gain of 31.1%. An average of 28% of total benchmarks is migrated to ARM during the experiment to vacate cores for benchmarks with higher processor-preference. An average of 24.4% is eventually migrated back to the x86-based server. We tested larger high processor-preference group ratios (e.g., 50%, 100%). In both cases, het-dynamic has no performance gain over x86-static in all workload cases.

From multi-application workload experiments, we gain further understanding of our heterogeneous-ISA systems design. Het-dynamic system configuration is best equipped to handle workloads that contain large low processor-preference and medium processor-preference application ratios. These additional evaluations further expand the “sweet spot” of het-dynamic because,

in the two-application scenario, we do not see any benchmark that has medium processor-preference benefiting from het-dynamic.

The multi-application experiments thus reveal that in a more realistic workload with multiple diverse applications, het-dynamic can achieve increased performance for applications with a higher degree of processor-preferences. Het-dynamic can, therefore, achieve higher throughput over a broader workload spectrum than comparable homogeneous setups. This additional performance improvement is achieved by better matching each application's processor-preference to an optimal processor with added migration and scheduling capabilities, improving the system throughput.

Chapter 8

Conclusion & Future Works

This chapter wraps up the entire thesis. We illustrate the conclusions we have drawn from this research and discuss the potential importance of having processor-diversity considered as an important dimension in future processor designs.

This chapter starts with Section 8.1 discussing the conclusions we have made. Future directions in this research area are provided in Section 8.2.

8.1 Conclusion

We championed the usability of heterogeneous-ISA systems compared to mainstream homogeneous-ISA systems. We explored whether heterogeneous-ISA systems can be leveraged for performance gains. We proved that having processors with different ISAs provide a new interesting dimension in processor design. We extended the Popcorn Linux [12] framework to support migration inside SIMD regions. Efficiently using each application's processor-preference and dynamically migrating them to use optimal cores in heterogeneous-ISA systems can result in significant performance gains over traditional homogeneous-ISA systems. In the end, we reinforced the support for developing a commodity heterogeneous-ISA chip multi-processor with cache-coherent shared memory.

Our work's main conclusion is that there is "no processor design that fits all." The fact that het-dynamic allows two servers with vastly different processors that are five years apart in production to outperform two servers with the same 2018-released processors within the same budget is a strong validation of our results.

8.2 Future works

We believe that our work only scratches the surface of the heterogeneous-ISA system space. Many promising future directions still exist. For example, this thesis focused only on improving system performance but scoped out on investigating energy costs. System energy optimizations are not investigated in this thesis mainly due to the process node gap between the two servers that we selected. The Cavium ThunderX server (initially released in 2014) uses a more power consuming 28 nm process, whereas the Xeon server (released in 2018) uses a more recent 14 nm process. Combined with the fact that the Cavium ThunderX does not implement many energy-saving features such as low-power states, Dynamic Voltage Frequency Scaling, and clock gating, the Cavium ThunderX is not an energy-efficient processor.

Recent ARM servers such as Marvell's ThunderX2 [110] (Marvell recently required Cavium) and Ampere's eMAG server [25] use 14nm process and are likely more energy-efficient and have higher performance than Cavium ThunderX. Thus, pairing together either one of the two latest ARM servers with our existing x86-based servers are anticipated to have a performance benefit over homogeneous-ISA systems in more scenarios and potentially enable us to investigate system energy optimizations. However, due to the significant amount of engineering efforts required in porting the Popcorn Linux infrastructure for cross-ISA migration on these platforms, we can only extrapolate what will Xeon-ThunderX2, and Xeon-Ampere

het-dynamic setup performs using het-static configuration. Het-static setup for both configurations can be easily measured as the setup does not involve cross-ISA migration. Figure 8.1 shows the het-static setup performance number for the two new setups. For comparison purposes, we also put in the het-dynamic and x86-static performance numbers from Xeon-ThunderX setup earlier.

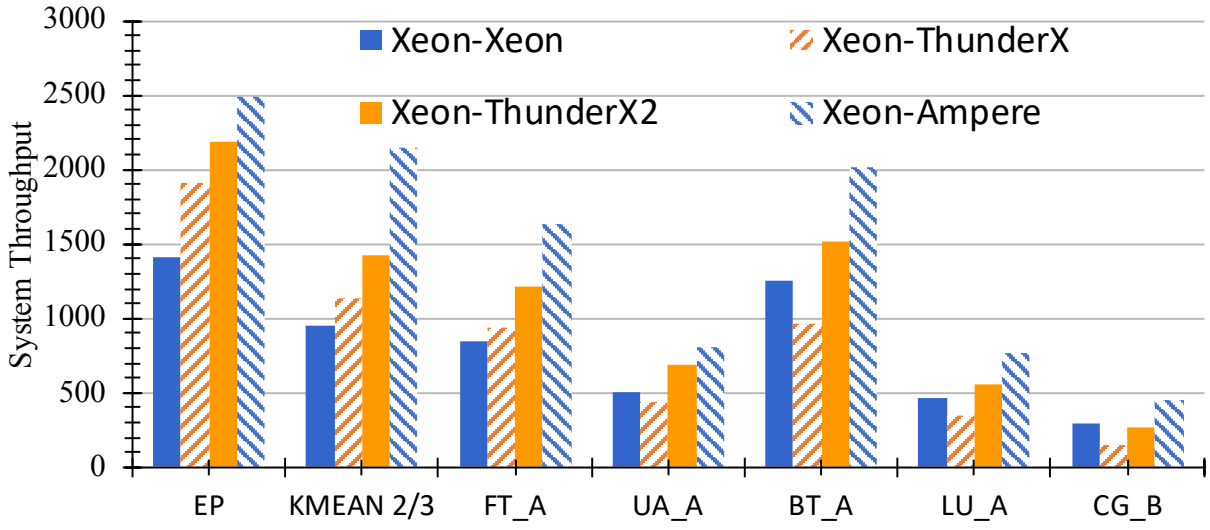


Figure 8.1: Throughput of het-static and x86-static on Marvell ThunderX2 [110] and Ampere eMAG [25] servers.

As shown in Figure 8.1, het-static on Xeon-Ampere shows the best performance with average throughput gain of 100% over het-dynamic on Xeon-ThunderX and 76% over x86-static on Xeon-Xeon. From our evaluation results, het-dynamic outperforms het-static on similar heterogeneous-ISA servers in almost every scenario. Thus, extrapolating from Figure 8.1, het-dynamic will likely outperform het-static and x86-static on these newer servers as well. Thus future ARM-based processors combined with the latest x86-based processors offer even more intriguing processor design choices. Another promising direction is scheduling. Recent results such as [87] reveal that machine learning-based approaches can accurately predict program performance for superior scheduling policies. This approach can be leveraged in the heterogeneous-ISA SIMD scheduling space, as well.

Bibliography

[1] Xeon gold 5118 - intel. URL https://en.wikichip.org/wiki/intel/xeon_gold/5118.

[2] A landscape of the new dark silicon design regime. *IEEE Micro, Micro, IEEE*, (5):8, 2013. ISSN 0272-1732. URL http://login.ezproxy.lib.vt.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.6583151&site=eds-live&scope=site.

[3] Ayaz Akram. A study on the impact of instruction set architectures on processor's performance, 2017. Master Thesis, Western Michigan University.

[4] Ayaz Akram and Lina Sawalha. The impact of isas on performance. In *Workshop on Duplicating, Deconstructing and Debunking (WDDD) co-located with 44th International Symposium on Computer Architecture (ISCA), Toronto, Canada*, 2017.

[5] Amazon. Ec2 instances powered by arm-based aws graviton processors, 2018. https://aws.amazon.com/blogs/aws/new-ec2-instances-a1-powered-by-arm-based-aws-graviton-processors/.

[6] Kazumaro Aoki, Fumitaka Hoshino, Tetsutaro Kobayashi, and Hiroaki Oguro. Elliptic curve arithmetic using simd. In George I. Davida and Yair Frankel, editors, *Information Security*, pages 235–247, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-45439-7.

[7] ARM. Neon. Technical report, November 2018. https://developer.arm.com/technologies/neon.

[8] ARM. Arm hpc tools for sve. Technical report, November 2018. https://developer.arm.com/products/software-development-tools/hpc/sve.

[9] Mehmet Ali Arslan, Flavius Gruian, Krzysztof Kuchcinski, and Andréas Karlsson. Code generation for a simd architecture with custom memory organisation. In *Design and Architectures for Signal and Image Processing (DASIP), 2016 Conference on*, pages 90–97. IEEE, 2016.

[10] David F. Bacon, Susan L. Graham, and Oliver J. Sharp. Compiler transformations for high-performance computing. *ACM Comput. Surv.*, 26(4):345–420, December 1994. ISSN 0360-0300.

[11] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon. Nas parallel benchmark results. In *Supercomputing '92:Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, pages 386–393, Nov 1992.

[12] 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, pages 645–659, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4465-4. doi: 10.1145/3037697.3037738. URL http://doi.acm.org/10.1145/3037697.3037738.

[13] 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. IEEE, 2011.

[14] Eli Benderskys. Analyzing function cfgs with llvm. URL https://eli.thegreenplace.net/2013/09/16/analyzing-function-cfgs-with-llvm.

[15] Sergey Blagodurov, Sergey Zhuravlev, Alexandra Fedorova, and Ali Kamali. A case for numa-aware contention management on multicore systems. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 557–558. ACM, 2010.

[16] Chandler Carruth Bob Wilson, Diego Novillo. Pgo and llvm, status and current work, 2007. URL https://llvm.org/devmtg/2013-11/slides/Carruth-PGO.pdf.

[17] Shekhar Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual Design Automation Conference*, pages 746–749. ACM, 2007.

[18] Dennis Bradford, Sundaram Chinthamani, Jesus Corbal, Adhiraj Hassan, Ken Janik, and Nawab Ali. Knights mill: New intel processor for machine learning. In *Hot Chips 29*, August 2017.

[19] D. Callahan, J. Dongarra, and D. Levine. Vectorizing compilers: A test suite and results. In *Proceedings of the 1988 ACM/IEEE Conference on Supercomputing*, Supercomputing '88, pages 98–105, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press. ISBN 0-8186-0882-X.

[20] Cavium. Thunderx cn8890 - cavium, 2013. URL https://en.wikichip.org/wiki/cavium/thunderx/cn8890.

[21] Hao Chen, Nicholas S Flann, and Daniel W Watson. Parallel genetic simulated annealing: a massively parallel simd algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):126–136, 1998.

[22] 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. IEEE, 2009.

[23] Chi Ching Chi, Mauricio Alvarez-Mesa, Benjamin Bross, Ben Juurlink, and Thomas Schierl. Simd acceleration for hevc decoding. *IEEE Transactions on circuits and systems for video technology*, 25(5):841–855, 2015.

[24] Nathan Clark, Amir Hormati, Sami Yehia, Scott Mahlke, and Krisztian Flautner. Liquid simd: Abstracting simd hardware using lightweight dynamic mapping. In *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 216–227. IEEE, 2007.

[25] Ampere Computing. Ampere processors, 2018. URL https://amperecomputing.com/product/.

[26] Jason Cong and Bo Yuan. Energy-efficient scheduling on heterogeneous multi-core architectures. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pages 345–350. ACM, 2012.

[27] Gregory W Cook and Edward J Delp. An investigation of scalable simd i/o techniques with application to parallel jpeg compression. *Journal of Parallel and distributed computing*, 30(2):111–128, 1995.

[28] Keith Cooper and Linda Torczon. *Engineering a compiler*. Elsevier, 2011.

[29] R. Cypher and J. L. C. Sanz. Simd architectures and algorithms for image processing and computer vision. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(12):2158–2174, Dec 1989. ISSN 0096-3518.

[30] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 365–376, June 2011.

[31] Michael Feldman. Intel dumps knights hill, future of xeon phi product line uncertain, November 2017. https://www.top500.org/news/intel-dumps-knights-hill-future-of-xeon-phi-product-line-uncertain/.

[32] Michael Feldman. Intel ships xeon skylake processor with integrated fpga, May 2018. https://www.top500.org/news/intel-ships-xeon-skylake-processor-with-integrated-fpga/.

[33] Sheng-Yu Fu, Ding-Yong Hong, Jan-Jan Wu, Pangfeng Liu, and Wei-Chung Hsu. Simd code translation in an enhanced hqemu. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pages 507–514. IEEE, 2015.

[34] Sheng-Yu Fu, Jan-Jan Wu, and Wei-Chung Hsu. Improving simd code generation in qemu. In *Proceedings of the 2015 design, automation & test in europe conference & exhibition*, pages 1233–1236. EDA Consortium, 2015.

[35] Sheng-Yu Fu, Ding-Yong Hong, Yu-Ping Liu, Jan-Jan Wu, and Wei-Chung Hsu. Optimizing data permutations in structured loads/stores translation and simd register mapping for a cross-isa dynamic binary translator. *Journal of Systems Architecture*, 98:173–190, 2019.

[36] Xinwei Fu, Dongyoon Lee, and Changhee Jung. nadroid: statically detecting ordering violations in android applications. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, pages 62–74. ACM, 2018.

[37] Xinwei Fu, Talha Ghaffar, James C Davis, and Dongyoon Lee. Edgewise: a better

stream processing engine for the edge. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 929–946, 2019.

[38] Ramy Gad, Tim Süß, and André Brinkmann. Compiler driven automatic kernel context migration for heterogeneous computing. In *2014 IEEE 34th International Conference on Distributed Computing Systems*, pages 389–398. IEEE, 2014.

[39] P. Greenhalgh. big.little processing with arm cortex-a15 & cortex-a7, 2011. Technical report, ARM.

[40] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. Biscuit: A framework for near-data processing of big data workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 153–165, Piscataway, NJ, USA, 2016. IEEE Press. ISBN 978-1-4673-8947-1. doi: 10.1109/ISCA.2016.23. URL https://doi.org/10.1109/ISCA.2016.23.

[41] Arthur Hennequin, Ian Masliah, and Lionel Lacassagne. Designing efficient simd algorithms for direct connected component labeling. In *Proceedings of the 5th Workshop on Programming Models for SIMD/Vector Processing*, page 4. ACM, 2019.

[42] Johannes Hofmann, Jan Treibig, Georg Hager, and Gerhard Wellein. Comparing the performance of different x86 simd instruction sets for a medical imaging application on modern multi-and manycore chips. In *Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing*, pages 57–64. ACM, 2014.

[43] Ding-Yong Hong, Sheng-Yu Fu, Yu-Ping Liu, Jan-Jan Wu, and Wei-Chung Hsu. Exploiting longer simd lanes in dynamic binary translation. In *Parallel and Distributed Systems (ICPADS), 2016 IEEE 22nd International Conference on*, pages 853–860. IEEE, 2016.

[44] IK Hong, ST Chung, HK Kim, YB Kim, YD Son, and ZH Cho. Ultra fast symmetry and simd-based projection-backprojection (ssp) algorithm for 3-d pet image reconstruction. *IEEE transactions on medical imaging*, 26(6):789–803, 2007.

[45] Joel Hruska. Intel uses new foveros 3d chip-stacking to build core, atom on same silicon, Dec 2018. URL https://bit.ly/2SSQ62R.

[46] Connor Imes and Henry Hoffmann. Minimizing energy under performance constraints on embedded platforms: resource allocation heuristics for homogeneous and single-isa heterogeneous multi-cores. *ACM SIGBED Review*, 11(4):49–54, 2015.

[47] Hiroshi Inoue, Takao Moriyama, Hideaki Komatsu, and Toshio Nakatani. Aa-sort: A new parallel sorting algorithm for multi-core simd processors. In *Parallel Architecture and Compilation Techniques, 2007. PACT 2007. 16th International Conference on*, pages 189–198. IEEE, 2007.

[48] Texas Instruments. Omap5912 multimedia processor device overview and architecture reference guide, 2004.

[49] Intel. Intel advanced vector extensions 512 (intel avx-512), 2013. https://intel.ly/2SyYl4i.

[50] Intel. Intel xeon silver 4110 processor product specifications, 2017. URL https://ark.intel.com/content/www/us/en/ark/products/123547/intel-xeon-silver-4110-processor-11m-cache-2-10-ghz.html.

[51] Intel. Intel xeon gold 5118 processor product specifications, 2017. URL https://ark.intel.com/products/120473/Intel-Xeon-Gold-5118-Processor-16-5M-Cache-2-30-GHz-.

[52] Intel. Intel Xeon processor scalable family, 2018. https://intel.ly/2t1apTH.

[53] 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 *2nd IEEE international conference on cloud computing technology and science*, pages 159–168. IEEE, 2010.

[54] Aamer Jaleel, Hashem H Najaf-Abadi, Samantika Subramaniam, Simon C Steely, and Joel Emer. Cruise: cache replacement and utility-aware scheduling. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 249–260. ACM, 2012.

[55] Xiaowei Jiang, Asit Mishra, Li Zhao, Ravishankar Iyer, Zhen Fang, Sadagopan Srinivasan, Srihari Makineni, Paul Brett, and Chita R Das. Access: Smart scheduling for asymmetric cache cmps. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 527–538. IEEE, 2011.

[56] Feras Karablieh and Rida A Bazzi. Heterogeneous checkpointing for multithreaded applications. In *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.*, pages 140–149. IEEE, 2002.

[57] Mohamed Karaoui, Anthony Carno, Robert Lyerly, Sang-Hoon Kim, Pierre Olivier, Changwoo Min, and Binoy Ravindran. Scheduling HPC workloads on heterogeneous-ISA architectures. In *24th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP'19)*, February 2019. Poster paper.

[58] Patrick Kennedy. Cavium thunderx2 review and benchmarks a real arm server option, Jun 2018. URL https://www.servethehome.com/cavium-thunderx2-review-benchmarks-real-arm-server-option/.

[59] Nam Sung Kim and Pankaj Mehra. Practical near-data processing to evolve memory

and storage devices into mainstream heterogeneous computing systems. In *Proceedings of the 56th Annual Design Automation Conference 2019*, page 22. ACM, 2019.

[60] David Koufaty, Dheeraj Reddy, and Scott Hahn. Bias scheduling in heterogeneous multi-core architectures. In *Proceedings of the 5th European conference on Computer systems*, pages 125–138. ACM, 2010.

[61] Vlad Krasnov. On the dangers of intel's frequency scaling, November 2017. https://blog.cloudflare.com/on-the-dangers-of-intels-frequency-scaling/.

[62] Rakesh Kumar, Keith I Farkas, Norman P Jouppi, Parthasarathy Ranganathan, and Dean M Tullsen. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 81. IEEE Computer Society, 2003.

[63] 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 *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 64–75. IEEE, 2004.

[64] Rakesh Kumar, Dean M Tullsen, and Norman P Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *Parallel Architectures and Compilation Techniques (PACT), 2006 International Conference on*, pages 23–32. IEEE, 2006.

[65] Chris Lamont. Introduction to intel advanced vector extensions. Technical report, June 2011. https://intel.ly/2ETCWyt.

[66] Chris Lattner. Introduction to the llvm compiler system. In *Proceedings of International Workshop on Advanced Computing and Analysis Techniques in Physics Research, Erice, Sicily, Italy*, page 19, 2008.

[67] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, page 75. IEEE Computer Society, 2004.

[68] G. Lee, H. Park, S. Heo, K. Chang, H. Lee, and H. Kim. Architecture-aware automatic computation offload for native applications. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 521–532, Dec 2015. doi: 10.1145/2830772.2830833.

[69] Janghaeng Lee, Mehrzad Samadi, Yongjun Park, and Scott Mahlke. Transparent cpu-gpu collaboration for data-parallel kernels on heterogeneous systems. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, pages 245–256. IEEE Press, 2013.

[70] Baptiste Lepers, Vivien Quéma, and Alexandra Fedorova. Thread and memory placement on {NUMA} systems: Asymmetry matters. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, pages 277–289, 2015.

[71] Jianhui Li, Qi Zhang, Shu Xu, and Bo Huang. Optimizing dynamic binary translation for simd instructions. In *International Symposium on Code Generation and Optimization (CGO'06)*, pages 12–pp. IEEE, 2006.

[72] Felix Xiaozhu Lin, Zhen Wang, and Lin Zhong. K2: A mobile operating system for heterogeneous coherence domains. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 285–300, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2305-5.

[73] Jiawen Liu, Hengyu Zhao, Matheus A Ogleari, Dong Li, and Jishen Zhao. Processing-in-memory for energy-efficient neural network training: A heterogeneous approach.

In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 655–668. IEEE, 2018.

[74] Yu-Ping Liu, Ding-Yong Hong, Jan-Jan Wu, Sheng-Yu Fu, and Wei-Chung Hsu. Exploiting asymmetric simd register configurations in arm-to-x86 dynamic binary translation. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 343–355. IEEE, 2017.

[75] LLVM. How to build clang and llvm with profile-guided optimizations, 2003. URL https://llvm.org/docs/HowToBuildWithPGO.html.

[76] LLVM. Auto-vectorization in llvm, 2003. URL https://llvm.org/docs/Vectorizers.html.

[77] LLVM. Profile guided optimization, 2007. URL https://clang.llvm.org/docs/UsersManual.html#profile-guided-optimization.

[78] Arm Ltd. Technologies | dynamiq arm developer, 2017. URL https://developer.arm.com/technologies/dynamiq.

[79] Luca Lugini, Vinicius Petrucci, and Daniel Mosse. Online thread assignment for heterogeneous multicore systems. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 538–544. IEEE, 2012.

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

[81] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible

co-locations. In *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*, pages 248–259. ACM, 2011.

[82] Marvell. Liquidio ii 10/25gbe Adapter family, 2019. https://bit.ly/2H7NWLk.

[83] F H McMahon. *The Livermore Fortran kernels: a computer test of the numerical performance range*. Lawrence Berkeley Nat. Lab., Berkeley, CA, 1986. URL https://cds.cern.ch/record/178064.

[84] Sparsh Mittal. A survey of techniques for architecting and managing asymmetric multicore processors. *ACM Comput. Surv.*, 48(3):45:1–45:38, February 2016. ISSN 0360-0300. doi: 10.1145/2856125. URL http://doi.acm.org/10.1145/2856125.

[85] Sparsh Mittal and Jeffrey S. Vetter. A survey of cpu-gpu heterogeneous computing techniques. *ACM Comput. Surv.*, 47(4):69:1–69:35, July 2015. ISSN 0360-0300. doi: 10.1145/2788396. URL http://doi.acm.org/10.1145/2788396.

[86] Chuck Moore. Data processing in exascale-class computer systems. In *The Salishan Conference on High Speed Computing*, 2011.

[87] D. Nemirovsky, T. Arkose, N. Markovic, M. Nemirovsky, O. Unsal, and A. Cristal. A machine learning approach for performance prediction and scheduling on heterogeneous cpus. In *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 121–128, Oct 2017.

[88] Netronome. Agilio SmartNICs, 2019. https://www.netronome.com/products/agilio-cx/.

[89] Nvidia. The benefits of multiple cpu cores in mobile devices, 2010. URL https://www.nvidia.com/content/PDF/tegra_white_papers/Benefits-of-Multi-core-CPUs-in-Mobile-Devices_Ver1.2.pdf.

[90] Nvidia. Variable smp - a multi-core cpu architecture for low power and high performance, 2011. URL https://www.nvidia.com/content/PDF/tegra_white_papers/tegra-whitepaper-0911b.pdf.

[91] Pierre Olivier, Sang-Hoon Kim, and Binoy Ravindran. OS support for thread migration and distribution in the fully heterogeneous datacenter. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, pages 174–179, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5068-6. doi: 10.1145/3102980.3103009. URL http://doi.acm.org/10.1145/3102980.3103009.

[92] OpenCV. Opencv: Introduction. Technical report, November 2018. https://docs.opencv.org/3.3.1/d1/dfb/intro.html.

[93] Edson Luiz Padoin, Laércio Lima Pilla, Márcio Castro, Francieli Z Boito, Philippe Olivier Alexandre Navaux, and Jean-François Méhaut. Performance/energy trade-off in scientific computing: the case of arm big. little and intel sandy bridge. *IET Computers & Digital Techniques*, 9(1):27–35, 2014.

[94] Alex Pajuelo, Antonio González, and Mateo Valero. Speculative dynamic vectorization. In *ACM SIGARCH Computer Architecture News*, volume 30, pages 271–280. IEEE Computer Society, 2002.

[95] Szilárd Páll and Berk Hess. A flexible algorithm for calculating pair interactions on simd architectures. *Computer Physics Communications*, 184(12):2641–2650, 2013.

[96] S. Panneerselvam and M. Swift. Rinnegan: Efficient resource use in heterogeneous architectures. In *2016 International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 373–386, Sept 2016. doi: 10.1145/2967938.2967964.

[97] Vinicius Petrucci, Orlando Loques, and Daniel Mossé. Lucky scheduling for energy-efficient heterogeneous multi-core systems. In *HotPower*, 2012.

[98] Ioannis Pitas, editor. *Parallel Algorithms: For Digital Image Processing, Computer Vision and Neural Networks*. John Wiley & Sons, Inc., New York, NY, USA, 1993. ISBN 0-471-93566-2.

[99] Andreas Prodromou, Ashish Venkat, and Dean M Tullsen. Deciphering predictive schedulers for heterogeneous-isa multicore architectures. 2019.

[100] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 13–24, Feb 2007. doi: 10.1109/HPCA.2007.346181.

[101] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 7. ACM, 2012.

[102] Greg Sadowski. Design challenges facing cpu-gpu-accelerator integrated heterogeneous systems. In *Design Automation Conference (DAC'14)*, 2014.

[103] David Schor. Intel reveals 10nm sunny cove core, a new core roadmap, and teases ice lake chips, Dec 2018. URL https://bit.ly/2NLEbTg.

[104] J. M. Shalf and R. Leland. Computing beyond moore's law. *Computer*, 48(12):14–23, Dec 2015. ISSN 0018-9162. doi: 10.1109/MC.2015.374.

[105] Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, Kurt B Ferreira, Jon Stearley, John Shalf, and Sudhanva Gurumurthi. Memory errors in modern systems: The good, the bad, and the ugly. *ACM SIGPLAN Notices*, 50(4):297–310, 2015.

[106] Richard M Stallman et al. Using the gnu compiler collection. *Free Software Foundation*, 4(02), 2003.

[107] David Tam, Reza Azimi, and Michael Stumm. Thread clustering: sharing-aware scheduling on smp-cmp-smt multiprocessors. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 47–58. ACM, 2007.

[108] Michael B Taylor. A landscape of the new dark silicon design regime. *IEEE Micro*, 33 (5):8–19, 2013.

[109] Marvell Technology. ThunderX ARM-based processors, 2013. https://www.marvell.com/server-processors/thunderx-arm-processors/.

[110] Marvell Technology. ThunderX2 ARM-based processors, 2018. https://www.marvell.com/server-processors/thunderx2-arm-processors/.

[111] Po-An Tsai, Changping Chen, and Daniel Sanchez. Adaptive scheduling for systems with asymmetric memory hierarchies. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 641–654. IEEE, 2018.

[112] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, and Joel Emer. Scheduling heterogeneous multi-cores through performance impact estimation (pie). In *ACM SIGARCH Computer Architecture News*, volume 40, pages 213–224. IEEE Computer Society, 2012.

[113] 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*, pages 177–187. IEEE, 2013.

[114] 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, pages 121–132, Piscataway, NJ, USA, 2014. IEEE Press. ISBN 978-1-4799-4394-4. URL http://dl.acm.org/citation.cfm?id=2665671.2665692.

[115] Ashish Venkat, H. Basavaraj, and D. M. Tullsen. Composite-isa cores: Enabling multi-isa heterogeneity using a single isa. HPCA, 2019.

[116] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation cores: Reducing the energy of mature computations. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pages 205–218, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-839-1. doi: 10.1145/1736020.1736044.

[117] David G. von Bank, Charles M. Shub, and Robert W. Sebesta. A unified model of pointwise equivalence of procedural computations. *ACM Trans. Program. Lang. Syst.*, 16(6):1842–1874, November 1994. ISSN 0164-0925. doi: 10.1145/197320.197402. URL http://doi.acm.org/10.1145/197320.197402.

[118] M Mitchell Waldrop. The chips are down for moore's law. *Nature News*, 530(7589): 144, 2016.

[119] Xin Wang, Yunchun Li, and Xiaoxiang Zou. Flow-based sm4 encryption via tilera multiprocessor. In *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 741–748. IEEE, 2016.

[120] Paul Whytock. 3d tri-gate device keeps moore's law in order, Nov 2012. URL https://www.electronicdesign.com/components/3d-tri-gate-device-keeps-moore-s-law-order.

[121] R. S. Williams. What's next? [the end of moore's law]. *Computing in Science Engineering*, 19(2):7–13, Mar 2017. ISSN 1521-9615. doi: 10.1109/MCSE.2017.31.

[122] Michael Witbrock and Marco Zagha. An implementation of backpropagation learning on gf11, a large simd parallel computer. *Parallel Computing*, 14(3):329 – 346, 1990. ISSN 0167-8191. doi: https://doi.org/10.1016/0167-8191(90)90085-N. URL http://www.sciencedirect.com/science/article/pii/016781919090085N.

[123] Demin Xiong and Duane F Marble. Strategies for real-time spatial analysis using massively parallel simd computers: an application to urban traffic flow analysis. *International Journal of Geographical Information Systems*, 10(6):769–789, 1996.

[124] Xinhai Xu, Yufei Lin, Tao Tang, and Yisong Lin. Hial-ckpt: A hierarchical application-level checkpointing for cpu-gpu hybrid systems. In *2010 5th International Conference on Computer Science & Education*, pages 1895–1899. IEEE, 2010.

[125] Da Zhang, Hao Wang, Kaixi Hou, Jing Zhang, and Wu-chun Feng. pdindel: Accelerating indel detection on a multicore cpu architecture with simd. In *2015 IEEE 5th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*, pages 1–6. IEEE, 2015.

[126] Yuhao Zhu and Vijay Janapa Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 13–24. IEEE, 2013.

[127] Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. Addressing shared

resource contention in multicore processors via scheduling. In *ACM Sigplan Notices*, volume 45, pages 129–142. ACM, 2010.