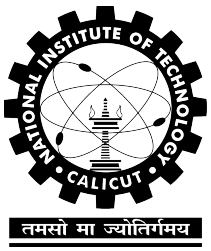
Binary ISA Prediction in Fine-grain Scheduling in Heterogeneous ISA CMP

CS4099 Project Final Report

Submitted by

Hatim Shakir	**Reg No: B200830CS**
Md. Arif Raza Mansuri	**Reg No: B200808CS**
Anagha M V	**Reg No: B200762CS**

Under the Guidance of
Dr. Nirmal Kumar Boran



Department of Computer Science and Engineering
National Institute of Technology Calicut
Calicut, Kerala, India - 673 601

May 7, 2024

NATIONAL INSTITUTE OF TECHNOLOGY CALICUT, KERALA, INDIA - 673 601

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

2024

CERTIFICATE

Certified that this is a bonafide record of the project work titled

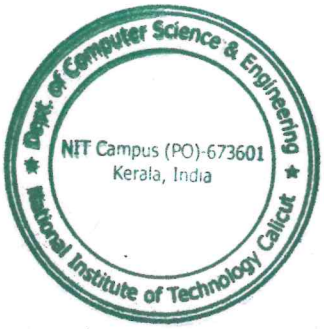
BINARY ISA PREDICTION IN FINE-GRAIN SCHEDULING IN HETEROGENEOUS ISA CMP

done by

Hatim Shakir
Md. Arif Raza Mansuri
Anagha M V

of eighth semester B. Tech in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering of the National Institute of Technology Calicut

Project Guide
Dr. Nirmal Kumar Boran
Assistant Professor



DECLARATION

I hereby declare that the project titled, **Binary ISA Prediction in Fine-grain Scheduling in Heterogeneous ISA CMP**, is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or any other institute of higher learning, except where due acknowledgement and reference has been made in the text.

Place : Calicut
Date : May 13, 2024

Signature : Hatim
Name : Hatim Shakir
Reg. No. : B200830CS

Signature: Arif Raza
Name: Md. Arif Raza Mansuri
Reg. No. : B200808CS

Signature:
Name: Anagha M V
Reg. No. : B200762CS

Abstract

Heterogeneous Instruction Set Architecture (ISA) Chip Multiprocessors (CMPs) have demonstrated substantial performance enhancements by leveraging the diversity of two or more ISAs, both at coarse-grain and fine-grain scheduling levels. The efficient resolution of the ISA scheduling problem at the function-level, with minimal migration time and hardware overhead, is imperative. This research explores multiple solutions with varying hardware requirements and energy consumption, aiming to address the evolving challenges of heterogeneous ISA CMPs.

ACKNOWLEDGEMENT

We express our deepest gratitude to Dr. Nirmal K. Boran, our esteemed supervisor, for his invaluable guidance, support, and mentorship throughout our pursuit of a B.Tech. degree. We also extend our thanks to Dr. Saidalavi Kalady and Dr. S. Sheerazuddin for their insightful feedback.

Additionally, we are indebted to Mr. Prakhar Diwan for his indispensable support, which significantly influenced the development of our experimental methods and the critical analysis of our results. We are also grateful to the Computer Science and Engineering Department for their significant contributions and assistance in providing an enriching academic environment.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the realm of computational technology advancement, efforts by computer architects and researchers persistently aim to accelerate computation. Among the various directions pursued to enhance performance for different applications, the heterogeneity of application programs has emerged as a notable avenue. Venkat et al. [4] demonstrated that different programs exhibit diverse affinities to different ISAs, including variations within the same ram. Subsequent research by Boran et al.[5] [1] transitioned from coarse-grained to finer-grained phases, proposing a function-level scheduling mechanism to execute each function on its most appropriate ISA.

Boran et al. [1] introduced a heuristic-based approach to scheduling function calls to predict ISAs, grouping functions and scheduling them collectively based on the assumption that the first function's affinity determines subsequent function calls in the group. However, this approach failed to realise performance gains approaching the theoretical limit proposed by Boran in the same work.

This research endeavours to explore more efficient scheduling approaches for function-wise scheduling between the 32-bit ARM architecture and the 64-bit X86 architecture. The study investigates two main ideas:

1. Exploiting irregularities in groups of function calls to facilitate finer

scheduling without significant migration overhead.

2. Modelling the ISA scheduling problem as a binary decision problem, analogous to branch prediction, to leverage existing hardware solutions for more accurate scheduling decisions.

This paper also aims to give a full picture of the different performance improvement strategies that can be used with heterogeneous ISA CMPs, including ways to deal with the difficulties of exploring heuristic-based solutions and lowering hardware overhead for prediction.

Chapter 2

Literature Survey

In the domain of computer architecture, optimising performance in heterogeneous ISA chip multiprocessors (CMPs) has garnered considerable attention. This section delves into existing literature and research projects, elucidating strategies for enhancing performance within the domain of heterogeneous ISA CMPs. By surveying relevant studies and innovations, insights into ongoing efforts to maximise computational potential in these architectures are provided.

2.1 Heterogeneous ISA CMPs

The heterogeneous ISA multi-core architecture, introduced in the works of Ashish Venkat [4], offers a versatile approach to improving the performance and energy efficiency of single-threaded programs. This approach addresses bottlenecks arising from frequency scaling saturation and power constraints by introducing diverse resources such as an increased register file size, reorder buffer, and instruction width. Significant improvements in single-thread performance and throughput on multi-programmed mixed workloads have been demonstrated, along with reductions in energy delay product.

To achieve high performance and energy efficiency, this paper confronts

two major challenges:

1. Design space exploration
2. Migration Overhead

Design space exploration: The possible design space of a heterogeneous ISA CMP encompasses a diverse set of ISAs and micro-architectural parameters, posing a challenging problem. Venkat et al.[4] proposed an exhaustive exploration of an architecture with limited, tractable options, optimising parameters such as code density, dynamic instruction count, and register pressure.

Migration Overhead: Migration between ISAs in a heterogeneous ISA environment involves expensive program state transformation [6] , presenting a significant challenge. Venkat et al. [4] built upon prior research to reduce migration overhead through several optimisations, including shared address space and dynamic binary translation.

2.1.1 Mitigating Migration Overhead

Addressing migration overhead in heterogeneous ISA multi-core (HeIMC) architectures involves several strategies aimed at reducing the costs associated with transitioning between different instruction set architectures (ISAs).One key approach is the implementation of a shared address space, although this poses challenges due to the architecture-specific memory layout of programs. Another technique involves generating a "fat binary," which comprises multiple target-specific code sections alongside a common target-independent data section. Challenges include managing the common data section and maintaining ISA-specific compilation optimizations.To overcome these challenges, a common intermediate representation is employed, enabling data representation and optimization across different ISAs. The LLVM compiler framework and Clang front end facilitate this process.

During migration, dynamic binary translation is used until reaching an equivalence point. Compiler-generated transforms convert program states from one ISA to another. These transforms are generated at compile time to reduce computational costs.

Venkat et al..[4] have made significant strides in enhancing single-threaded program performance, power efficiency, and the energy-delay product through the introduction of ISA (Instruction Set Architecture) heterogeneity. Their research divided programs into multiple phases, each comprising 100 million dynamic instructions, revealing that each phase exhibited an affinity toward a specific ISA. This affinity was influenced by factors like code density, dynamic instruction count, and register pressure. By executing each phase on its most affined ISA, rather than running all phases on a single ISA, substantial speed-ups were achieved. However, their focus primarily rested on the potential gains of Heterogeneous ISA Multicore (HeIMC) architectures, lacking attention on how to practically realize these benefits. Their work also assumed the existence of a prediction oracle, an impractical assumption in real-world computing environments.

2.2 Coarse-grain, Phase-wise Scheduling

Boran et al..[5] addressed the challenge of maximizing performance gains while minimizing potential performance degradation due to inaccurate migrations in HeIMC architectures. While the upper bound for performance increase reached an impressive 39% without considering migration overhead, inaccurately executed migrations could lead to a significant performance drop of up to 26%. To address this, Boran et al..[5] developed a prediction and scheduling mechanism for migrations, aiming to optimize benefits while avoiding detrimental impacts.

One limitation they identified in existing prediction models was their sole reliance on micro-architectural factors, with the decoupling of execution time

into two phases and the omission of parameters characterizing inter-ISA heterogeneity. Their solution involved a departure from traditional regression models, utilizing micro-architectural and ISA-specific parameters gathered from the executing core. These parameters enabled predictions of execution time for various core and ISA combinations, incorporating a linear model to directly predict execution time and eliminating the decoupling issue. Notably, except for parameters related to Instruction-Level Parallelism (ILP) and Memory-Level Parallelism (MLP), all necessary parameters could be obtained from hardware performance counters. ILP was assessed using the inverse of instructions in the issue stage, while MLP was managed through the Memory Side Hierarchical Replacement (MSHR) mechanism.

2.3 Fine-grain Function-wise Scheduling

Additionally, to effectively exploit program heterogeneity, Boran et al..[1] introduced a fine-grained function-wise scheduling technique. This approach dynamically scheduled every function to its most affined ISA, resulting in significant speed-ups compared to phase-wise scheduling and execution on a single ISA. Function-wise scheduling not only optimized performance but also reduced migration overhead by only transforming function arguments between ISAs, avoiding the need for complete stack transformations.The migration decision for a function is taken just before a function is called. Therefore, the function is executed on it's best affine ISA and the stack frame for the function is also formed in it's best affine ISA format. For that we need to transform only local parameters passed by the caller function to callee function. Rest of the memory i.e., global and heap, is common across both ISAs. Hence we do not need to handle heap, pointers and global memory. The memory map is consistent across both the ISAs as in [6].

To tackle the challenge of varying affinities within a phase, a heuristic-based scheduling algorithm was presented. Leveraging the observation that

program affinity remained consistent for around 20 consecutive calls, the algorithm used a sampling-based technique to predict ISAs for the next 20 function calls. By executing the program on the predicted ISA for 19 calls and switching on the 20^{th}, the algorithm refined its predictions based on the execution times of these calls. This proactive approach effectively adapted to changing program affinities and optimized dynamic execution.

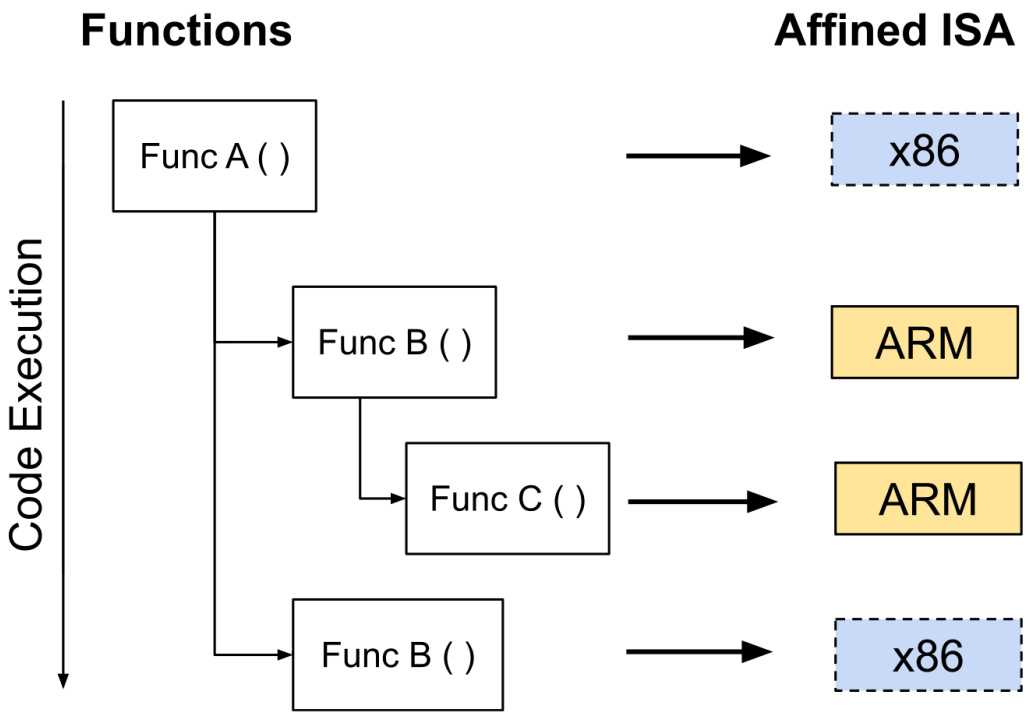


Figure 2.1: Dynamic Changing of Function Affinities (Source: Boran et al. [1])

2.4 Composite Cores

In addition to the approaches discussed earlier, there is a promising avenue for energy savings proposed by Andrew et. al [7] In their paper on 'Composite Core', they introduce an innovative architecture that addresses switching overheads in heterogeneous multicore systems. This architecture pairs 'big' and 'little' compute µEngines within a single core, enabling the simultaneous achievement of high performance and energy efficiency. By sharing a significant portion of the architectural state between these µEngines, they effectively reduce the switching overhead to nearly zero, facilitating fine-grained switching while maximizing the utilization of the 'little' µEngine without compromising overall performance. Their findings indicate that this approach can yield remarkable energy savings of up to 18% while maintaining performance losses as low as 5%.

Chapter 3

Problem Definition

The seminal works of Venkat et al. [4] and Boran et al. [5] have established a solid foundation for enhancing performance and conserving energy in heterogeneous Instruction Set Architecture (ISA) Chip Multiprocessors (CMPs). While Venkat et al. [4] shed light on the advantages of exploiting ISA diversity, Boran et al. [5] introduced novel scheduling techniques tailored for heterogeneous ISA CMPs, representing a significant advancement in the field.

In further works, Boran et. al [1] proposed fine-grain scheduling, at functional levels. Their work proposes to only migrate the next called function to the new ISA and return back once the called function is executed. This work was able to significantly reduce the migration overhead, by a factor of up-to 1000 and achieved novel potential gains of 22.8%. Despite these strides, a critical gap remained between the potential performance gains and actually achieved performance. While this heuristic-based scheduler showed promise in boosting performance, it also highlighted the challenge of unnecessary cross-ISA migrations, which often led to sub-optimal utilization of the heterogeneous architecture's capabilities.

To address this gap and optimize performance in heterogeneous ISA CMPs, this study sets out to explore more efficient scheduling methods, fo-

cusing on function-wise scheduling between the 32-bit *ARM* and the 64-bit *X*86 architectures. By investigating innovative strategies to minimize migration overhead and maximize performance gains, the research aims to narrow the disparity between theoretical performance limits and real-world implementation outcomes. The ultimate objective is to unlock the full potential of heterogeneous architectures, achieving significant performance enhancements while maintaining energy efficiency. Through rigorous exploration and creative solutions, this research endeavors to propel the field of heterogeneous ISA CMPs to new levels of computational efficiency and efficacy.

Chapter 4

Methodology

4.1 Core Configurations

Upon referring to the works of Venkat et. al [4] and Boran et. al [1], we have designed a 32-bit $ARMv7$ core and a 64-bit $X86-64$ core, with a two-level cache hierarchy. Similar ALU units are used for both cores for integer operations, but floating point support is not provided for $ARMv7$. A detailed comparison of both cores has been attached in Table 5.2.

4.2 Equivalence Points

As stated by Devyust et. al [6] in their works in cross-ISA migration, program can only be migrated at function call sites. Boran et. al [1] extend this approach in their works on Finer-grain scheduling to allow migrations at both call and return sites. They propose cross-ISA migration at only the function-level, and therefore *coming back* to the earlier ISA once a migrated function has been completely executed. Their methodology reduces the average migration overhead of 100μs [4] [6] to 0.02μs - 0.1μs [1]. We have used the same migration schema, with an average migration overhead of 0.05μs for each migration.

Table 4.1: Core Configurations

Designed Parameters	**Arm**	**x86**
Architectural Registers	32	16
Cache line size (bytes)	64	64
LSQ size	32	32
Fetch width	4	4
Decode, Dispatch, Issue		
Writeback, Commit Width	4	4
Instruction Queue entries	64	64
ROB entries	192	192
DCache, ICache size	32KB	32KB
L2 Cache size	256KB	256KB
SIMD Support	No	Yes

The works of Boran et. al [1] propose to migrate at each and every function call. However, a large number of calls run for 50-100 or less dynamic instructions (Figure 4.1). These functions cause the cores to migrate back and forth, creating an overhead that is comparable, or sometimes exceeds the potential gain. As a solution, we perform a static analysis of source code to create a list of *end functions*. These are the functions which do not call any other function, and are the end points of every call tree. We omit these end functions in our equivalence points.

4.3 Spike Switch

The Switch Algorithm, an extension of the Boran's heuristic approach [1], aims to enhance performance by introducing flexibility by providing a choice to switch ISA inside a window iteration on special case - *where a sudden increase is observed in ticks.*

Boran et. al [1] employ a heuristic-based approach, by bunching together

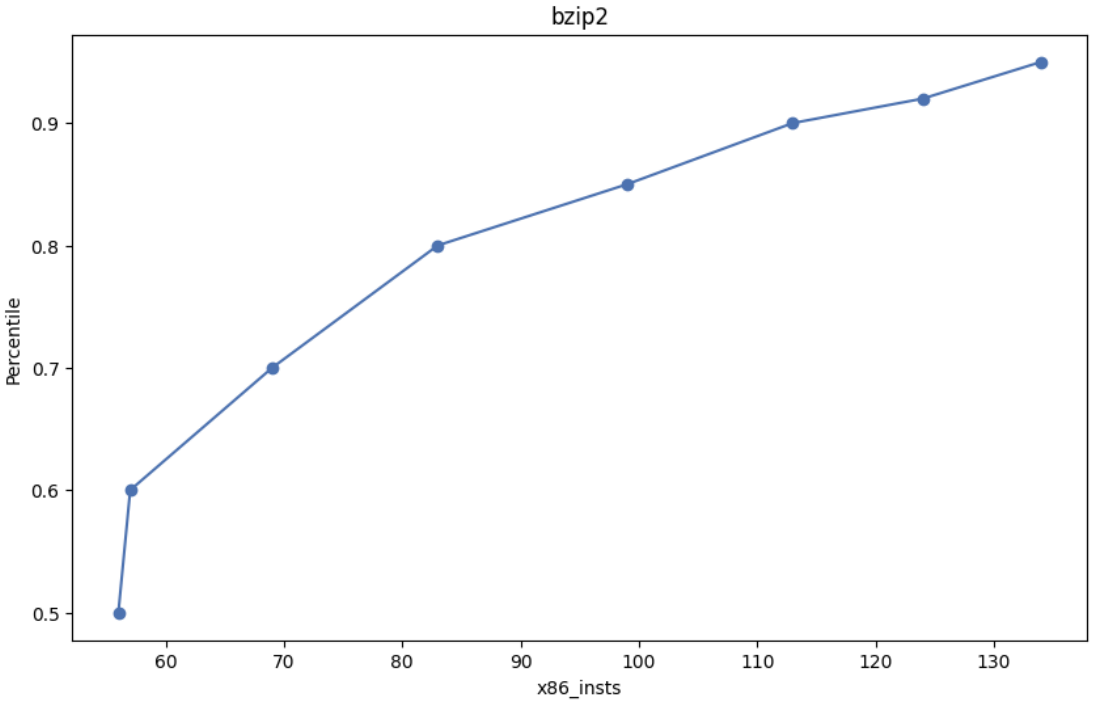


Figure 4.1: `nab` Dynamic Instruction Count for `bzip2` [2]

20 iterations of every particular function. This bunch of 20 iterations of each function is subsequently referred as a *window*. Boran et. al [1] determine the affinity of every *window* by comparing the 19^{th} and 20^{th} function call time. By this approach they forcefully migrate to the other ISA for the 20^{th} function call and thus, by comparing the time of two different ISAs the affinity of the next window is decided.

Boran's heuristic-based approach [1] is very sensitive to inaccurate 19^{th} and 20^{th} affinities, where the 20^{th} can be affined to ISA A but the subsequent window is affined to ISA B. In such a case, all potential gains are lost for all 20 function-call iterations. Furthermore, a sudden increase in time taken to execute a function. called a *spike*, on a single ISA devastatingly impacts the performance of Boran's heuristics. Boran's heuristics are not adaptive to

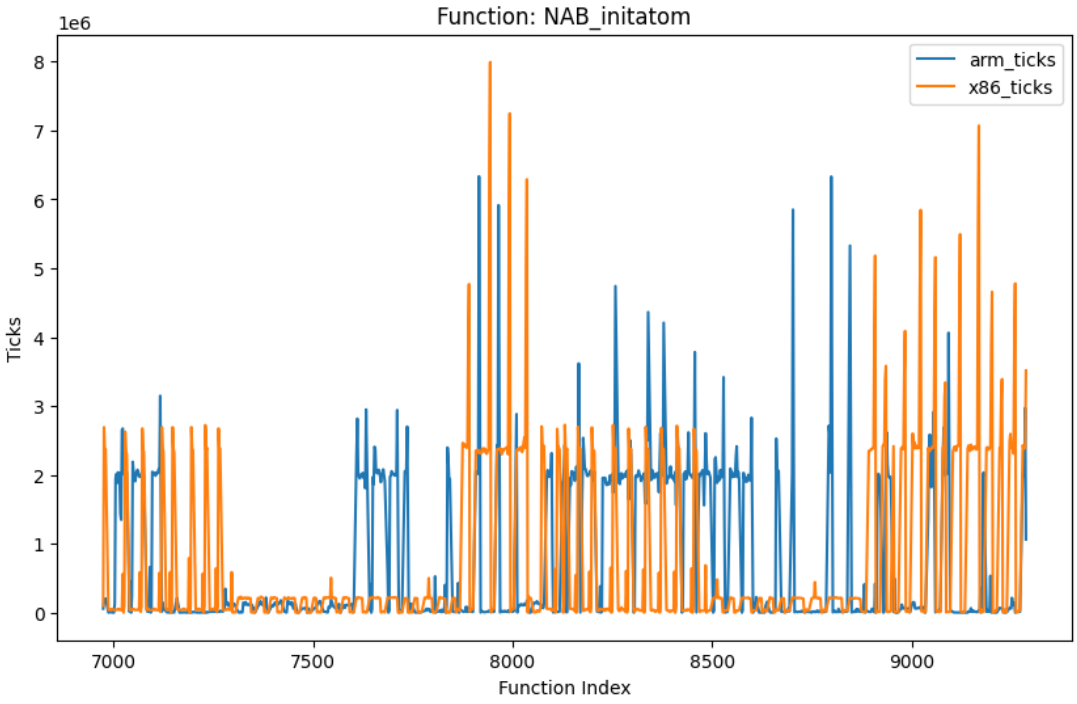


Figure 4.2: `nab` Function call index vs Ticks for `NAB_initatom` [2]

such a scenario.

In an attempt to address both issues, we have implemented a **2-bit Binary Affinity Predictor, with zero-tolerance to spikes**. This approach handles both shortcomings of Boran's heuristics in the following manner:

1. **19th/20th Inaccuracies:** Our approach implements a 2-bit predictor logic to predict the ISA between $ARMv7$ and $X86-64$. Unlike Boran's heuristic to change ISA affinity after observing a different ISA affinity from head and tail times, our predictor only changes window affinity after two consecutive changes in window ISA affinity. This is illustrated in Figure 4.3.

2. **Resilience to Spikes:** To prevent loss of performance gains due to

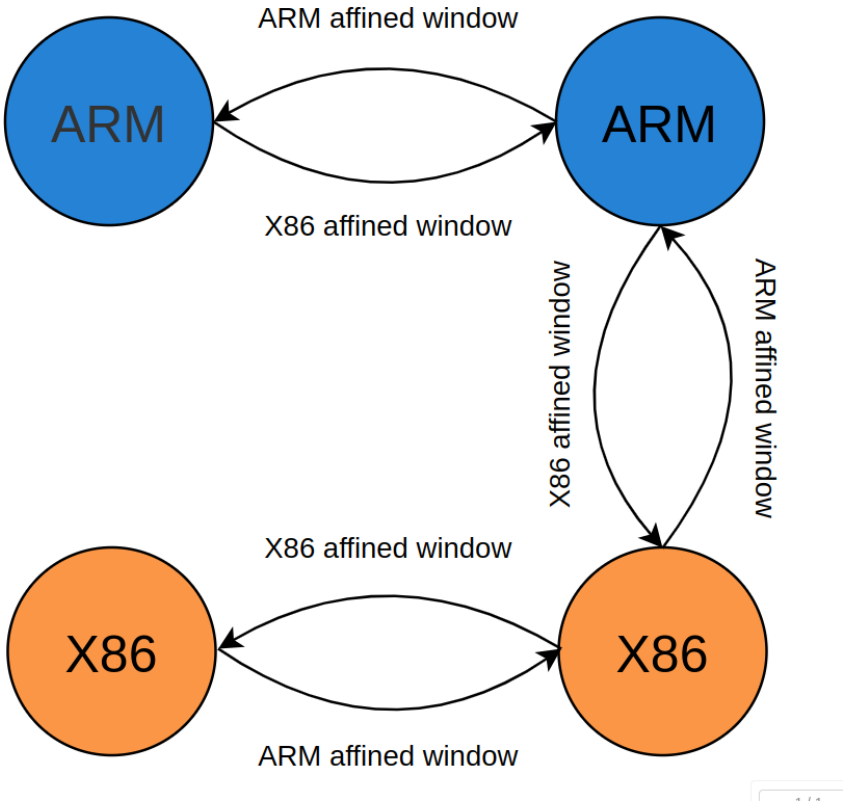


Figure 4.3: Two-bit prediction logic

spikes, we immediately migrate the next call to other ISA. This renders the predictor dynamic in response to drastically changing affinity. The predictor keeps tracks of ticks and instructions of every function's last iteration. Whenever the current ticks and/or dynamic instructions exceed 50 times the values of last call, the current call is considered a `spike` and ISA affinity is immediately changed for the subsequent calls.

4.4 (m, n) Correlating Predictor

The works of Venkat et al. [4] propose a phase-wise program affinity, where an entire phase of 100 million instructions is performance optimised to a single ISA, which indicates a global average of each individual function-wise affinities. Thus, predicting the affinity of a function call at the finer level is an aggregate decision of the function's affinity at the *local* level, and the program's affinity at the *global* level. Furthermore, the decision to choose the ISA for every function call also depends upon the decision taken for some of the last function calls. A similar problem is also prevalent in branch prediction, where the prediction of every branch is also influenced by other branches taken or not taken, as well as history of the current branch taken or not taken. In addition, since most modern branch predictors have a very high accuracy, we modelled the affinity prediction problem as a branch prediction problem.

We model the question of what ISA should the **next function call** take as a binary decision between $ARMv7$ and $X86-64$, similar to *take* and *not-take* in Branch Predictors (BPs). Thus, unlike Boran's aggregate *window-based affinity*, we are able to predict the affinity at a much finer level. Since it is not possible to run the program on both ISAs to get the *actual* values of affinities, we predict the actual affinity. We use a Logistic Regression Classifier to predict the *actual* value of function affinity that is used to train the Correlating Affinity Predictor. The classifier takes values of micro-architectural parameters outlined by Boran et al. and outputs a prediction of the *actual* affinity of the window, with an accuracy of 74.8%.

4.5 Static ISA Prediction

An analysis of individual functions and their affinities shows that most functions have a strong affinity to be either ARM-affined or X86-affined, as high

Figure 4.4: (2, 2) Correlating Predictor [3]

as $90 - 95\%$ affined to either ISA. This provides a strong intuition to determine the correct affined ISA ahead of time and generate a static prediction. This approach directly increases the accuracy of prediction to the proportion of major affined ISA. A static prediction not only achieves an accuracy proportional to the affinity of each function, but also reduces the migration overhead.

To approximate the affinity of each function, we execute the first 60 function calls of the function in an approach proposed by Boran [1]. However, to optimize the execution, we implement a switch-based approach, as discussed in Section 4.3. Finally, we calculate average ticks for functions executed on

$ARMv7$ and $X86-64$ and assign static to the better performing ISA.

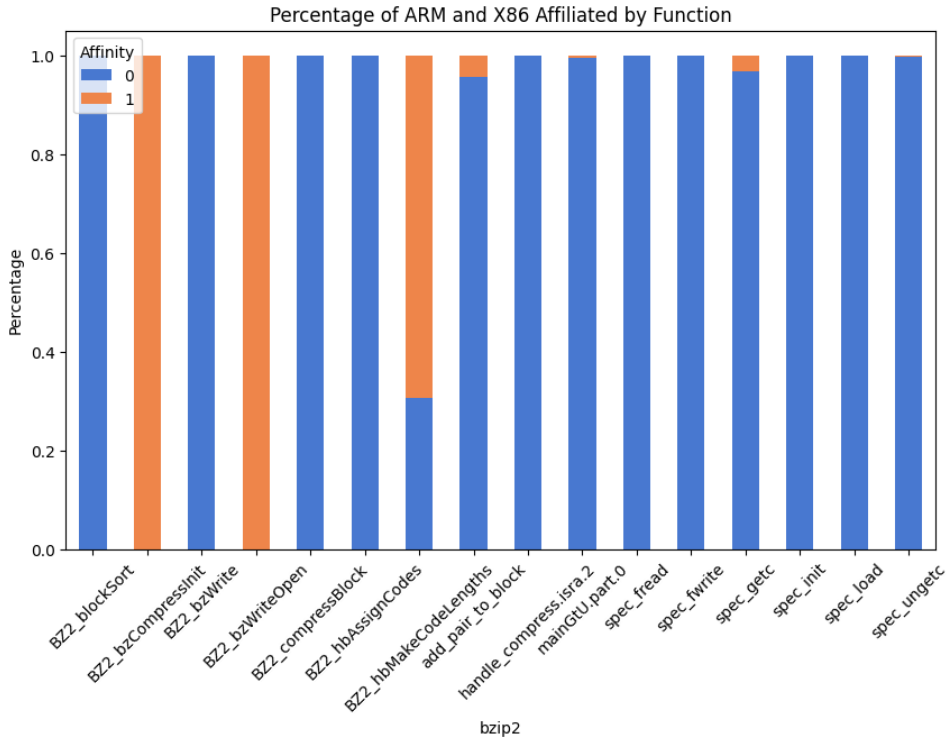


Figure 4.5: Function-wise Affinities for `bzip2` [2]

Chapter 5

Results

5.1 Benchmarks

For evaluating our solutions, and comparing with existing state-of-the-art, we use $SPEC2017$ and $SPEC2006$ benchmark suits, with a diverse set of integer and floating point benchmarks. We have made use of Gem5 [2] simulator to simulate benchmarks on core configurations as mentioned in Table 5.2. Each benchmark has been simulated for, whichever occurs first:

1. 5 billion dynamic instructions are simulated
2. 1 million equivalence points reached
3. Benchmark terminated

We have simulated a total of 12 benchmarks total, from both $SPEC2017$ and $SPEC2006$ benchmark suites. A detailed description of these benchmarks is provided in Table 5.1

5.1.1 Migration Overhead

For the migration overhead we have referred Boran et al. [1] paper and have taken 100μs for the coarse-grain algorithm and 0.05μs for the fine grain

Table 5.1: Benchmark Descriptions

Benchmark	Suite	Description
bzip2	$SPEC2006$	Compression
deepsjeng	$SPEC2017$	Artificial Intelligence: alpha-beta tree search (Chess)
gobmk	$SPEC2006$	Artificial Intelligence: Go
lbm	$SPEC2017$	Fluid Dynamics
leela	$SPEC2017$	Artificial Intelligence: Monte Carlo tree search (Go)
libquantum	$SPEC2006$	Physics / Quantum Computing
mcf	$SPEC2017$	Route planning
mcf-06	$SPEC2006$	Combinatorial Optimization
milc	$SPEC2017$	Physics / Quantum Chromodynamics
nab	$SPEC2017$	Molecular dynamics
namd-06	$SPEC2006$	Biology / Molecular Dynamics
namd	$SPEC2017$	Molecular dynamics

scheduling algorithm. Migration overhead for coarse-grain approach is on higher side because only few register values have to be transformed for function level scheduling as compared to whole stack transformations required in the coarse-grain approach.

5.2 Performance Results

We have taken the coarse-grain, 10 million phase-wise model proposed by Boran el at.[5] as the base for comparing all of our approaches. The oracle achieved a 30.2% speed-up compared to the base. This speed-up has increased from the oracle results proposed by Boran [1], and the increase can be attributed to the inclusion of several new benchmarks from $SPEC2017$ suite and their different behaviour than $SPEC2006$ suite.

5.2.1 Spike-Switch

So our spike-switch algorithm has achieved a 13.8% speed-up as compared the the phase wise model proposed by Boran el at.[5]. This approach failed to full-fill the expectation because of the presence of lot of single spikes occurrences in the benchmark due to which we switch the ISA in between even though the rest of the function was affined to the previous ISA. This approach will work significantly better for the benchmarks in which the spikes occur in bunches.

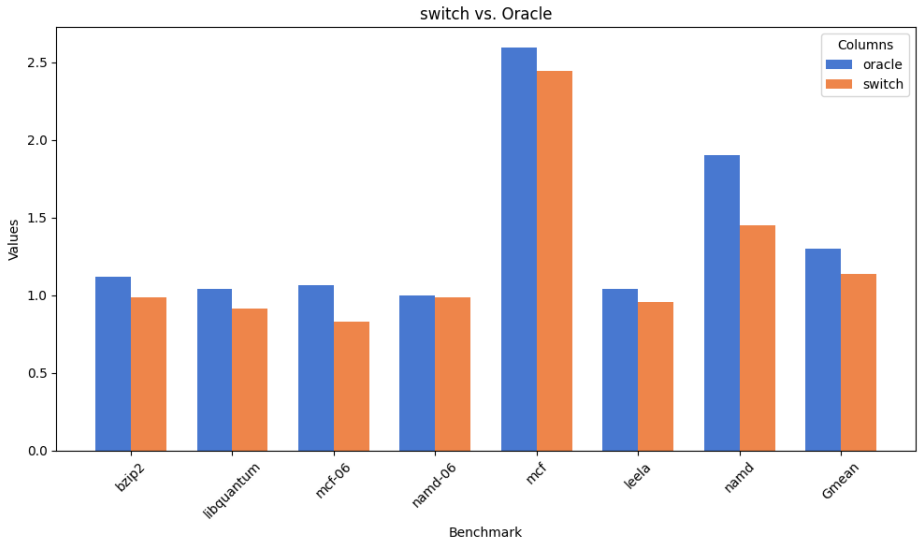


Figure 5.1: speed-up of **Spike-Switch** heuristic with respect to Het-ISA CMP

5.2.2 Correlating Predictor

We have used a (m, n) *Correlating Predictor* to predict the affinity of *function windows*. We have experimented with multiple values for *number of bits*, *number of history calls* and different *window sizes*, which are outlined in

Table 5.2.2. We have used a left-shift register to store `arm` or `x86` denoting the last **x** window affinities, and each **n**-bit predictor begins with `x86`.

Parameter	**Values**
`n_bits`	1, 2
`last_x`	2, 4, 5, 8, 10
`win_size`	10, 20, 50

Table 5.2: Correlating Predictor Parameters

By using a model with 84.7% accuracy, and no warmup for the Correlating Predictor, it achieves a mean speed-up of 20% in comparision to 16% mean speed-up of Boran's heuristics. (Figure 5.2

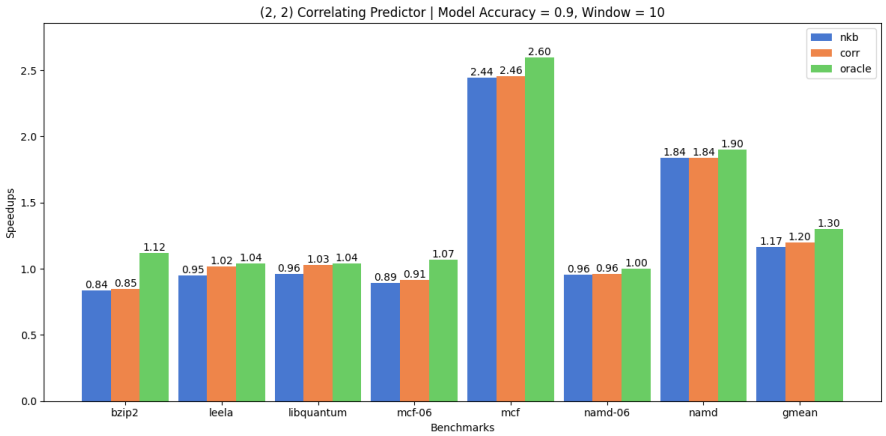


Figure 5.2: speed-up of **(2, 2) Correlating Predictor** and Boran's heuristics with respect to Het-ISA CMP

5.2.3 Static-Isa

The proposed static based scheduling approach has achieved a 15% speed-up. However the oracle of this approach has achieved a significant speed-up of 28.4% which is very much close to the 30.2% speed-up as showed by the

function level scheduling oracle. One major reason of the lack of performance of this approach as compared to its oracle speed-up is that presence of the functions which occur once or twice and have a significant difference between their execution time on ARM and $X86$. Since we start, by default, on ARM, we fail to capatalize the potential gains on functions which are heavily $X86$ affined.

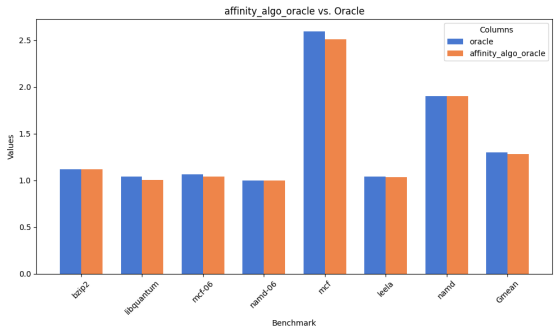


Figure 5.3: speed-up of **Static ISA Affinity Oracle** with respect to Het-ISA CMP

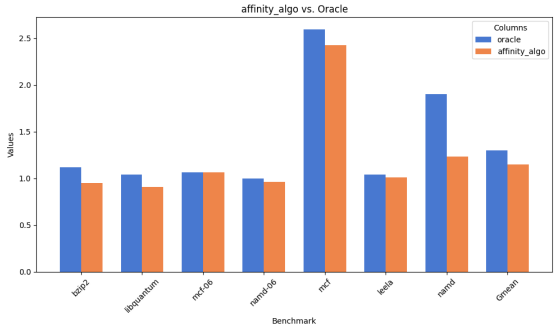


Figure 5.4: speed-up of **Static ISA Affinity** heuristic with respect to Het-ISA CMP

Chapter 6

Conclusion and Future work

We have achieved a speed-up of 13.8% for *Switch-Spike* heuristic with *window size* of 20, 15% for *Static ISA scheduling* by starting with *ARM*, and 20% for *(2, 2) Correlating predictor*, with *window size* of 10. We also achieved 28.4% potential speed-up for `Static ISA Scheduling` approach. Further research needs to be done on improving the heuristics needed to determine the correct-affined ISA for each function. In addition, static analysis of functions could be used as potential indicators for their run-time affinities.

References

[1] N. K. Boran, S. Rathore, M. Udeshi, and V. Singh, "Fine-grained scheduling in heterogeneous-isa architectures," *IEEE Computer Architecture Letters*, vol. 20, no. 1, pp. 9–12, 2020.

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, p. 1–7, aug 2011.

[3] H. Mahmood, "Fpga configuration of an alloyed correlated branch predictor used with risc processor for educational purposes," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, p. 265, 02 2021.

[4] A. Venkat and D. M. Tullsen, "Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 121–132, 2014.

[5] N. K. Boran, D. K. Yadav, and R. Iyer, "Performance modelling and dynamic scheduling on heterogeneous-isa multi-core architectures," in *VLSI Design and Test: 23rd International Symposium, VDAT 2019, Indore, India, July 4–6, 2019, Revised Selected Papers 23*, pp. 702–715, Springer, 2019.

[6] M. DeVuyst, A. Venkat, and D. M. Tullsen, "Execution migration in a heterogeneous-isa chip multiprocessor," in *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, pp. 261–272, 2012.

[7] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. Dreslinski, T. F. Wenisch, and S. Mahlke, "Composite cores: Pushing heterogeneity into a core," *IEEE/ACM International Symposium on Microarchitecture*, pp. 317–328, 2012.

[8] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," *Computer*, vol. 38, no. 11, pp. 32–38, 2005.

[9] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-isa heterogeneous multi-core architectures: The potential for processor power reduction," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pp. 81–92, IEEE, 2003.

[10] S. Mittal, "A survey of techniques for dynamic branch prediction," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 1, p. e4666, 2019.