ABSTRACT

HASHEMI, HASHEM. Core-Selectable Chip Multiprocessor Design. (Under the direction of Dr. Eric Rotenberg).

Hitherto, in the design of microprocessors, a major barrier to greater single-thread performance has been the inevitable limitation of a single microarchitectural solution being used to execute all types of applications. However, with technology feature-sizes shrinking to the point that it is now possible to package multiple processing cores in a single microprocessor, an opening to this barrier is becoming visible. Although this increase in the number of cores has primarily been viewed as a means to provide thread-level and task-level parallelism, it also provides the opportunity to improve single-thread performance through the incorporation of diversely microarchitected processing cores.

What is more, the aggregate die area taken up by the cores in multicore microprocessors, or Chip Multiprocessors (CMPs), is shrinking in proportion to that of the non-core portion. What this essentially means is that it is becoming cheaper to stamp-out more cores into the design of a CMP – as long as the cores need not be fully interconnected. Thus, a promising design approach, that forms the central proposal of this thesis, is to incorporate more than one core at each port to the interconnect, gear the microarchitectural design of these cores to distinct workload characteristics, and provide the ability to dynamically select which core to actively employ at each port depending on the immediate application. This solution, which will be referred to as *core-selectability*, retains the design symmetry that is essential to scalable multithreaded performance (but lost when unique core designs are spread out across

the system), while providing the microarchitectural diversity that is essential to single-thread and multiprogrammed performance.

With this approach, the more customized the different core designs are to the behavior of their constituent workload behavior, the more potential there is for performance gain. Thus, accurately studying the potential of this design solution requires accurate understanding of (a) how the microarchitectural design space should be explored, (b) how the workload space should be split up between different core designs, (c) how circuit-level propagation delays should be modeled and accounted for, (d) how overall system performance should be evaluated, and (e) how variability in application phase behavior can be dealt with. These are the issues that are addressed in this thesis.

Among other results, it is revealed that in splitting up the workload space between different core designs, doing so based on the Euclidian distance of applications in their raw workload behavior can lead to suboptimal design choices. It is shown that, in the evaluation of the overall performance of CMP designs, accounting for task arrival behavior can be critical to accuracy. It is shown that core-selectability has the potential to provide notable overall performance benefit to both multi-programmed workloads with different task arrival patterns and multi-threaded workloads. Moreover, it is shown that by extending core-selectability to exploit fine-grain changes in application behavior, with a technique called *architectural contesting*, it is possible to achieve even greater single-thread performance enhancement.

Core-Selectable Chip Multiprocessor Design

by
Hashem Hashemi

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Computer Engineering

Raleigh, North Carolina

2011

APPROVED BY:

_____	_____
Dr. Eric Rotenberg Committee Chair	Dr. Gregory T. Byrd
_____	_____
Dr. James M. Tuck	Dr. Vincent W. Freeh

DEDICATION

To Tooran Khanoom

BIOGRAPHY

Hashem Hashemi was born in Tehran, Iran, on the 22nd of December 1978. He graduated from Shahid Beheshti High School in 1997. Subsequently, he attended Azad University at Central Tehran to pursue a Bachelor's Degree in Computer Engineering (Hardware), graduating in 2002. During the last year of his undergraduate studies he worked part-time at the Iranian Research Organization for Science and Technology (IROST) on the development of an industrial fermentation system. In the same year, he passed the national entrance exam and was admitted to Sharif University of Technology in Tehran to pursue a Master's Degree in Computer Engineering. During his MS studies, under the advisory of Dr. Hamid Sarbazi-Azad and with funding from the Institute for Research in Fundamental Science, he worked on performance evaluation and modeling of various multiprocessor interconnection network architectures. In 2005, he was admitted to North Carolina State University, in Raleigh, to pursue a PhD in Computer Engineering, and granted a research assistantship under the advisory of Dr. Eric Rotenberg. Since then he has been working on a number of research topics surrounding the employment of diversely microarchitected processing cores in chip multiprocessors.

ACKNOWLEDGMENTS

For "Is" and "Is-not" though with Rule and Line,
And "Up-and-down" without, I could define,
I yet in all I only cared to know,
Was never deep in anything but – Wine.

Rubaiyat of Omar Khayyam

I have two large groups of people to thank for where I woozily find myself today; those that introduced me to the Wine and those that kept the vase full.

Azad University, with its rundown buildings and underpaid staff, was a treasure-trove of individuals that had given up on everything else, and kept coming back purely for the joy of teaching and learning a little more. It was there, one autumn day, in the course on Computer Architecture that it seemed everything came together for me to grasp a fascinating concept: how a fetch-execute engine works. It was an incredible thrill that got me hooked on Computer Architecture until this day. For that I am thankful of Mr. Taherbaneh, my undergraduate thesis advisor, and for the delightful project he got me working on.

Dr. Sarbazi-Azad, my MS advisor at Sharif University, with his insatiable interest in finding new ideas, and machine-like ability to prepare studies for publication, was instrumental in my dazed crossover from the undergraduate world of grasping concepts, to the graduate world of envisioning them. For granting me a position through which I was both able to publish papers and earn enough to apply for PhD admission, I am forever indebted,

Dr. Rotenberg, my PhD advisor at NC State University, being from Wisconsin, is more of a beer person. Yet he has kept the Wine flowing with his incredible ability to delineate complex concepts, and envision the proper methodology to evaluate ideas. For granting me the experience of working with him, not to mention all the years of research funding, I am eternally grateful.

Special gratitude goes to my father. I now realize that I would not have been able to complete my undergraduate studies without his evenhanded support (and that over-supportiveness would have killed the sense of rebellion that, confessedly, kept me going). Much appreciation also goes to the North Carolinian and Iranian tax payers for their generosity in subsidizing my graduate studies. I am also appreciative of my PhD committee members, Dr. Byrd, Dr. Tuck and Dr. Freeh for their valuable feedback.

There are many others I owe gratitude to, of whom I can only mention a few here. Muawya Al-Otoom for being a great companion throughout my PhD years, and the always gregarious Niket Choudhary. Mark Dechene, Elliot Forbes, Sandeep Navada, and Salil Wadhavkar for the countless hours of brainstorming and banter. Mustafa Rezazad and Mohammad-Reza Husseiny for making my years of MS study fruitful and fun – the memories of which still bring on chuckles. And my high-school friends, Babak Damadi and Hussein Feghhi, for it was they who first showed me that learning is cool.

To all the above, I also owe a sober apology, for despite trying, I have failed to produce anything remotely commensurate in value to what they bestowed me. What follows is all I have to show for a botched attempt at that.

This research was supported with funding from the NSF (grant No. CCF-0811707), Intel, and IBM. Any opinions, findings, and conclusions or recommendations expressed herein are those of the author and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: Introduction

A major hurdle to better microprocessor performance is the sheer generality of application behavior for which the centralized microarchitectural units necessary for the extraction of instruction-level parallelism (such as the issue-queue, load/store queue and reorder buffer) must be designed for. If it were possible to dynamically change these microarchitectural units to suit the characteristics of the application at hand, notable instruction-level performance could be gained. The inherent criticality of these units, however, renders practically infeasible the option of designing them to be dynamically changeable – due to the circuit-level overheads it would entail.

A more feasible solution that has been proposed in prior art is the incorporation of different fixed processing cores, each designed for different types of application behavior, with the ability to somehow route applications to cores based on suitability. This approach, however, in its own, is not a scalable solution. Adding to the diversity of the core designs without increasing the number of cores results in degradation of the system's effectiveness in executing multithreaded applications. And increasing the number of cores requires an increase in the provisioned cache and interconnection resources (to accommodate all of the cores) – resulting in an outweighing increase in complexity and degradation of efficiency. Moreover, from a high-level standpoint, this solution results in a design that is irregular, asymmetric and non-homogeneous.

Meanwhile, the actual processing cores in Chip Multiprocessors (CMPs) have been consuming progressively smaller portions of the total physical layout. This trend is a result of

a number of factors: (i) merely scaling the size of the ILP-extracting units does not necessarily improve general performance, (ii) increasing the amount of cache and interconnection bandwidth can at the very least improve data throughput, and (iii) increasing in the number of cores, in a well-provisioned manner, requires accompanying increase in the cache capacity and interconnection bandwidth. In layman's terms, giving more die area to the cores either results in no performance benefit (when used to scale microarchitectural structures) or it necessitates more die area to also be given to the non-core portion of the design (when used to increase the number for cores).

Evidence of this trend can be observed in Figure 1-1, which illustrates the ratio of die area consumed by the actual processing cores in chip multiprocessors over the past years. A

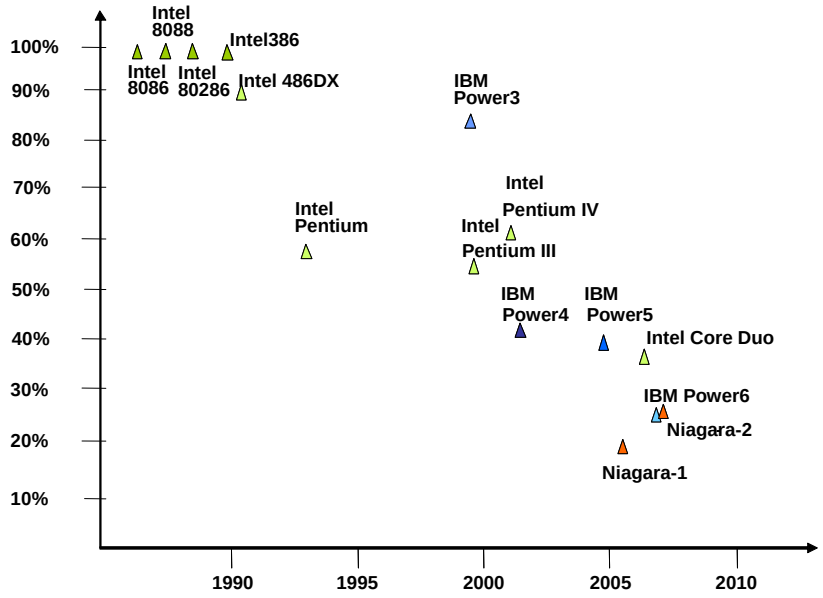


Figure 1-1: Percentage die area consumed by processing cores in commercial microprocessors over the years[1].

[1] These estimates are slightly subjective as they were extracted through observation of die photographs of different processors, and visually distinguishing the actual cores from the non-core portion (including all levels of cache).

separate recent study [12] also forecasts this shrinking trend to continue.

The goal of this thesis is to study how this trend in the aggregate die area consumed by cores might open an avenue to a different form of instruction-level performance enhancement; by enabling the availability of alternative core designs in the system without complicating the non-core portion of the design. Specifically, a solution is proposed in which each core in the CMP is replaced with a cluster of multiple differently designed cores, forming what will be referred to as a *node*, with the option to dynamically select which core in each node to actively employ (depending on the application at hand). The purpose of these different cores is to provide microarchitectural diversity, rather than concurrency. Thus, only one core in each node need be actively employed at any time.

This solution allows for the cores to share the complex resources that interconnect nodes together in the CMP, while maintaining the original provisioning of these resources. In other words, the cache and interconnect need only be provisioned for the number of nodes, rather than the number of cores. A chip multiprocessor with such a design will be referred to here as a core-selectable design [38], as the fundamental benefit this design solution provides is the ability to *select* the microarchitectural design to employ.

This approach is a scalable solution to exploiting the increase in availability of transistors to enhance performance without overly increasing design complexity, verification effort or power consumption. It allows for the microarchitecture design to be partitioned and focused on specific types of workload behavior, rather than pack everything into a single design solution that attempts to eke out performance with complex all-inclusive design tricks.

As illustrated in Figure 1-2, core-selectability provides the potential for the system to dynamically transform between:

- Different homogeneous configurations (each employing a different core design)
- Different heterogeneous configurations (each employing a different combination of core designs)

Employing a suitable core design in the homogeneous configuration enables better multi-threaded performance. Employing a suitable combination of core designs in the heterogeneous configuration enables greater throughput to multiprogrammed workloads [87] and better performance to critical-section intensive multithreaded workloads [62]. As illustrated in Figure 1-3, this is all achieved while retaining overall design symmetry.

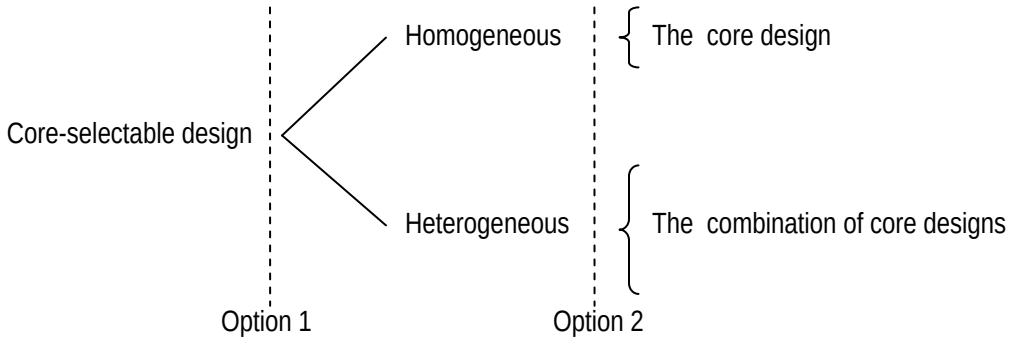


Figure 1-2: The configuration options in a Core-selectable system.

The value of this paradigm can be better understood in contemplation of the following layman's question: *What has been preventing microprocessor manufacturers from employing a larger number of processing cores?* Each additional core consumes minimal die area (compared to the other circuitry that has recently been brought on-chip, e.g. the memory

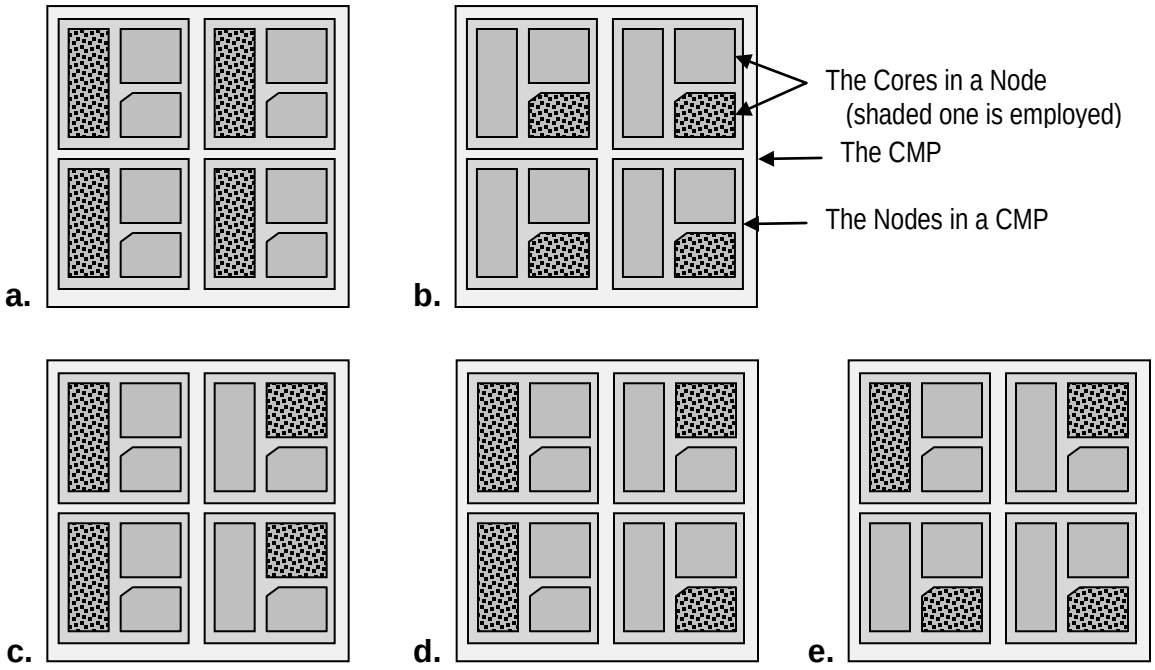


Figure 1-3: Examples of how the cores in a Core-selectable CMP can be employed. (a) (b): Homogeneous configurations. (c) (d) (e): Heterogeneous configurations.

controller). Moreover, merely replicating the cores themselves consumes no extra design or verification effort. Thus, the answer must lie in the power budget, and the complexity fully-interconnected cores introduce to the interconnection network and cache hierarchy.

As illustrated in Figure 1-4, in conventional chip multiprocessor design, increasing the number of cores in the system necessitates more interconnection bandwidth and overall cache capacity. This extra circuitry increases core-to-core propagation delay, the system's power consumption and manufacturing cost, in addition to design and verification effort. In general, the major problem with on-chip networks is that they simply do not scale very well.

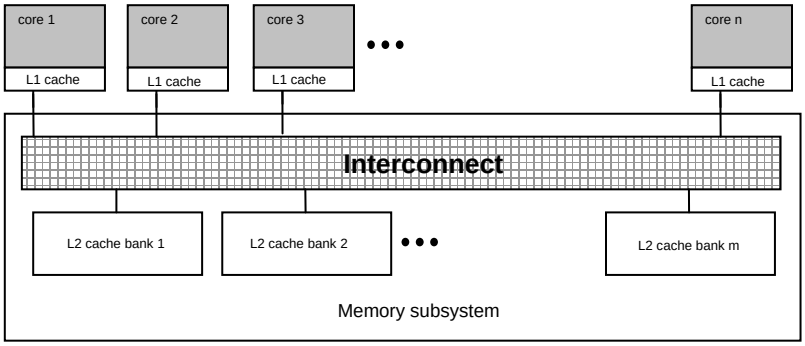


Figure 1-4: Complexity of the interconnection in a conventional CMP design.

Nevertheless, the issue of concern here is not *cost* per se (as it is natural for greater performance to entail higher cost), but rather, it is an issue of *wastefulness*. It is wasteful to dedicate further resources (e.g. power budget, design effort) to any portion of the design before that portion has been fully exploited. However, a large portion of workloads tend to underutilize the cache and interconnect. The major reason for this is that workloads that place high demand on these resources are more dominant in influencing what entails a well provisioned design – as insufficient provisioning for them dramatically degrades *their* performance (and consequently overall performance).

It is important to recognize that adding cores to a system without increasing the provisioned interconnection resources can alternatively be achieved by splitting the existing resources among the cores (i.e. splitting the channel bit-width in crossbar-based interconnect, and the bisection-width in tile-based designs). However, if doing so were to yield better overall performance, it would indicate that the interconnect portion of the original design was over-provisioned to begin with. More importantly, the overhead of this naive form of increasing

concurrency permanently impacts system performance – such that applications that are inherently less scalable permanently observe performance degradation.

Therefore, it is the confluence of a number of factors that motivates core-selectability. Figure 1-5 summarizes how these factors can be viewed as leading to each other. The fact that the cache and interconnect need to be provisioned for the number of simultaneously-operable cores (the *provisioning* factor), coupled with all cores currently being of the same design (the *one-size-fits-all* factor), leads to the fact that the cores are in aggregate consuming a diminishing portion of die area (the *shrinking* factor). This, coupled with the fact that there is single-thread benefit in employing diversely designed cores for different applications (the *diversity-is-beneficial* factor), leads to the benefit in the incorporation of diversely designed cores such that they need not all be simultaneously operable (the *presence-is-beneficial* factor). This, coupled with the fact that for many applications the actual performance bottleneck is not the cache or interconnect (the *underutilization* factor), and that port sharing has been shown to be a feasible design option (the *port-sharing* factor), lead to the solution of incorporating similar clusters (or nodes) of differently designed cores such that all the cores in a cluster share a single port to the cache and interconnect (*core-selectability*).

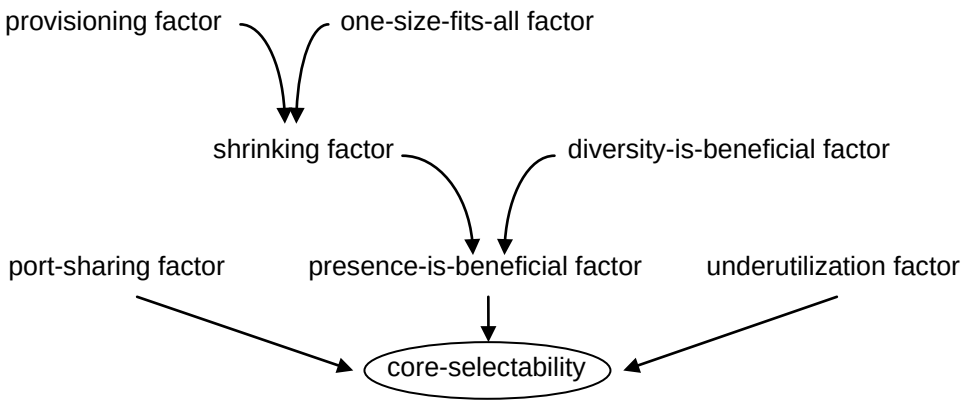


Figure 1-5: The confluence of factors that motivate core-selectable design.

Figure 1-6 illustrates the basic schematic of a two-way core-selectable design. Within each node, the active core takes over the port to the L1 data cache. Note that this figure only shows the multiplexing of the address signals to the L1 cache, not how the data paths are directed to both cores.

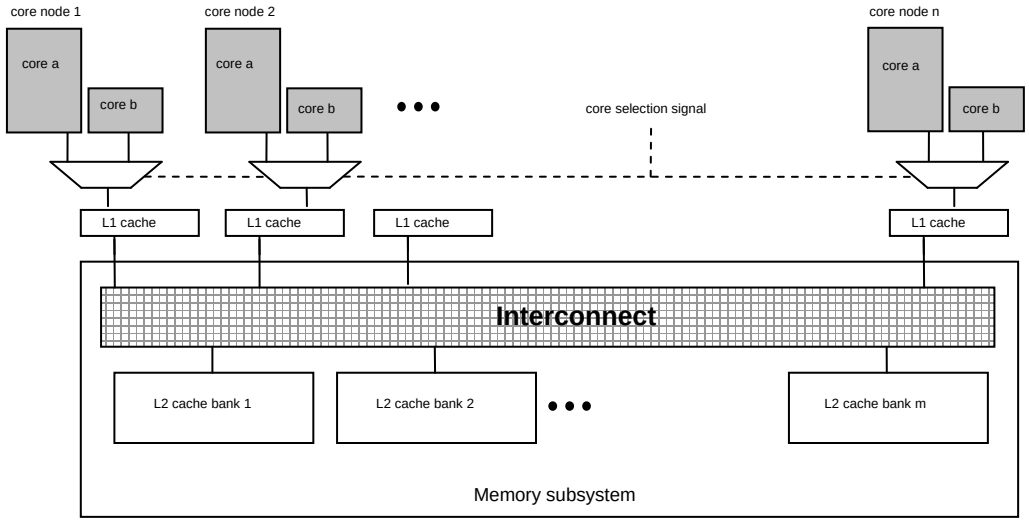


Figure 1-6: Complexity of the interconnection in a core-selectable CMP design.

The true value of core-selectability lies in its simplicity and the absence of microarchitectural invasiveness. In fact, the microarchitectural adjustments needed for implementing this design solution lie only in the back-end and front-end of the core designs. This simplicity is vital to minimizing design and verification effort.

The ability to employ different core designs opens the door to a much broader overall system design space, and the potential for considerable performance gain and improvement in power efficiency. However, a number of questions surrounding this design solution need to be addressed. For instance, there are the details of exactly how inactive cores can be awoken and execution transferred to them. There are issues that need to be addressed with regards to which portions of the different core designs should be shared among the cores, and which should be kept private. There are questions surrounding the design space exploration of such a system, and how the workload space should be split up among the cores. There are also questions regarding the effect of fine-grain change in application behavior, and the effect of the pattern with which applications tend to arrive at the system. The follow chapters address these issues.

The next chapter provides an overview of related work. Chapter 3 addresses how the workload space can be split up between differently designed cores in order to enhance overall performance. Moreover, it provides insight into how the microarchitectural design space of the processing cores should be explored, and the importance of accurately accounting for circuit-level details in this exploration.

With the evaluation background setup, an in-depth study of various implications of core-selectability is presented in Chapter 4. Results are presented that show that, even when

limiting the diversity between core designs to only the sizing of microarchitectural structures, core-selectability has the potential to provide notable performance enhancement to scalable multithreaded applications, without the need to extract thread-level concurrency. In addition, it is shown that core-selectability can provide greater throughput to multiprogrammed workloads by providing the potential for the system to transform into a heterogeneous design. Chapter 5 addresses the opportunity for performance enhancement presented by frequent changes in application behavior, and how a core-selectable system is inherently suited to exploit this. An implementation solution is proposed that enables speedy transfer of effective execution between the differently designed cores in a node. This solution will be referred to as Architectural Contesting. Chapter 6 provides a summary, and outlines future directions for the continuation of this work.

CHAPTER 2: Background and Related Work

Prior work has shown that there is significant performance benefit in employing customized core designs for different workload behavior [88] [87]. Nevertheless, when limited to a single core design, the best solution is one that provides reasonable performance across a wide range of workload behavior. Thus, the different units of a general-purpose processing core need to be designed in anticipation of typical workload behavior. Such a system will perform suboptimally when the actual workload being executed on the system displays atypical behavior – with undersized structures degrading IPC and oversized structures wasting propagation time.

The characteristics of the employed technology can also impact the best tradeoffs in the design of a processing core. For instance, extracting greater parallelism requires more complex logic (and longer propagation delays), which either results in deeper pipelining or an increased clock period. While increasing the clock period can directly degrade performance, deeper pipelining can adversely impact parallelism by impacting the cycle delay between the wakeup of back-to-back dependent instructions.

Moreover, intricate circuit-level details can dramatically sway the best design tradeoffs for a given workload behavior. For instance, the unified clock period intertwines the different microarchitectural design units. Thus, in a high-performance design, the scaling of any unit must either result in change in the pipeline depth of that unit or it must be accompanied by proportional scaling in the propagation delay of all other units (to enable frequency scaling). To make things even more complicated, different units of the design tend to scale differently

and are not ideally pipelinable. This can result in pipeline slack, which increases effective propagation delay, and degrades performance.

The ability to employ different core designs opens the door to a much broader overall system design space. One aspect of this design space is the manner in which the workload space should be split up among the cores, for each to be customized to [35]. Another aspect is the customization of core designs to their constituent workload behavior. Correctly exploring this design space requires core customization in which intricate circuit-level details are accurately accounted for. In this design space, abstracting away the circuit-level details can lead to inaccurate assessments and the adoption of severely suboptimal design solutions. The following sections focus on these issues.

2.1. A High-Fidelity Circuit-Level Model

In order to be able to account for detailed circuit-level propagation delays, and conduct accurate core customization, we have developed a fully-synthesizable Verilog model of a contemporary pipelined out-of-order superscalar processor – referred to as Fabscalar [77]. This model has parameterized microarchitectural features, and is aimed at high-fidelity design space exploration. It enables evaluation of the effect of the propagation delay and pipeline depth of different microarchitectural units on overall performance under different technology characteristics. The results presented in the analysis of the performance of core-selectability are based on results attained from the synthesis of different configurations in 45nm technology. Other results are based on propagation delays estimated with the CACTI modeling tool [98].

As an example, Figure 2-1 shows the propagation delay of the select logic, extracted from the model, when the issue-queue size and issue width are scaled. These results show that, for a specific propagation delay, there is a tradeoff between the size of the issue queue and the issue width. The best tradeoff depends on the workload behavior. Thus, without knowing the characteristics of the technology, and that of the applications, it not possible to accurately determine the best core configuration for executing a given set of applications.

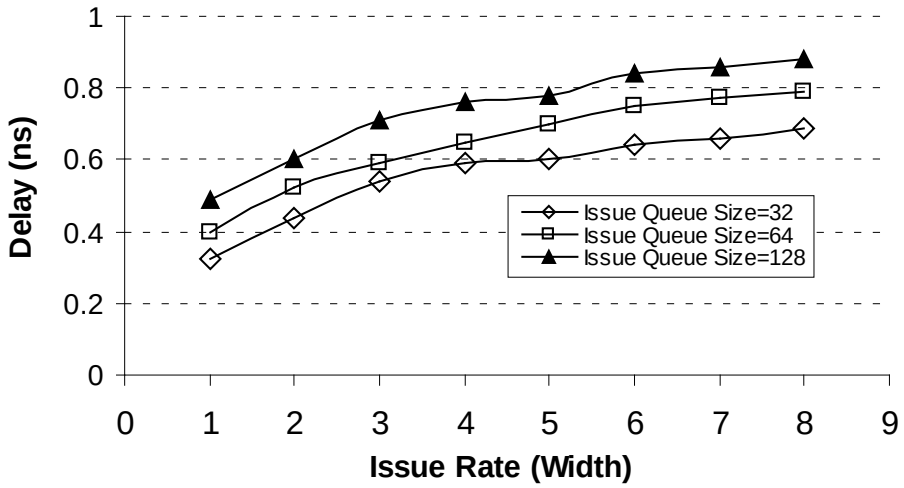


Figure 2-1: Propagation delay of the selection logic in 45nm technology, when the issue width and issue-queue size vary.

2.2. The Infeasibility of Gaining Performance from Reconfigurability/Adaptability

One approach to enabling different workloads to be executed on suitable core designs is through reconfigurability or adaptability. In implementing reconfigurability or adaptability, there is a tradeoff between the flexibility of the design and the overhead introduced to the system. Adaptable architectures [16] have less overhead, but provide less flexibility in the design. In contrast, FPGA-based reconfigurability [97] provides high flexibility at the cost of

large overheads. Nevertheless, in critical microarchitectural units the overhead of even the most inflexible forms of adaptability can outweigh the would-be benefit.

Although prior studies have proposed adaptable implementations of various microarchitectural units, their focus tends to be on reducing power consumption when needed, with minimal performance degradation. Achieving performance enhancement through adaptability, however, is more challenging. The following subsections address why this is.

2.2.1. Reconfigurability/Adaptability in the Logic

An example of the difficulty in implementing reconfigurability (or adaptability) can be observed in the superscalar wakeup logic. The main component of propagation delay in the wakeup logic is the load on the rails that broadcast newly issued instruction tags to the comparators that compare them with those of waiting instructions.

Downsizing the effective issue-queue size can, in theory, be achieved by switching off the load of the unwanted comparators. In practice, however, switching off the capacitance can only be achieved through buffering a portion of the broadcasting rail. Such buffering will provide the effect of a repeater when the issue-queue is not downsized [16]. But, since the buffer will need to be large enough to drive the rest of the broadcasting rail, it will permanently place a large extra load on the rail, increasing the propagation delay of the downsized portion.

Using the optimal repeater placement of SoC Encounter, it was found that in 45nm technology the best design for a 128-entry issue-queue consists of 4 large buffers, each with 9fF input capacitance, driving all the issue-queue comparators in a tree structured layout.

Implementing the ability to downsize the issue-queue within this circuitry results in a notably larger propagation delay compared to custom downsized issue-queue designs.

Table 2-1 shows the propagation delay of the wakeup and select logic in a 4-wide issue-queue with a maximum size of 128 entries that can dynamically be downsized. Also shown in this table is the propagation delay for custom downsized issue-queue designs. These results show that, the reconfigurable design, at its full size (of 128 entries) is 15% slower than a same-sized custom design. For smaller issue-queue sizes, the reconfigurable design becomes even slower relative to same-sized custom designs – with 46% longer propagation delay for the 16-entry size. Therefore, although there may be some benefit in this form of reconfigurability, it is far from the true benefit of customization.

Table 2-1: Effect of Issue-Queue Size on Propagation Delay With and Without Reconfigurability.

Issue-Q size	Wakeup Delay (ns)	Select Delay (ns)	Wake & Select Delay (ns)	Reconfig. Delay (ns)
16	0.55	0.54	1.09	1.55
32	0.635	0.59	1.38	1.89
64	0.67	0.65	1.62	2.1
128	0.82	0.76	2	2.3

Implementing reconfigurability in the issue width is even more challenging. The dependent instruction tags of waiting instructions are connected to as many comparators as there is issue width. The extra load of the unnecessary comparators in the downsized setting cannot be dynamically removed from the circuit without inserting extra buffering. This buffering introduces extra load in the system, which degrades performance.

Nevertheless, the direct overhead in propagation delay is not the only factor limiting performance gain from such reconfigurability.

2.2.2. Reconfigurability/Adaptability in the Pipeline Structure

Even if certain microarchitectural units were to somehow be made optimally changeable at the logic-level, the most difficult factor in attaining performance benefit from such changeability would be in conjunction with the tightly-coupled pipeline stages feeding to and from that unit.

For instance, *selectability*, as a general concept (providing the ability to *select* from among different implementations), can be incorporated to emulate ideal changeability in any microarchitectural unit. In addition, adaptability in caches [80] has also been shown to be implementable with fairly low overheads. However, it is challenging to draw performance from such confined changeability without causing slack and imbalance in the pipeline structure. The reason for this is that different design units scale differently and some do not scale at all.

For instance, the operation of the functional units is determined by the ISA and is unaffected by the microarchitectural configuration. Moreover, the functional units reside within the feedback loop of any microarchitecture, and are central correct functionality. Thus, any extra circuitry in their design will impact the raw performance of the system, and dramatically increase the verification effort of the system. Moreover, if such critical portions of a changeable core design were to be kept unchangeable, they would need to observe pipeline slack in low clock frequency configurations, for correct functionality in high frequency configurations.

Dynamic pipeline scaling (DPS) provides the potential for variable depth pipelining of a microarchitectural unit [52]. This can, in theory, enable the independent scaling of different

design units, while preventing slack in any pipeline stage. In practice, however, DPS introduces overheads in the form of extra latch propagation delay and extra power consumption. More importantly, it is only practical for multiplying or dividing the clock frequency, which dramatically limits the viable design space.

2.2.3. The Verifiability of Reconfigurability/Adaptability

Design verification needs to be accounted for in high level design choices as it has become the major consumer of man-hours in the development of modern processors [86]. It has been shown that design symmetry reduces verification cost by mitigating the effective number of functional states that need to be accounted for [4].

Reconfigurability, however, *desymmetrizes* the design of a processing core by creating differences between portions of the microarchitecture that would otherwise be identical. For instance, in the implementation of an adaptable issue-queue, the entries that are disabled in the down-sized mode differ from those that are never disabled. In addition, the logic that implements reconfigurability itself is inherently not symmetric. This can lead to an explosion in the number of states, and a potentially dramatic increase in verification effort.

Note that the potential for dramatic increase in verification effort is not exclusive to reconfigurable/adaptable designs. Most of the microarchitectural innovations of the past decade that attempt to push the envelope of overall single-thread performance inevitably end up complicating and desymmetrizing the design. For instance, techniques that attempt to make better use of critical structures such as the issue-queue or ROB, by temporarily decongesting them (i.e. draining stalled instructions from them) [7] [2] desymmetrize the design by introducing irregular control logic and data-paths to the system. Techniques that

attempt to exploit locality in data values [71] end up desymmetrizing the design with extra irregular circuitry necessary to recover from misspredictions. And Techniques that attempt to speculatively execute future code in advance [13] [15] desymmetrize the design with irregular circuitry necessary to monitor for changes to the input set. Core-selectability, however, allows for performance gain while keeping the complexity of each core design to a minimum by focusing it on only the constituent workload behavior of that core.

2.3. Related Work

2.3.1. Approaches to Splitting up the Space of Application Behavior

Numerous prior studies have analyzed different aspects of commercial benchmark suites [91] [105] [31] [30]. Other related studies have focused on understanding the similarities between different benchmarks [60] [40] with the objective of reducing the amount of simulation required to evaluate architectural innovations. For instance, Joshi et al. propose a methodology for measuring the similarities between programs based on microarchitecture-independent characteristics [3]. Hoste et al. use workload similarity to predict application performance [56]. Vandierendonck et al., rank SPEC2000 integer benchmarks based on whether they exhibited different speedups on different machines, and use this as a guideline for identifying similar workloads [39].

Yi et al. present a thorough summary of workload subsetting approaches [54], and propose a statistically rigorous approach based on the Placket-Burman design [85]. A similar technique is proposed by Dujmovic et al [46]. In these approaches, the execution of different workloads is evaluated on different processor designs in order to expose their architectural bottlenecks – which are considered to be indicative of similarity. An underlying assumption on which these

approaches are built is that interaction between the different workload characteristics (and their corresponding architectural units) is negligible.

In recent work Lee and Brooks comprehensively examine the problem of splitting up the workload space between differently designed cores, from a statistical standpoint [9]. An intriguing aspect of their study is the point that the workload space may be split up for very different overall figures of merit – an aspect that is unfortunately not addressed in this thesis. Navada et al. [102] also look at the potential of criticality analysis in achieving more efficient superscalar design space exploration. A central goal of such studies, however, is *expedition* of the exploration process. Alternatively, the present thesis remains focused on accuracy – as expedition of the exploration process can be achieved by means of a more tractable solution than recondite statistical analysis: simply employing more simulation resources.

Also, an aspect that is lacking in the evaluation methodology of such prior studies is the importance of accounting for the pattern with which tasks/applications tend to arrive at the system [37].

2.3.2. Solutions to the Dynamic Employment of Differently Designed Cores

The *Conjoined-core* [90] approach to Chip Multiprocessor design has been proposed as a solution to efficiently increase concurrency. The approach is a tradeoff between simultaneous multithreading [23] and single-chip multiprocessing [61]. Multiple homogeneous cores are added to the processing nodes of a CMP, and the cores time-share resources such as the floating-point unit, instruction cache, data cache and crossbar ports. In this manner, concurrency can be increased with fewer resources. The CASH architecture [83] is a similar approach.

In the implementation of core-selectability, resources are also shared between cores. However, the objective is to enhance performance through dynamically changing the employed core, rather than concurrency. In fact, while the sharing of resources is part of the objective in the conjoined-core and CASH techniques, it is more of an imposition in core-selectability.

Core-fusion [26], *Composable Lightweight Processors* [14], and *CoreGenesis* [96] are similar techniques to enable reconfiguration of the cores in a chip multiprocessor by combining smaller cores or subcomponents, to form larger cores (i.e. sharing core resources). This enables the cores to dynamically vary and become more suitable for the application at hand. However, these techniques are not aimed at providing performance benefit to scalable multithreaded applications, as maximizing the number of available cores would provide the best overall performance to such applications. More importantly, the reconfiguration overhead of implementing both techniques manifests itself in the microarchitectural units that are most critical to ILP extraction, i.e., the issue-queue, ROB and LSQ. Salverda and Zilles have also shown that there are fundamental obstacles to achieving good performance through core-fusion with in-order cores [79]. Moreover, as our circuit-level estimates indicate (see Section 2.2.1), even simple forms of adaptability are largely suboptimal compared to custom design solutions – let alone techniques that share time-critical units.

Previous studies have demonstrated the performance and power benefit of heterogeneity over homogeneity in CMPs designed for multi-programming environments [88] [89] [34] [87]. Heterogeneity entails the employment of differently designed cores for the execution of tasks with different workload behavior. However, it has also been shown that heterogeneity can

degrade the performance predictability and scalability of multi-threaded applications [99]. In panel talks, Patt [49] has suggested the abstract notion of employing large specialized units that can be powered down when not needed (the "Refrigerator" analogy), as a power-efficient approach to putting the growing availability of transistors to use. However, to the best of our knowledge, no prior work has looked at the option of employing complete processing cores that are differently designed with the intent to employ only one at a time.

In a recent study, Lue et al. [112], among other analysis, evaluate the potential of a very interesting form of core-selectability (that is unfortunately not addressed in this thesis); core-selectability between SMT and non-SMT cores. This form of core-selectability expands the potential of this technique to the realm of dynamic customization in the thread-level concurrency provided by the system.

2.3.3. Solutions to Bringing About Change in the Core Configuration

One approach to enabling dynamic change in the employed core microarchitecture is intuitive: make the microarchitecture *dynamically changeable*. Designs based on this approach are referred to as adaptable (when flexibility in the dynamic change attainable is traded for less circuit-level overhead) or reconfigurable architectures (when circuit-level efficiency is traded for more flexibility in the dynamic change attainable). There are, however, fundamental technology limitations with any such approach.

Reconfigurable computing [55] exploits field-programmable technology to build processors that can fundamentally transform their architecture. Such approaches can provide flexible changeability. However, the implementation of a specific architecture in reconfigurable technology is prone to severe sub-optimal fixed-configuration performance. By trading in

some flexibility it is possible to ameliorate the circuit-level overheads. However, in doing so, dynamically maintaining a balanced pipeline is rendered extremely challenging. This imbalance is a result of the fact that all microarchitectural units are tied to a common pipeline structure.

2.3.4. Solutions to Detecting the Most Suitable Configuration for an Application

A plethora of prior work has studied different approaches to enable aspects of a processor microarchitecture to change and become more suitable for the immediate workload behavior in order to achieve better power efficiency. Ponomarev et al. [21] and Folengnani et al. [18] propose approaches to learning the optimum issue queue size, and Yang et al. [104] propose cache miss-rate as a metric for determining when to downsize or upsize an adaptable I-cache. In Complexity Adaptive Processing (CAPs) [16], a single processor is architected such that the tradeoff between IPC and clock-rate can be dynamically altered. An essential component of the CAP architecture is the "configuration control" unit which selects the optimal configuration for the immediate workload through a heuristic learning technique or profiling information. Dhodapkar and Smith [1] and Balasubramonian et al. [80] propose tuning processes for identifying the best-suited configuration for the immediate code. *Temporal* approaches, such as the Rochester algorithm [80] or signature based approaches [1], require lengthy tuning processes. *Positional* approaches [110] enable faster adaptation, and have been found to be more effective, yet they are unsuitable for fine-grain switching due to the drastic increase in storage requirement.

Dropsho et al. [93] propose the separation of microarchitectural units, in what is referred to as the Globally-Asynchronous Locally-Synchronous (GALS) design. In this approach,

different units are asynchronous to each other, thus allowing each to be scaled independently. This allows for more predictability in the effect of independently scaled units on overall performance. Thus, it relieves the system of the need for an exhaustive tuning process that tries out different configurations.

Chen et al. [59] investigate the potential of employing pipelines of different widths and dynamically directing work to them based on local ILP. They use the parallelism metrics gathered from a dynamic Data Dependence Tracking mechanism to steer windows of instructions to suitable pipelines. In order to avoid most of the inter-cluster penalty, they limit switching between clusters to coarse granularities and continuously forward values to the disabled pipeline. However, data dependence tracking takes a two-prong view of the microarchitectural design space, consisting of either simple pipelines that can be clocked at high frequencies or wide superscalars that can be clocked at lower frequencies. In reality, a large range of microarchitectural parameters affect overall performance. After all, even a wide superscalar processor can be clocked at a high frequency if it is pipelined deeply.

2.3.5. Solutions to Accurately Modeling Circuit-level Details

Integral to our evaluation methodology is the circuit-level modeling of the propagation delay of different microarchitectural units in the design of processing cores. The freely available HDL processor models, Opencores [42] and Opensparc [43], do not represent the complexity of an out-of-order superscalar microarchitecture.

The Illinois Verilog Model (IVM) [41] is functionally the closest to Fabscalar, although it is not fully synthesizable. Moreover, IVM has not been developed to model arbitrary core designs – a key feature of Fabscalar. Most importantly, Fabscalar includes different pipeline

depths and superscalar widths, which is unprecedented for superscalar HDL models, while being crucial to comprehensive design space exploration.

CHAPTER 3: Splitting Up the Workload Space

A key dimension of designing a core-selectable system is the question of how the workload space should be partitioned for different cores to be customized to. It is this design challenge – which will be referred to as the *communal customization* problem – that adds to the importance of understanding the nature of workload similarity. Communal customization is akin to the more established notion of *workload subsetting* [54]. However, the objective of subsetting is to identify workloads that, in the space of workload characteristics, have relatively less Euclidian distance between each other. An underlying assumption is that such workloads are affected similarly by the architectural configuration – thus providing potential for less simulation-based evaluation cost. In contrast, communal customization involves the identification of workloads that can attain *close-to-optimal* performance with the same architectural configuration.

In general, characterizing and understanding the behavior of common workloads is an essential facet of computer architecture research. But the commercial advent of the chip multiprocessor (CMP) in recent years has added to the value of understanding the similarities (and differences) between workloads, as it has increased the viability of employing differently design cores. In the context of this chapter, a heterogeneous CMP is a chip multiprocessor in which the cores are architected differently, each sacrificing general-purpose performance for better workload-specific performance – so as to in *commune* achieve an overall performance that would not be attainable with uniform core designs.

The major argument presented in this chapter is that, in the design of a core-selectable CMP, or any heterogeneous multi-core system, viewing workload subsetting as a substitute for communal customization – or even as a preliminary step to reduce exploration complexity – may direct the ultimate design towards suboptimality. The fundamental reason for this is that there are complex interdependencies between the different units of a processor design. These interdependencies cause different components of the optimal processor configuration to be affected by the workload behavior *as a whole* (rather than distinct characteristics). In addition, these interdependencies are influenced by the physical properties of the underlying technology and are thus not reflected in the workload characteristics alone.

Workload characteristics that pertain to functionally independent subcomponents of a processor design are commonly viewed as independent gauges of the optimal configuration of those subcomponents. Metrics for the biasness of branches, memory access localities and the distribution of dependent instructions are microarchitecture-independent examples of such characteristics that respectively relate to the branch predictor, data caches and the scheduling unit. However, the optimal configuration of each of these units is influenced by the clock period of the system, while the optimal clock period itself depends on the dynamics of how the different units scale. Thus, in the search for an optimal design, the unified clock period intertwines the different subcomponents.

For instance it is ill-conceived to consider similar optimal L1 cache configurations for two workloads based solely on the fact that they have similar spatial/temporal locality of memory access and working-set size. This is because other characteristics of the workloads (such as the control-flow behavior) affect the best configuration for other aspects of the design (such

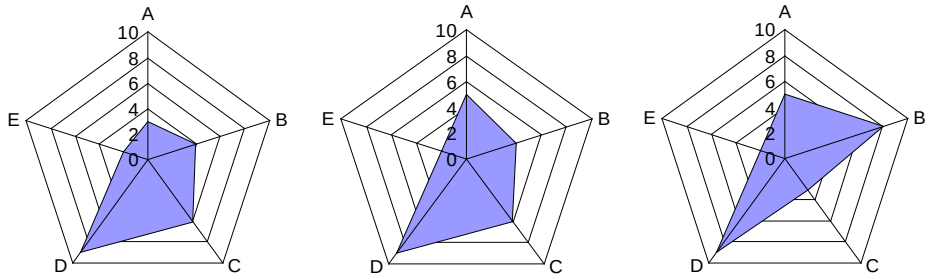
as the processor width and pipeline depth), including the clock period. The clock period determines the number of cycles necessary for accessing a cache unit with a particular configuration, which in turn influences the attainable IPC. Also the physical properties of the technology in which the system is implemented determine how the different design aspects scale relative to each other. These properties are not reflected in the raw characteristics of a workload. Moreover, it is not correct to assume that these interdependencies are of mere second-order effect as they involve issues as critical as the latency between back-to-back dependent instructions.

3.1. An Illustrative Example

As an illustrative example, consider three workloads α, β, and γ that have equal *importance weights*. The workloads have mostly similar characteristics, other than workloads β and γ having much larger working-sets than α, and γ having greater branch biasness and less dense dependence chains than α and β. Considering the Kiviat graphs of these workloads to be those displayed in Figure 3-1. It is evident that from the standpoint of raw workload characteristics, α and β are *relatively more similar* – as they differ only in the size of their working-set. Thus, a naive observation may conclude that customizing one of the two cores for workloads α and β, and the other for workload γ, will result in the best overall single-thread performance.

However, optimal performance with workload β may require a large L1 cache, due to its larger working set, and any compromise may severely degrade performance, while any increase in the L1 cache size may severely degrade the performance of workload α due to the high frequency of loads. Although workload γ also has a large working set, due its less dense

dependence chains and higher branch predictability, it will from an IPC standpoint be able to better tolerate cache misses. Therefore, depending on how the different units scale in the given technology, workload γ may be more suitable than β for execution on a configuration that is also suitable for α.



(α) **(β)** **(γ)**

Figure 3-1: Kiviat graphs of three sample workloads for the architecture-independent characteristics of: A) Working-set size, B) Branch predictability, C) Density of dependence chains, D) Frequency of loads, E) Frequency of conditional branches – all normalized to a scale of 0~10.

3.2. Configurational Workload Characterization

For the purpose of communal customization, an effective form of workload characterization would more transparently reflect how relatively suited different workloads are for being executed on the same core design – rather than how relatively similar their raw characteristics are. It is the characteristics of the best microarchitectural configuration for a workload that encapsulate this. Such characterization allows for direct and accurate measurement of how well different workloads perform on each other's customized architectures. Moreover, the best microarchitectural configuration for a workload encapsulates the effect of all the different facets of raw workload behavior on the optimal configuration for the microarchitectural units in a given technology.

It is undoubtedly more costly to determine the best configuration for a workload – which requires numerous cycle-accurate simulations of the execution of code – than it is to extract the workload behavior. However, knowing the best microarchitectural configurations for individual workloads enables the best combination of core configurations to become extractable through a *systematic* task of reducing the set of workloads/configurations. Moreover, determining the best configurations for workloads can be performed prior to the design phase that is critical for time-to-market demands, as the physical properties of future generation technologies are often predictable before commercial feasibility.

Although the discussion presented here is framed around the design of core-selectable CMPs, the notion of characterizing workloads based on their customized architectural configurations can have broader implications for processor architecture research in general. For instance, the solidity of conventional methodologies in the evaluation of microarchitectural techniques has long been scrutinized [50]. This is in part due to the fact that when any component of the workload characteristics, architectural configuration or technology characteristics change, it shifts the balance between workloads. In actuality, it is this shift in balance that should be considered representative of the effect of a microarchitectural technique – rather than the 'speedup' it provides on a given baseline microarchitecture. In other words, the customized processor configuration of a workload can be viewed as an *atomic* entity representing this balance between workload behavior and microarchitectural configuration. Standardizing the customized configurations of popular benchmark suites can pave the way to a more standard evaluation methodology – the need for which is also stressed in [50].

Nevertheless, this issue is of less practical significance in the domain of general-purpose core design, as such designs are constrained on multiple fronts by widely diverse workload behavior. It is when the workload space is to be split among different processing cores that this issue gains true significance.

In this chapter, a simulated-annealing exploration process is employed to explore the design space of the superscalar processor for each of the C integer benchmarks from the SPEC2000 suite, and determine their configurational characteristics. The contributions of this chapter are threefold: 1) Introducing a superscalar design-space exploration framework based on the popular Simplescalar simulator [106] and CACTI modeling tool [98]. 2) Determining the configurational characteristics of the integer SPEC benchmarks in a specific technology. 3) Illustrating how configurational workload characterization can enable the coherent and systematic extraction of the best cores to employ in a heterogeneous CMP.

3.3. Background

3.3.1. The Interdependence of Pipeline Stages

Under realistic design constraints the pipelined nature of traditional processor design brings about interdependence in the configuration of seemingly uncorrelated architectural units.

The uniform clock period determines the pipeline depth and the slack observed in different stages. Pipeline slack can greatly impacts the performance of non-linear pipelines such as the superscalar processor. Figure 3-2 illustrates how the unified clock can affect the optimal sizing of the issue queue and L1 cache. In each scenario the solid lines represent the delay of the issue queue, the dashed lines represent the access delay of the L1 cache, and the scales at

the bottom represent the clock cycles. The propagation delay of the issue queue and the delay of the L1 cache are based on a representative sizing of these units.

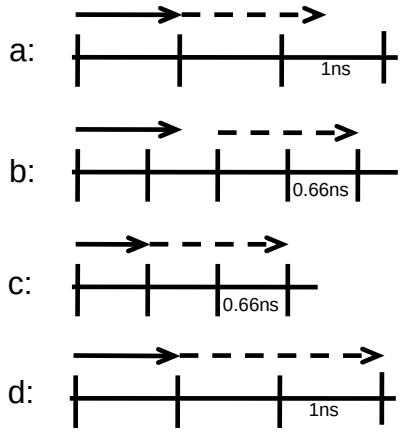


Figure 3-2: Illustrative scenarios for the design of a processor with different clock periods, issue queue and L1 cache sizes.

In scenario a, access to the L1 cache observes considerable slack. The overall slack can be reduced by changing the clock period, as displayed in scenario b. Doing so, however, deepens the pipeline which may improve or degrade overall performance. So is the case in scenario c, where the slack is further reduced by downsizing the issue queue size. Finally in scenario d, instead of scaling down the clock period, the size of the L1 cache is increased to make full benefit of the available two cycles. Depending on the working set of the application this extra L1 cache capacity may not be of any value, and scenario b or c may be of better overall performance.

3.3.2. Two Approaches to Communal Customization

The focus of this chapter revolves around the question of how workloads should be ‘characterized’ for this purpose and what the criterion for ‘commonality’ should be. To the

best of the author's knowledge, there are only two relevant prior studies that touch upon the issue of communal customization.

Kumar et al. [87] reduce the set of benchmarks based on similarity in workload characteristics. By doing so, and limiting the architectural diversity of the cores, an exhaustive search of different core-combinations across different groupings of workloads is made feasible. This approach is ad hoc in that there is no solid justification (other than exploration cost) to reduce the benchmark set to a specific size.

Lee and Brooks [10] use regression modeling to enable fast exploration of the microarchitectural design space. Then, through an iterative process (K-means clustering) *centroids* are identified for the customized architectures. The closest centroid to the customized architecture of each benchmark is assigned as the *compromise architecture* of that benchmark. This approach is also ad hoc in that its outcome is highly dependent on how the different architectural parameters are normalized and weighed. It is, however, the most related approach to that proposed here, and addresses the major focus of this chapter.

Therefore, as illustrated in Figure 3-3, there are two general approaches to communal customization. One is to initially extract a small enough subset of the essential workloads for it to be feasible to conduct an exhaustive exploration of the workload-architecture combinations. The other is to initially determine the optimal architectural configuration for each considered application and then reduce the set of resultant architectures to a representative subset. With the availability of the optimal configuration for each workload the true *closeness* between them can systematically and accurately be measured and the most representative subset determined.

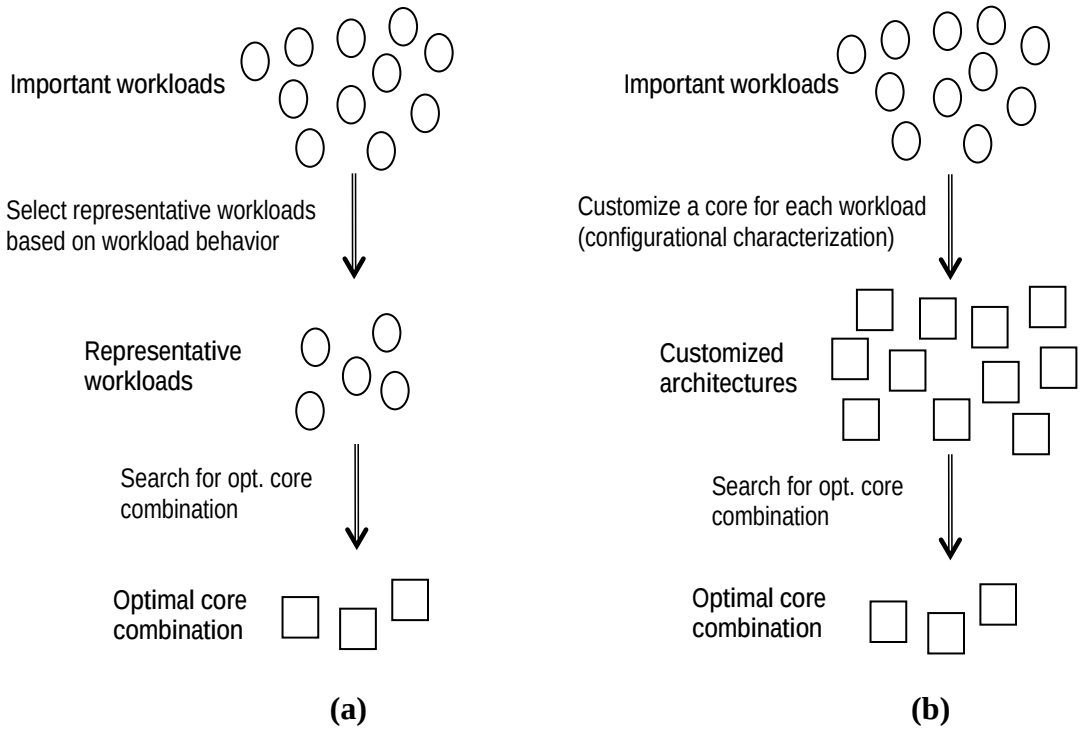


Figure 3-3: Two general approaches to identifying the optimal core combination for a heterogeneous CMP: (a) An exhaustive search – which for feasibility requires selection of representative workloads, (b) Determining the customized architectures of applications and then reducing the set of architectures.

3.3.3. Automated Design Space Exploration

Due to the sheer size of the superscalar processor design space, determining the optimal architectural configuration for a workload is itself a demanding task. A wide spectrum of studies have focused on developing tools to enable efficient exploration of the design space with different degrees of accuracy and architectural variability [67] [45] [113] [20].

At one end of the spectrum are approaches that are more concerned with specific design details. For instance, AMPLE [67] is a wire-driven microarchitectural design space exploration framework in which the size of different units and the floor-planning are customized to workload behavior. Initial microarchitecture parameters for the initial search

point of each application are determined based on the application's characteristics. The clock period is not among the customizable design parameters and different design units are not pipelinable (it only weighs the power benefit of downsizing against its performance degradation).

Due to the non-discrete nature of the clock period, considering it as a customizable parameter considerably increases the processor design space. It is for this reason that prior design exploration studies either limit the design space to a set of pre-designed configurations [87] [46] or consider a fixed clock period across variability in other architectural parameters [67]. Both effectively diminish the true performance potential of customization (and heterogeneity).

On the other end of the spectrum are approaches that are more concerned with the speed of performance evaluation (and exploration). For instance, Lee and Brooks introduce a non-linear microarchitectural regression model, and propose its use to enable fast exploration of the processor design space [10]. While pipeline depth is among the customizable design parameters, it is employed as a speed-power factor and not necessarily a parameter influencing the sizing of different units in a balanced pipeline design.

However, the major issue with such mathematical models is the space in which their accuracy is verified. In general, misleading conclusions may be drawn on the accuracy of a model if the evaluation is conducted in a distorted space. This can occur when the evaluated space is a subset of the actual space due to the absence of variability in certain parameters, or is its superset due to the absence of the enforcement of certain restrictions – or a combination of both factors. The problem in evaluating superscalar regression models is that accounting

for independent variability in the pipeline depth of different units and enforcing the constraints imposed by a global clock period results in a design space that cannot be concisely delineated (its shape and bounds specified) in parametric form.

Therefore, there is more difficulty in evaluating the accuracy of such models than meets the eye. However, inaccuracy in this area can lead to incorrect conclusions when the model is employed for design-space exploration, principle component analysis or clustering. The advocates of using regression models for design-space exploration argue that employing full-blown simulation is too time consuming and costly. However, the process is highly parallelizable and that with sufficient resources (which are typically available in large development groups) a design space exploration with reasonable rigor should be achievable in a matter of days. For this reason basing the exploration process on recondite regression models serves little benefit to such a study.

3.4. XP-Scalar: A Superscalar Design-Exploration Framework

A light-weight superscalar design-space exploration framework named *xp-scalar* has been developed. The major component of the framework consists of a tool that employs a simulated annealing process to find the best superscalar architectural configuration for executing a specific workload. Also available is a tool for visualizing the performance of the benchmarks on each other's customized configurations, which eases the identification of discrepancies and can help expedite the exploration process. The source code of the framework and directions for use are accessible at http://www4.ncsu.edu/~hhashem/xpscalar.htm. The tool employs the Simplescalar v.4 simulator to perform execution-driven simulations, and the CACTI model to approximate the access latency of the different units of the

superscalar processor. Both Simplescalar and CACTI can be employed with or without any modification, as long as the format of the input and output does not change.

In each iteration, either the clock period is varied, and the size of the issue queue, register-file/ROB, load-store queue, L1 and L2 caches, and processor width adjusted to make their access times fit within the number of pipeline stages assigned to them, or the number of pipeline stages of a unit is varied and its configuration appropriately adjusted. Table 3-1 displays the manner in which the tool uses the CACTI output parameters to estimate the access latency of different architectural units. Note that the issue queue delay consists of wake-up (an associative component) and select (a direct mapped component) delays.

Table 3-1: The CACTI parameters used to determine the access latency of various units based on architectural parameters.

Unit	Line size	Associativity	No. of sets	No. read ports	No. write ports	CACTI output component
`L1 d-$	line size	assoc. of cache	no. of sets of cache	2	2	Access time
L2 d-$	line size	assoc. of cache	no. of sets of cache	2	2	Access time
Wakeup	8 bytes	fully associative	2 x size of issue queue	Issue width	0	Tag comparison
Select	8 bytes	direct mapped	size of issue queue	Issue width	0	+ Total data-path without output driver
reg. file	8 bytes	direct mapped	size of ROB	2 x issue width	issue width	Access time
LSQ	8 bytes	fully associative	size of LSQ	2	2	Total data-path without output driver

After the different units are scaled to fit the product of the clock period and their pipeline depth, minus the aggregate latch latency, the benchmark is executed on the *sim-mase* simulator (from the Simplescalar toolset) configured correspondingly. If a configuration executes the workload with greater IPT (Instructions per Time-unit) than the best observed until that point in the exploration, the configuration is recorded as the new optimal solution. When a configuration is reached for which the IPT is less than half that of the optimal configuration, the exploration process rolls back to the optimal solution and is continued.

In this study, power and die area are not considered in the evaluation process, and optimum design is concerned only with performance. It is however found that under realistic assumptions for the access latency to different superscalar subcomponents, these aspects of the optimum architectural configuration remain within acceptable limits. Extending the tool to conduct exploration based on a metric that represents some combination of performance, power and die area should not be exceptionally difficult.

3.5. Exploration Results

3.5.1. Methodology

The workloads evaluated are the C integer benchmarks from the SPEC2000 suite compiled for the PISA instruction-set. The exploration process was conducted on a quad-core hyper-threaded blade for a period of three weeks. During this period, each workload was also executed on the customized architectures of the other workloads. If a workload was found to perform better on some other workload's optimal configuration, that configuration would replace its own configuration in order to expedite the exploration process. The evaluation of each architectural configuration during the exploration process consists of the execution of a 100-million instruction Simpoint [74]. A considerably large number of such evaluations need to be conducted for each benchmark in order for the evolutional process to approach the optimum design. Therefore, in the initial stages of the exploration, each evaluation was limited to the first 10 million instructions.

Three microarchitecture-independent technology-dependent factors were found to be influential on the ultimate customized configurations attained for the benchmarks. Table 3-2 displays the values considered for these parameters in this study. The *memory access latency*

determines the amount of time required to access the main memory, i.e., the latency of a load that misses in all cache levels. The *front-end latency* is the amount of time required for an instruction to be retrieved, decoded and renamed, i.e., the extra branch misprediction penalty in the Simplescalar simulator. CACTI does not produce accurate modeling for block sizes smaller than 8 bytes. Therefore, in this thesis this lower bound is considered as the width of the issue-queue entries. Another important design constant is the latch latency which affects the optimum pipeline depth of different subcomponents. These values are in general accordance with common processor designs.

Table 3-2: Fixed design parameters across all configurations.

memory access latency	50ns
front-end latency	2ns
bit-width of IQ entries	64
latch latency	0.03ns

Table 3-3 displays the initial architectural configuration employed across all benchmarks. Note that only the access latencies (in clock cycles) of the caches are indicated. This is because the cache configurations are randomly varied to fit the product of the clock period and number of access cycles during the first iteration of the exploration process if the default does not fit.

Table 3-3: Initial configuration used across all benchmarks.

No. of cycles for memory access	172
No. of pipeline stages of front-end	6
Dispatch, issue, and commit width	3
ROB size	128
Issue queue size	64
Min. lat. for awakening of dep. instr.	1
Pipeline depth of Scheduler/Reg-file	1
Clock period (ns)	0.33
L1D access latency	4
L2D access latency	12
Load-sotre ques size	64
Pipeline depth of LSQ	2

3.5.2. Individual Customized Core Designs

Table 3-4 displays the characteristics of the optimum architectural configuration for each of the considered benchmarks. The optimum processor width is observed to vary between 3 and 7. The optimum ROB size varies between 64 and 1024. The optimum clock frequency varies between 1.72 GHz and 5.2 GHz. The optimum size for the L1 cache capacity is in the range of 8K to 256K, while that of the L2 cache is in the range of 128K to 4M bytes.

Table 3-4: The customized architectural configurations for the SPEC2000 benchmarks.

	bzip	crafty	gap	gcc	gzip	mcf	parser	perl	twolf	vortex	vpr
No. of cycles for memory access	112	321	173	186	198	120	198	321	172	213	172
# front-end pipeline stage	4	12	6	7	7	4	7	12	6	8	6
Dispatch, issue, and commit width	5	8	4	4	4	3	4	5	5	7	5
ROB size	512	64	128	256	64	1024	512	256	512	512	256
Issue queue size	64	32	32	32	32	64	32	32	64	32	64
Min. lat. for awakening of dep. Instr.	0	3	1	1	1	0	1	3	1	2	1
Scheduler/Reg-file pipeline depth	1	3	1	2	1	1	2	4	2	4	2
Clock period	0.49	0.19	0.33	0.31	0.29	0.45	0.29	0.19	0.33	0.27	0.3
L1D associativity	2	1	1	1	1	2	1	1	8	4	2
L1D block-size	32	8	8	8	128	128	64	8	64	32	32
L1D no. of sets	1k	16k	2k	32k	256	1k	2k	2k	128	1k	128
L1D access latency	2	5	2	4	3	5	3	3	3	5	2
L2D associativity	4	16	4	8	1	4	8	16	4	16	8
L2D block-size	64	64	256	64	128	128	512	64	128	128	128
L2D no. of sets	8k	128	128	1k	4k	8k	32	128	2k	128	1k
L2D access latency	15	7	4	6	5	27	12	7	12	6	12
LS-queue size	128	64	256	256	128	64	256	128	256	256	64

Please see the Fascalar website for more up-to-date results from further evolution of exploration process, and the effect of improvements in the accuracy of modeling the latencies of different units – a process that with feedback from the community will be on-going. Nevertheless, the major conclusions drawn here are unlikely to be annulled with greater accuracy in the modeling of the propagation delay of different microarchitectural units.

3.6. Measuring the Merit of Communal Customization

3.6.1. Cross-Configuration Performance

Once a customized architectural configuration for each benchmark has been established, the performance of each benchmark on the configuration of other benchmarks can be determined. This allows for the performance difference between different architectures to be extracted, and the architectures that provide close-to-optimal performance across numerous benchmarks identified. The best core configurations are not necessarily among the workload-customized cores. However, the broader the workload diversity, the better coverage there will be of the design space.

Table 3-5 displays the IPT of each of the SPEC2000 benchmarks executed on the optimal architecture of all the other benchmarks. From these results, the percentage slowdown of each benchmark when executed on the architectures of other benchmarks over the performance on its own architecture can be extracted. The importance of carefully choosing the cores of a heterogeneous CMP is evident in these results – with up to ~50% slowdown (for the execution of *mcf*) observed.

Table 3-5: The performance of each benchmark (rows) on the customized architectures (columns) of other benchmarks.

	bzip	crafty	gap	gcc	gzip	mcf	parser	perl	twolf	vortex	vpr
bzip	3.15	2.02	1.73	2.41	2.11	2.56	2.09	2.03	3.05	2.24	2.95
crafty	0.78	2.31	1.15	2.11	1.91	0.48	1.97	2.06	1.29	2.12	1.30
gap	1.39	2.75	3.02	2.60	2.92	0.89	2.89	2.79	2.00	2.47	2.05
gcc	1.17	2.17	1.42	2.27	2.03	0.75	2.02	1.63	1.79	2.06	1.80
gzip	1.78	2.56	2.02	2.88	3.13	1.28	3.01	2.14	2.39	2.57	2.37
mcf	0.74	0.40	0.30	0.45	0.29	0.93	0.32	0.41	0.52	0.42	0.52
parser	1.86	2.11	2.19	2.08	2.47	1.32	2.62	1.86	2.39	2.15	2.30
perl	0.85	2.02	0.90	1.81	1.67	0.54	1.65	2.07	1.32	1.81	1.30
twolf	1.65	0.98	0.81	1.26	0.88	1.18	1.10	0.91	1.83	1.16	1.77
vortex	1.68	2.98	2.55	3.09	2.91	1.07	3.41	2.78	2.61	3.43	2.54
vpr	1.56	1.33	1.13	1.72	1.09	1.05	1.36	1.29	2.00	1.51	2.09

3.6.2. Overall Figures of Merit

Before the best set of core configurations to employ in a heterogeneous system can be identified, the design-goal needs to be determined and a figure of merit that represents that design goal.

If the goal is to minimize the total execution time of a set of *consecutive* benchmarks, as is customary in single-core miroarchitecture research, a representative figure of merit is the harmonic-mean of the performance of each benchmark when run on the most suitable core available for it. Such a design goal however does not account for core-contention. It may thus cause preference towards adoption of configurations that perform extremely well with a few benchmarks without considering the burden this may place on other more general configurations. If the objective is to increase the average performance with which an arbitrary benchmark from a set of benchmarks will be executed when submitted in isolation to the system, a representative figure of merit is the average performance of each benchmark on its most suitable core available.

A more real-world design goal is to minimize the total execution time of a set of benchmarks that can be executed concurrently on separate cores (if available). A representative figure of merit for this can be attained by first dividing the performance of each benchmark when run on the most suitable core available for it, by the number of benchmarks with which it shares that core, and then taking the harmonic mean. This will be referred to as the *contention-weighed harmonic-mean*.

3.6.3. The Performance of Different Core Combinations

Table 3-6 displays the best set of cores to employ for different core-counts, in order to maximize the harmonic-mean, average and contention-weighed harmonic-mean of the IPT (respectively represented by *har*, *avg* and *cw-har*) of the integer SPEC2000 benchmarks. These results were attained from the results of Table 3-5, by conducting a complete search of all possible core-combinations. A tool for automating this task is also part of the xp-scalar framework. These results show that a well-designed two-core heterogeneous CMP, can provide ~10% and ~20% speedup in average and harmonic-mean IPT respectively, over the best single-core configuration.

Table 3-6: The best core combinations and their performance.

	customized core(s)	avg. IPT	har. IPT
best config for avg. & har. IPT	gcc	2.06	1.57
2 best configs for avg. IPT	parser, twolf	2.27	1.76
2 best configs for har. IPT	gcc, mcf	2.12	1.88
2 best configs for cw-har. IPT	bzip, crafty	2.18	1.87
3 best configs for avg. IPT	crafty, parser, twolf	2.35	1.82
3 best configs for har. IPT	crafty, mcf, twolf	2.27	2.05
4 best configs for avg. & har. IPT	crafty, mcf, parser, twolf	2.32	2.08
each benchmark on its own customized architecture	-	2.38	2.12

Figure 3-4 displays the single-thread performance attainable from executing the benchmarks when the number of core configurations available is limited. These results show that the choice of available configurations can greatly impact individual benchmark performance. For instance, the benchmarks *twolf* and *parser* displays around 40% and 25% speedup respectively over the best single configuration when the best two configurations for average IPT are employed. Similarly, the benchmark *mcf* attains close to 2x speedup over the best single configuration when the best two cores for harmonic mean performance are available.

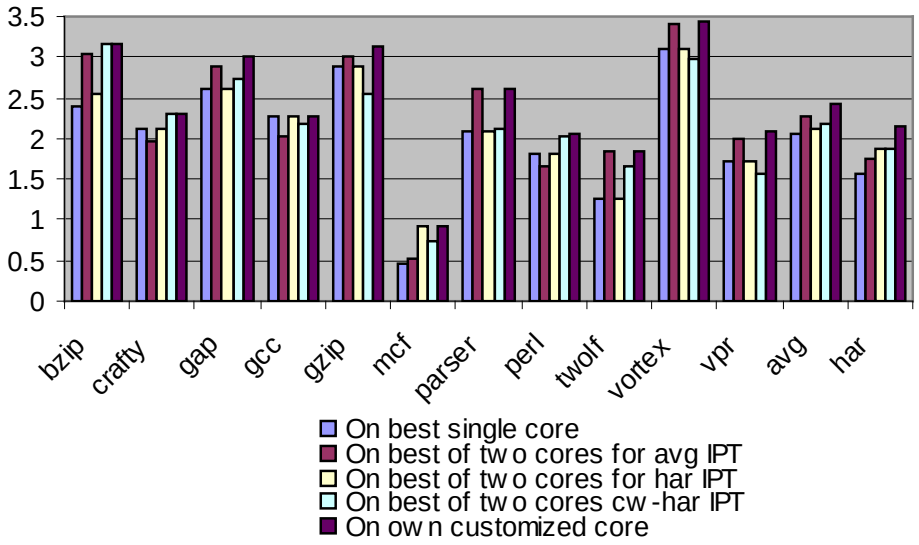


Figure 3-4: The IPT of the execution of different benchmarks on the best available core, with different configurations.

However, the availability of the customized architectural configuration of *mcf* provides hardly any benefit for the other benchmarks (only *bzip* attains a slight performance enhancement). This shows how other parameters, such as the *importance-weight* of benchmarks, can influence the best core combination. For instance if *mcf* were to have a considerably lower importance-weight than the other benchmarks, the best two configurations for harmonic-mean performance would potentially be different.

More importantly, using workload characteristics to eliminate benchmarks from the exploration process in the design of a heterogeneous system can lead to suboptimality in the final design solution. If *gzip* is assigned as the representative benchmark for *bzip*, a reevaluation of the dual-core combination for harmonic-mean IPT finds the configurations of *bzip* and *crafty* to be the best solution. These two configurations however result in a harmonic-mean IPT of ~1.87, and a ~0.5% slowdown compared to when *gcc* and *mcf* are

employed. Although the effect is small, this example shows the effect of excluding a *single* benchmark based on subsetting, and proves how relative similarity in workload characteristics can be misleading if interpreted incorrectly.

The regions of code and compilation settings employed in the aforementioned studies may differ from that employed here. Nevertheless, relative similarity in workload behavior is a discrete property and major similarities are not expected to be affected by minor variations in the characteristics. Therefore, as long as the regions of code are roughly representative of the whole benchmark, such a cross-publication comparison can be considered legitimate.

3.7. Chapter Summary

A superscalar design space exploration tool that allows variation in the sizing of different units of the superscalar processor is employed to determine the optimal architectural configuration for each of the integer SPEC benchmarks. The best core combinations to employ based on different criterions were determined, and it is shown that through initially reducing the set of workloads based on similarity in raw characteristics, may result in lower performance. This shows that the optimal architectures for executing workloads provide a more valuable source of information about the similarities between the workloads with respect to their resource needs.

CHAPTER 4: Core-Selectability in Chip Multiprocessors

In the design of a chip multiprocessor (CMP), if the balance in resource provisioning is to be maintained, an increase in the number or performance of processing cores requires an increase in the cache and interconnection resources. This does not necessarily mean that all applications utilize the provisions to the fullest extent, however. By overcoming the instruction-level bottlenecks of applications that underutilize the cache and interconnect, it is possible to enhance their execution performance, and yet maintain the balance in provisioned resources. This will result in better utilization of provisioned cache and interconnect.

A major factor that inhibits instruction-level performance enhancement is the one-size-fits-all approach to the design of the centralized units necessary for extracting instruction-level parallelism (ILP), e.g., the issue-queue, load/store queue (LSQ), reorder buffer (ROB). This is a result of the inherent criticality of these units, which renders them impractical for dynamically changeable design solutions (i.e., reconfiguration). If it were possible to genuinely adjust the configuration of these units to suit the application at hand, notable instruction-level performance could be gained.

This chapter investigates the implications and benefits of implementing core-selectability in a general-purpose chip multiprocessor. In the experimental evaluation, the considered core designs are based on propagation delays observed for different microarchitectural structures attained from a detailed synthesizable HDL model of a superscalar processor in 45nm technology.

4.1. Considerations in the Implementation of Core-selectability

The fundamental mechanisms required to implement core-selectability, e.g. port sharing, have been proposed in prior work, and employed in commercial products. Thus, the novelty of core-selectability lies in the purpose for which such mechanisms are employed, rather than the mechanisms themselves.

In the front-end, the cores can share a port to the instruction cache. However, it is important that each core possess a dedicated fetch engine, as this unit is closely tied to the functional units that determine branch outcomes. In the back-end, the data-paths of differently designed cores need to share a port to the data cache. Only the core that is selected for active employment will have access to these ports.

The cores can be made selectable at the L1 cache level or the port to the shared L2 cache. Implementing selectability at the L1 level has the advantage of better utilization of die area. Selectability at the L2 level has the advantage of enabling L1 cache customization, and places no overhead on the more critical L1 cache accesses. This study focuses on implementing core-selectability at the L1 level. The design of the Rock processor [28] ensures that, at the very least, port-sharing is physically implementable at this level.

Another option in the implementation of core-selectability is whether or not to allow the different cores to have different clock frequencies. Customizing the clock frequency allows for the pipeline depth of the core designs to be customized to the characteristics of the workload, and can considerably increase the viable design space of the processing cores.

Another implementation issue that needs to be addressed is the mechanism for activating and deactivating cores in a node, i.e., transitioning from one configuration to another. When the operating system scheduler chooses to schedule a task on a core other than the currently active core, it first needs to finish up what it is doing on the currently active core and then executes a final instruction on the currently active core, that simultaneously configures the currently active core to be deactivated and asserts an external interrupt signal of the core to be activated. This implies that (1) a core needs to be able to assert the activation interrupt signal of any other core and (2) a core's external interrupt unit needs to always be active whether or not the core is active. When an inactive core receives an activation interrupt, it needs to vector its program counter to an interrupt handler that will start the task. All other user-level and system-level instructions need only execute on the currently active core.

4.2. Evaluation Methodology

4.2.1. Customizing the Core Design

The goal of core customization is to find a global design optimum that captures the interplay between workload characteristics, the microarchitecture, and the physical implementation. Thus, propagation delays of microarchitectural units are fundamental to this exercise. To this end, we have developed a synthesizable Verilog model of an out-of-order superscalar processor. Details of this model are available in a preliminary report [77]. Major components or features are either parametrically configurable (*e.g.*, structure sizes) or different configurations for them have been explicitly designed (*e.g.*, number of superscalar ways in each pipeline stage). Different designs were synthesized with Synopsis Design Compiler

V2005.09-SP3 and placed-and-routed with Cadence SoC Encounter V7.1, using the FreePDK OpenAccess 45nm Standard Cell Library [48].

Since a superscalar processor makes use of many specialized and highly-ported RAMs (*e.g.*, rename map table, architectural map table, shadow map tables, free-list, active-list, physical register file, etc.), a register file compiler has also been developed as part of the Fabscalar infrastructure. It uses custom layouts of multi-ported bit-cells and peripheral circuits to generate RAMs and characterize their access times (SPICE model extraction).

4.2.2. Multicore Simulation Setup

We explore the core design space and evaluate the effect of core-selectability with full-system simulation using the Virtutech's Simics simulator [44] extended with the Wisconsin GEMS and OPAL [75] simulators. The GEMS simulator provides a detailed memory system timing model, and the OPAL simulator provides a detailed microarchitectural timing model of a processor with the Sparc ISA. The cache and interconnection characteristics considered in all studies are shown in Table 4-1.

Table 4-1: Cache and Interconnection Characteristics.

NETWORK TOPOLOGY	HIERARCHICAL SWITCH
COHERENCE PROTOCOL	MOESI
DATA BLOCK BYTES	64
L1 CACHE ASSOC	2
L1 CACHE NUM SETS BITS	9
L2 CACHE ASSOC	4
L2 CACHE NUM SETS BITS	12

A diverse set of multithreaded benchmarks from the Splash-2, Java-grande and SpecJbb benchmark suites, and the Blast biometric benchmark, are accounted for. The benchmarks

and the employed input parameters are listed in Table 4-2. The benchmarks were compiled using the PARMACS [82] library from UPC.

4.2.3. The Core Designs

A major factor in the evaluation of core-selectability is the design of the cores employed in the system. The different microarchitectural parameters were explored under the constraint that the propagation delays of different units remain within a certain number of clock cycles. The performance of each design solution was evaluated for the benchmarks detailed in Table 4-2.

The benchmarks were executed on the multithreaded simulation setup detailed in the previous section for 10 million instructions. This was preceded by skipping the initialization phase of each benchmark to arrive at the main execution loop, and warming up the caches for 10 million instructions.

Table 4-2: Benchmarks with Input Parameters

Suite	Benchmark + input parameters
	barnes 8192 123 0.025 0.05 1.0 2.0 5.0 0.075 0.25 4
	cholesky -p4 -B128 -C16384 < tk29.o
Splash2	fft -m22 -p4 -n65536 -l4
	fmm two cluster plummer 8192 1e-6 4 5 .025 0.0 cost zones
	lu -p4 -n2048 -b64
	ocean –n258 -p4 -e1e-07 -r20000 -t28800
	radiosity -p 4 -room –batch
	radix -p4 -n2621440 -r2048 -m524288
	raytrace -a8 -p4 teapot.env
	volrend 4 head
	water_spatial < input.p4
Java-Grande	java -cp .:/RayTracer/jg JGFRayTracerBenchSizeA 4
	java -cp .:/MoldDyn/jg JGFMolDynBenchSizeA 4
	java -cp .:/MonteCarlo/jg JGFMonteCarloBenchSizeA 4
Specjbb	java -classpath -propfile specjbb.props
Blast	blastall -p blastn -d ecoli_nt -a 4 < alu.n

The microarchitectural attributes of the best core design found for average execution time across all the benchmarks are listed under the label *Core-U* in Table 4-3. Two other core designs were also extracted that, compared to Core-U, provide notably higher performance on different subsets of the benchmarks. The microarchitectural attributes of each of these two core designs, which we will refer to as *Core-A* and *Core-B*, are also listed in Table 4-3. Each core design attains higher performance on a subset of benchmarks at the cost of lower overall performance across all benchmarks. In choosing these core designs, care was also taken to limit the design space to a fixed clock period, equal to that of Core-U (0.6 nanoseconds). This was necessary to preserve lucidity in the difference between the designs, and prevent the need for asynchronous buffering or adaptable caches.

Table 4-3: Configuration of Cores.

	Core-U	Core-A	Core-B
FETCH STAGES	4	3	5
DECODE STAGES	1	1	1
RETIRE STAGES	2	2	2
ISSUE WIDTH	3	2	5
ROB SIZE	512	1024	512
IWINDOW SIZE	64	128	32
Clock period	.6ns	.6ns	.6ns

Core-A provides higher performance to applications that have hard-to-access ILP. It has a large issue-queue and yet has limited issue width. This allows for the propagation delay of the wakeup-select logic to be focused on looking further ahead in the dynamic instruction stream to find the limited ILP. Core-B, on the other hand, provides higher performance to applications that have easier accessible local ILP. It has a smaller issue-queue, yet it has wider issue width. This allows for the propagation delay of the wakeup-select logic to be

focused on issuing more instructions per cycle. Figure 4-1 shows the average execution time of Core-A and Core-B across all the benchmarks, normalized to that of Core-U.

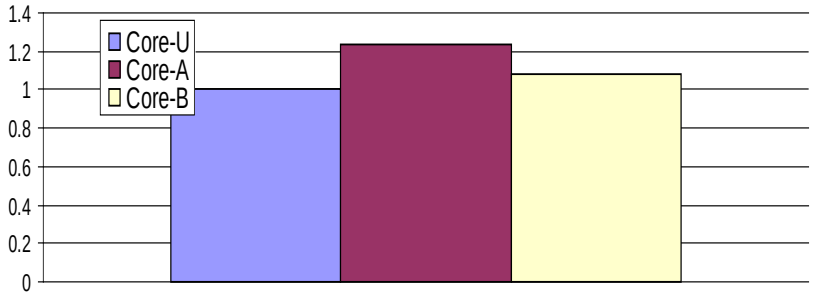


Figure 4-1: Average execution time across all benchmarks for different core designs normalized to that of Core-U.

Note that Core-A and Core-B are not truly the best core designs for core-selectability. However, they do present a scenario in which the source of performance difference between the cores is more lucidly discernable. Moreover, the manner in which the different benchmarks display preference towards being executed on these cores provides a convenient scenario to explain a few factors that need to be considered in the incorporated core designs.

4.3. Multithreaded Results

Figure 4-2 shows the execution time of individual benchmarks on Core-A and Core-B normalized to the execution time on Core-U. These results show that while Core-U displays the best overall performance across all benchmarks, it can display considerably suboptimal performance under individual benchmarks. The results also show that for all the benchmarks, other than RayTracer, either Core-A or Core-B performs better than Core-U. In addition, for all benchmarks, other than the biometric benchmark (Blast), either Core-A or Core-B performs worse than Core-U.

Core-selectability allows for the user to dynamically pick and choose the employed core design. Thus, a two-way core-selectable design that employs Core-A and Core-B, will be able to perform better than Core-U across almost all benchmarks. However, for the benchmark RayTracer, this solution will perform 20% worse than Core-U alone. This highlights the importance of good design space exploration for the employed cores. Ideally, the core designs should provide higher performance than the best single design to *collectively exhaustive* subsets of the workload space. In addition, the more *mutually exclusive* the subsets are, the more potential there will be for performance gain.

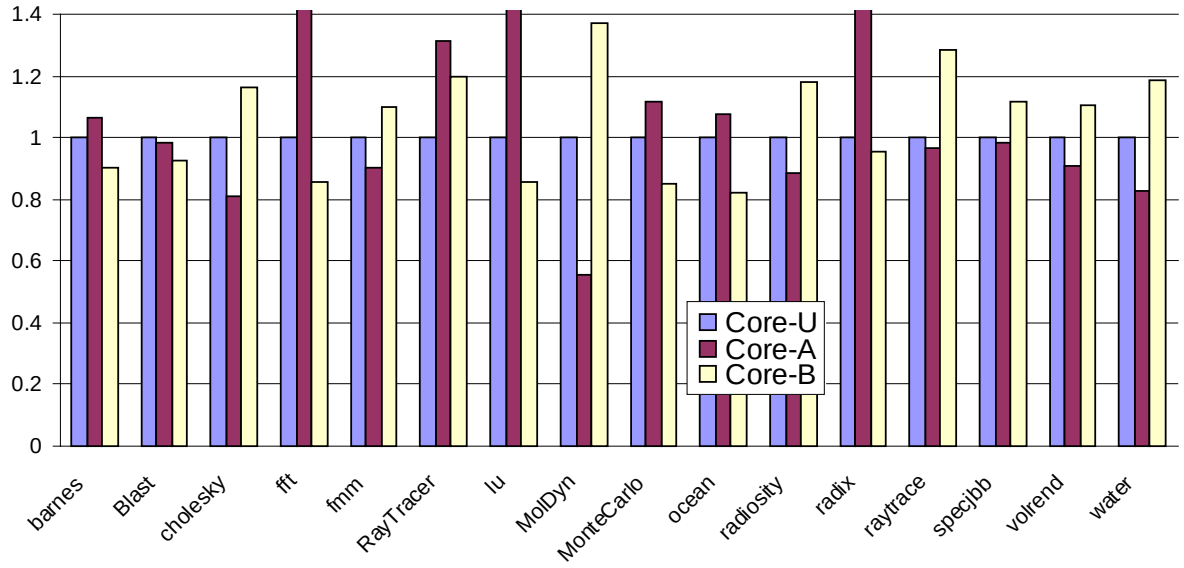


Figure 4-2: Execution time of each benchmark on each core design normalized to the execution time on Core-U.

The benchmarks for which Core-A and Core-B provide higher performance than Core-U are almost mutually exclusive and almost collectively exhaustive. Nevertheless, it may be of importance that absolutely none of the benchmarks display lower performance than the

baseline design (even if average performance is improved). If that is the case, the core design space needs to be explored more rigorously. One option is to increase the number of selectable cores, and employ one that is customized for RayTracer. A more straightforward solution is to employ Core-U as one of the selectable core designs.

Figure 4-3 shows the average execution time for three combinations of two-way core-selectable designs normalized to the execution time of the baseline design, Core-U. These results show that the best overall performance for two-way core-selectability is attained with Core-A and Core-B, showing an average performance enhancement greater than 10%. Core-selectability between Core-U and Core-B, will not display performance lower than the baseline design on any benchmark. But it attains a smaller overall performance enhancement, of close to 7%, compared to Core-U alone.

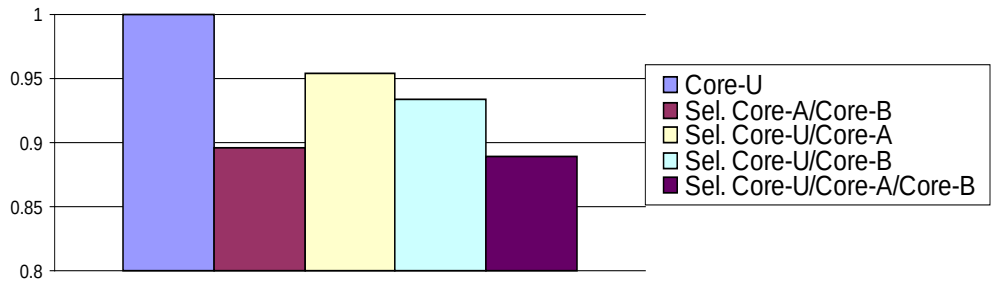


Figure 4-3: Execution time of core-selectability with different combinations of cores, normalized to that of Core-U.

Another option is core-selectability between all three core designs (Core-A, Core-B and Core-U). The average decrease in execution time for this scenario (also shown in Figure 4-3) is over 11%, while not displaying performance less than the baseline design under any of the benchmarks. However, increasing the number of selectable cores will increase the overhead in propagation delay.

4.3.1. The Overhead of Selectability

As discussed in the implementation section, core-selectability can introduce minor overheads to the front-end and back-end of the core designs.

Accounting for the multiplexing of address signals to the L1 data cache and the extra wire load, the propagation delay in the back-end was found to not be increased by more than 26ps. This delay is conveniently less than the slack observed in the L1 cache access time for all core designs. Although this cannot be generalized to an arbitrary design, cache access time does vary in steps (with the associativity or number of sets) and the optimal core clock period is dependent on many microarchitectural parameters.

We do not repeat such an evaluation here, as the effect of increased propagation delay in the front-end has been studied elsewhere, and the core designs considered here are not the very best for core-selectability anyway.

4.3.2. The Source of Performance

We investigated the source of performance enhancement by looking at the code of the benchmark that displays the largest performance gain from a customized core design, MolDyn (from Java-Grande). This benchmark simulates molecular dynamics. The main program loop of the application performs force calculations between only particles that are within a certain distance of one another. There is little local ILP within each iteration of the main loop of the application. But with a large enough window it is possible to reach future iterations [65], which are mostly independent. It is for this reason that the application gains such large performance through increasing the issue-queue size at the cost of narrower issue width.

One may argue that this form of distant parallelism may be better extractable with simultaneous threads [23]. But we argue that the performance gain of core-selectability is orthogonal to that of simultaneous multithreading. This is because, even with simultaneous multithreading (SMT), upsizing the issue-queue will yield better performance with such a benchmark (compared to the best design across all benchmarks), as it will enable the extraction of parallelism within individual threads. Of course, this is assuming that the scalability of the application is not such that SMT just happens to finish off all remaining parallelism. However, due to the lack of simultaneous multithreading in the OPAL simulator, we were unable to quantitatively verify this.

Although there is thread-level parallelism in this application, it has been shown that it is difficult to extract without speculation support [65]. Core-selectability enables extraction of this parallelism without burdening the design components that affect general performance.

4.4. Multiprogrammed Results

At a higher level, core-selectability can also be viewed as providing selectability between a homogeneous or heterogeneous multi-core design.

As pointed out in the related work section, it has been shown that a heterogeneous multiprocessor design can provide better throughput in a multiprogrammed environment. But, it has also been shown that heterogeneity can degrade the performance of multithreaded workloads. Therefore, a design that can transform between heterogeneity and homogeneity will have a degree of robustness in performance that is unachievable by other design solutions. In this section, we investigate the potential performance benefit of such robustness.

Note that, from a stochastic standpoint, what renders multithreaded applications unsuitable for heterogeneity is the fact that they cause tasks (i.e., threads) with the same workload behavior to arrive at the system in bursts. This is opposed to the more normal distribution of task arrival in a multiprogrammed environment (see [37] for a more detailed study of the importance of accurately accounting for the task arrival pattern).

4.4.1. Methodology

In order to stochastically model a multiprogrammed environment, we simulate the queuing and occupation of different processing cores for different workload types. Tasks of different workload types are generated according to a random process with a normal distribution.

Tasks are primarily placed in the dedicated task queue of the core with the most suitable microarchitecture for the task's workload type. If the most suitable core is occupied, the task is directed to the next best core. If all cores are in use, the task waits for the most suitable core to become available. When there are cores with the same architectural configuration in the system (homogeneous), tasks are randomly assigned to them based on availability. Once the core is free, the task at the head of the task queue is consumed by the core for a given amount of time.

The amount of time it takes each task to be executed by a specific core depends on the design of the core and the task workload behavior. In this analysis, the workload behavior that a task may display is limited to that of the Simpoints [109] of the integer SPEC2000 suite of benchmarks. The considered core designs are the same three derived in Section 4.2.3 (Core-U, Core-A and Core-B), but simulated with *sim-mase* from the Simplescalar V4.0 toolset [27].

All tasks are considered to consist of 3.2 billion instructions no matter what the workload type. The amount of time a core is occupied by a task is determined by the rate with which the task's workload type is executed on the microarchitectural configuration of that core. As an example, a task that executes on a specific core design at a rate of β instructions per nanosecond is executed on that core in 3.2÷β seconds.

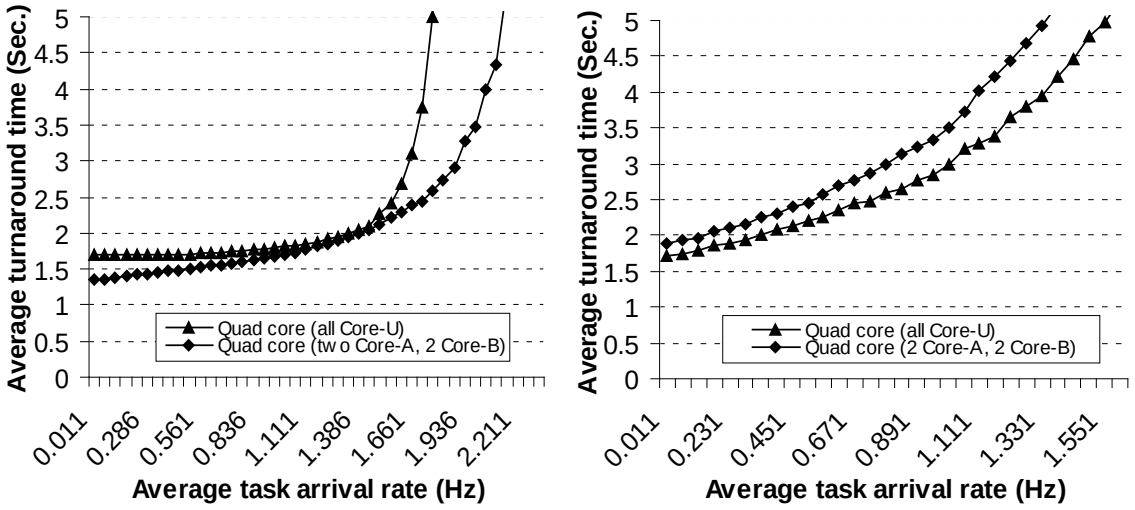
4.4.2. Evaluation Results

In order to evaluate the potential of core-selectability under different task arrival patterns, we compare the task turnaround time of systems with different combinations of core designs. Figure 4-4.a displays the average turnaround time of tasks submitted to two such quad-core systems. One system is homogeneous, and consists of four cores of the Core-U design. The other is heterogeneous, and consists of two cores of the Core-A design and two cores of the Core-B design. Results are presented across the spectrum of the task arrival rate, from low-contention to saturation. Tasks of different workload types arrive independently of each other. Thus, this task arrival pattern can be considered to be more representative of a multiprogramming environment.

These results show that the heterogeneous design results in around 25% lower task turnaround time in low arrival rates compared to the homogeneous design. Moreover, it displays roughly 14% higher execution bandwidth (the task arrival rate at which task turnaround time increases unboundedly). Therefore, employing 2-way core-selectability can provide such performance enhancement to multiprogramming environments. This is while, contrary to a fixed heterogeneous design, core-selectability does not degrade the performance of multithreaded applications, as it can switch back to a homogeneous design (and even

provide higher performance than a fixed homogeneous design, as observed in the prior section).

For comparison, Figure 4-4.b displays the average turnaround time of tasks submitted to the same two differently designed quad-core systems. Here, however, tasks of different workload types arrive in bursts of four tasks of the same workload type. Thus, this task arrival pattern can be considered to be more representative of a multithreaded environment. These results show that under this task arrival pattern the heterogeneous design results in 10% higher task turnaround time in low arrival rates, and lower execution bandwidth, compared to the homogenous design. A core-selectable design enables the system to transform into the best quad-core solution for the task arrival pattern at hand.



(a) ***(b)***

Figure 4-4: Average task turnaround time for (a) normal traffic, and (b) bursty traffic.

4.5. Discussion

4.5.1. The Limitations of this Study

The evaluation results presented here are by no means conclusive enough to place a solid verdict on the notion of core-selectability, as the results may potentially be biased by some subtle circuit-level details. Nevertheless, we believe that in conjunction with the qualitative merits outlined here, a compelling enough argument emerges to warrant further investigation of this technique.

It is also important to note that the results presented here do not represent an upper bound on the performance enhancement attainable from core-selectability. For instance, the customized core designs we consider display limited diversity (for demonstration purposes and limitations in simulation). In actual implementation, not only can the cores differ in their microarchitectural parameters, but also in fundamental functionality and even ISA.

A chief benefit of core-selectability is the fact that it allows for the instruction-level performance enhancement of different types of workload behavior to be separated from each other. This is an aspect of core-selectability that we do not quantitatively evaluate here, as it pertains to design and verification effort, which is challenging to quantify.

We do not present results for a head-on performance comparison of core-selectability with the option of using the die area of the added cores for other purposes, e.g., more cache. Although such comparison has become commonplace in the academic arena, we believe that the assumption that it is die area that limits the sizing of structures is out-dated. If added cache were to be of performance benefit, it would already be provisioned in a well-designed baseline system. If it is not, it is because the increase in access latency would outweigh the

benefit. Note that in the evaluations caches equivalent to those of modern processors are employed.

What does need to be evaluated in future work is the potential of core-selectability in the presence of simultaneous multithreading for multithreaded workloads. Although core-selectability is aimed at enhancing the extraction of ILP, a portion of this parallelism may be extractable through fine-grain threading.

An intriguing twist, that is not investigated here, is to employ core-selectability in conjunction with adaptable caches and cores with different clock domains. This can allow for much broader diversity in the design of the selectable core designs without introducing slack to the cache accesses.

4.5.2. Future Trends and Potential Drawbacks

Core-selectability exploits the increasing trend of transistors available on dies. It also does not exacerbate the trend of increasing power consumption and verification effort that alternative approaches to performance enhancement entail. This is because it enables the separation of the circuitry and design complexity necessary for dealing with different types of workload behavior, without drastically adding to the interconnection complexity.

There are two main potential drawbacks to the implementation of core-selectability. One is the added cost of engineering multiple core designs. The effort of designing a single core that has been tweaked to attain high performance across a wide range of applications may turn out to be less than that of designing multiple cores that are customized to specific workload behaviour, although not tweaked as much.

The other potential drawback is the added propagation delay at the shared ports to the system. Although with the technology library and the detailed baseline architecture employed in the evaluations of this study, the extra propagation delay was of negligible impact, it is not out of the question that it may be problematic in other settings. Whether the performance gain of core-selectability exceeds the potential loss due to increased propagation delay at the shared ports is a question that needs to be addressed in the context of the characteristics of the technology and development setup in which it is to be implemented.

4.6. Chapter Summary

This study investigates a potential solution to increasing the utilization of existing provisioning in the cache and interconnection resources of a chip multiprocessor. The approach is to place a number of differently designed cores (with different ILP-extracting units) within each node, and provide the ability to select which core to use depending on the characteristics of the applications at hand.

With the technology library considered in this study, it is shown that employing this technique with two core designs that focus on different ILP behavior, can result in better performance across a wide range of parallel applications, compared to a conventional design employing the best core design for overall performance across the same benchmarks. It is also shown that this design solution can provide greater throughput under multi-programmed workloads, by enabling the system to transform into a heterogeneous design when needed, i.e., providing selectability between homogeneity and heterogeneity.

CHAPTER 5: Architectural Contesting

A major impediment to the effectiveness of microarchitectural techniques is their high dependence on the workload behavior. It is for this reason that previous studies have proposed techniques that enable the employed microarchitecture to dynamically change and become more suitable for the immediate workload behavior. Such techniques can be broadly categorized as either *adaptational* or *migrational* approaches. Adaptational approaches are based on a single processor design with adjustable design features (e.g. [15]). Migrational approaches are based on a number of differently designed processing cores, *i.e.*, a heterogeneous multi-core (e.g. [23]).

Regardless of the approach, the rate at which the employed architecture can be effectively changed depends on the rate at which 1) change in workload behavior can be *detected*, 2) the most suitable architecture for the new code region can be *determined*, and 3) the change can be *performed*. The challenge with adaptational techniques is in determining when and how the architecture should change. Similarly, the challenge with migrational techniques is in determining when and to which core execution should be transferred.

In this chapter, we show that the speed of adjusting to change in workload behavior that is essential for high performance enhancement, is too fine-grain to be achieved with prior approaches. However, the availability of multiple cores can be exploited to enable the speedy transfer of execution to the most suitable architecture. In the proposed approach, code is simultaneously executed on a number of cores, each architected for optimum performance under a different class of workload behavior. With each core broadcasting its instruction

results to the other cores, completion of instructions can be expedited in cores that are not suitable for the immediate code region. Thus, upon change in the workload behavior, the core that is most suitable for the new workload behavior will be able to automatically take the lead. In other words, detecting changes in workload behavior, determining the best architectural configuration, and transferring execution to that configuration, all take place automatically and fluidly with minimal latency. We refer to this technique as *architectural contesting* (or simply *contesting*).

Contesting is orthogonal to other sources of single-thread performance enhancement, as it exploits a unique source of performance enhancement, namely, fine-grain customization. Moreover, like other redundant threading architectures, it can be employed on a need-to-have basis, providing robustness in how resources are employed (throughput or single-thread performance) and how performance and power are balanced.

Below are some highlights from the results and analysis presented in the following sections:

- 2-way contesting yields an average speedup of 15% (maximum speedup of 25%) over a benchmark's own customized core.
- For most benchmarks, most of the performance enhancement of contesting comes from heterogeneity in the microarchitecture, although the benefit of heterogeneity in the L2 caches is noticeable in a few cases. For both sources, it is contesting that enables this heterogeneity to be exploited at a fine granularity.
- The speedup of contesting is even more pronounced in constrained heterogeneous CMPs: contesting yields an average speedup of 22% compared to executing the benchmark on the most suitable available core. The availability of fewer available core types reduces the extent

to which application-level heterogeneity can be exploited. Contesting compensates for this deficit by additionally exploiting fine-grain switching within applications.

- Compared to the best homogeneous CMP design, a constrained heterogeneous CMP design achieves an average speedup of 11% without contesting and 34% with contesting. In other words, contesting triples the single-thread performance advantage of heterogeneity in this system.

- Compared to the best homogeneous CMP design, contesting between only two core types yields the same or higher single-thread performance enhancement as executing on the best of three core types. Therefore, in terms of maximizing single-thread performance, contesting may be a more cost-effective approach than increasing the number of core types in the CMP.

- The design of a constrained heterogeneous CMP involves compromises, by virtue of limiting the number of core types and optimizing for one figure of merit or another. Contesting provides a degree of performance robustness that compensates for certain side-effects of these compromises. For example, while a constrained heterogeneous CMP design improves single-thread performance for the benchmark suite as a whole compared to a homogeneous CMP design, specific benchmarks may nonetheless perform worse. Contested-execution of these benchmarks makes up for this performance deficit.

5.1. Motivation: The Speed of Change

For each SPEC2000 benchmark we evaluate the ability to switch execution between two microarchitectural configurations. The configurations are chosen from among eleven configurations, each customized for one of the SPEC2000 benchmarks. They were extracted

through a simulated annealing exploration process for 70nm technology (see Section 3.4 for further details).

The execution of each benchmark's 100-million instruction Simpoint [74] was simulated on the customized configuration of each benchmark and the number of cycles to retire every 20 dynamic instructions was logged. Then, for each benchmark and every combination of two configurations, every 20-instruction region was considered to be retired at the rate of the faster of the two for that region – while factoring in the clock periods. The time spent in each region was then aggregated to determine the total execution time and from that the best two configurations for each benchmark. The same process was repeated for 40-instruction regions, by summing the execution time of neighboring 20-instruction regions. The whole process was repeated for regions of up to 83 million instructions.

Figure 5-1 illustrates the speedup attained for each benchmark over the performance of its own customized architecture by switching execution between two core configurations at different rates. Also indicated in these graphs are the two configurations that provided the best speedup (at the finest granularity). The different data-point symbols indicate different two-core combinations. While the best pair of cores for switching execution between rarely varies across different granularities for benchmarks such as *bzip*, it is highly dependent on the granularity of switching in benchmarks such as *crafty*. At the coarsest granularity (i.e. the whole Simpoint), each benchmark achieves its best performance on its own customized configuration and attains no speedup.

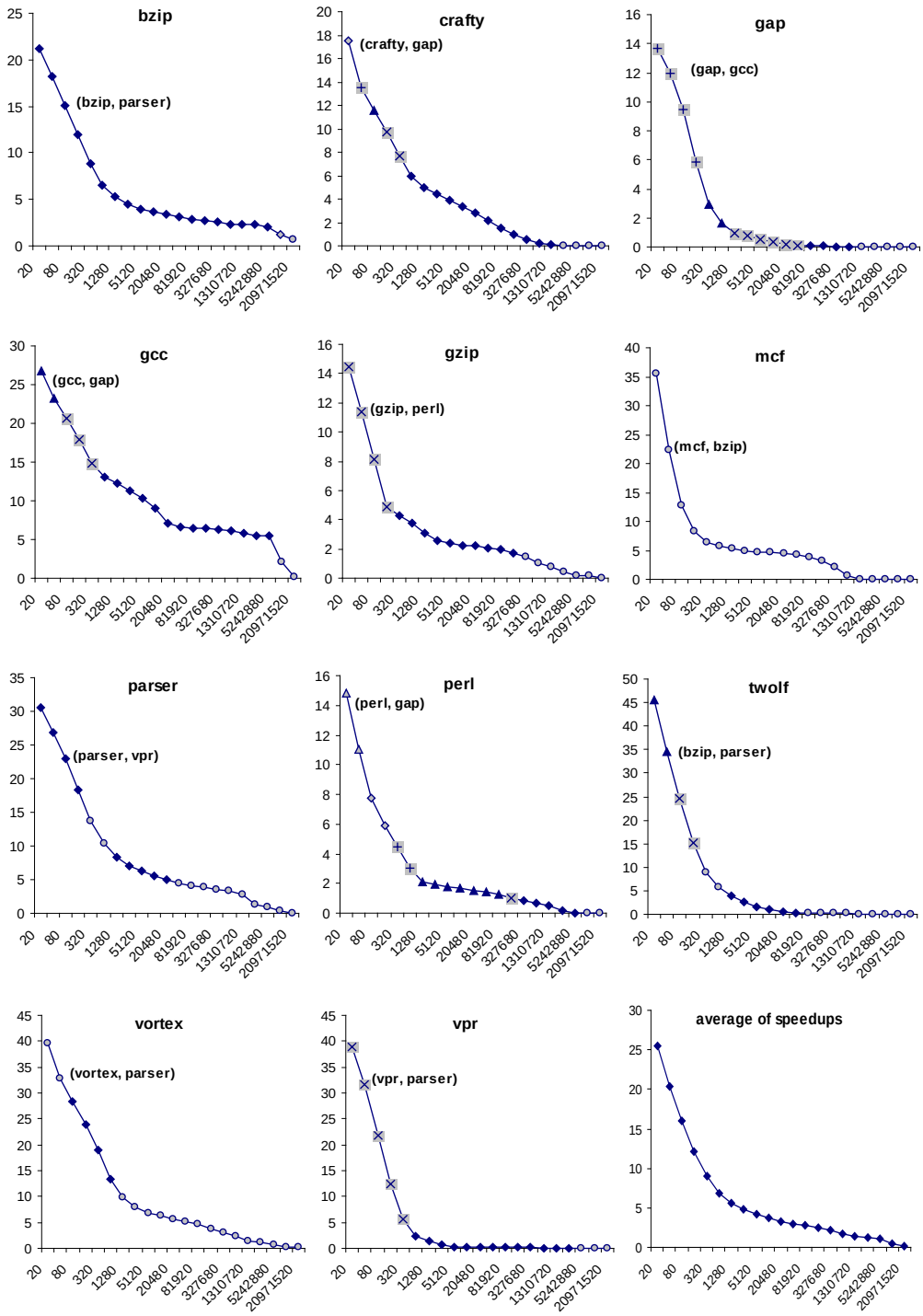


Figure 5-1: Percentage speedup of switching execution between two different configurations at different granularities, normalized to the performance of each benchmark's own customized configuration.

These results illustrate that the greatest potential of being able to dynamically adjust the microarchitecture to the workload is attainable at granularities of less than a thousand instructions. While the benchmarks *gcc* and *gzip* attain a modest portion of their maximum speedup in coarser granularities, most benchmarks display little or no performance enhancement with coarser switching of the microarchitecture. The knee in the curve in most of these benchmarks occurs near the 1280-instruction granularity. For instance the graph for average speedup displays a mere 5% speedup for granularities in this range, while displaying up to ~25% speedup for finer granularities. Previously proposed approaches to dynamically adjusting the architecture to the workload are unable to exploit such fine-grain change in workload behavior.

In most cases the customized microarchitecture of a benchmark is among the best two cores to switch execution between. However, for *twolf*, the benchmark that attains the largest fine-grain speedup, the customized architectures of *vortex* and *parser* are the best two. This is notable as an application-level customized architecture is forced to compromise performance in fine-grain regions in order to attain good overall performance. This infers that switching execution between architectures that are custom designed for applications may not necessarily provide the best performance enhancement from fine-grain switching. Nevertheless, these architectures are good candidates for improving application-level performance and multi-programming throughput – issues of general importance in a CMP design.

5.2. Related Prior Work

The *slipstream* paradigm [58] employs two simultaneous execution streams of the same code that interact to improve overall single-thread performance. One execution stream is expedited through speculatively skipping ineffectual work, but needs to be checked by a redundant stream. However, the redundant stream itself is also expedited as the speculative stream passes it highly accurate branch and value predictions. More recent related work is the *Paceline* leader-checker microarchitecture [11]. In this approach a leader-core runs the thread at a higher-than-rated frequency, while passing execution hints and prefetches to a safely-clocked checker core.

In both these techniques, the leading core is fixed (Paceline occasionally swaps cores' roles for temperature control) and is expedited in a manner that needs to be checked for correctness, thus the need for forwarding instruction results to a checker. In contesting however, the leading core varies depending on the workload behavior, and gains lead purely because it is more suitable for the immediate region of code. Thus, there is no need for it to be checked. Instruction results are forwarded to the other cores not for checking, but rather to keep them from falling behind so they can take lead as swiftly as possible when the workload behavior changes.

The *Datascalar* paradigm [17] enables single thread performance enhancement through enabling the distribution of the program data-set across the local memory of multiple cores.

Frequency scaling techniques [107] provide variability in the performance-power tradeoff.

5.3. Implementation Details

The implementation of a contesting system resembles other redundant-execution leader-follower architectures, such as Slipstream [58], SRT [100], AR-SMT [28], Paceline [11], Datascalar [17], etc. The main novelty of contesting is not in the employed mechanisms, but rather the purpose for which they are employed. This section describes the implementation of a contesting system. While the description is generalized for N-way contesting, the following results are for a 2-way contesting system.

5.3.1. Leveraging Results from Other Cores

Figure 5-2 illustrates an architectural contesting multi-core system. The different cores concurrently attempt to execute the same code. Each core broadcasts the results of its retired instructions to the other cores via a dedicated *global result bus* (GRB). A core receives results from incoming GRBs into *result FIFOs*.

A core maintains a counter in its fetch unit that is incremented for each fetched instruction (not shown in Figure 5-2). Likewise, the core maintains a counter per result FIFO that is incremented for each received result. These locally generated counts enable determining whether instructions in the receiving core are ahead of or behind a result FIFO in terms of logical position in the dynamic instruction stream.

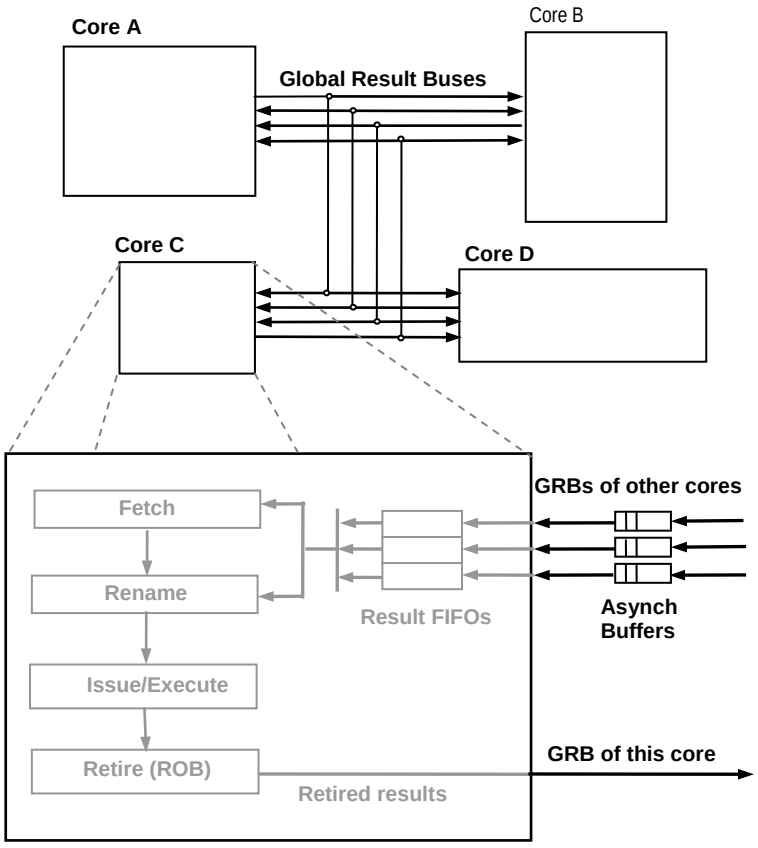


Figure 5-2: A generalized architectural contesting multi-core system.

If the fetch counter and a result FIFO counter match, the receiving core is unequivocally trailing the sending core, and the result is popped from the result FIFO and used in lieu of executing the instruction. The instruction is completed early in the fetch stage (if it is a branch) or the rename stage (if it is a register-producing instruction). Early completion in the fetch stage is implemented by overriding the branch prediction logic. Early completion in the rename stage is implemented by directly writing a value into the destination physical register. On the other hand, for as long as a result FIFO counter is less than the fetch counter, results are popped and discarded. However, an incoming branch result is not automatically discarded. If the count of the incoming branch result matches the fetch count of an

unresolved branch in the core (the branch's fetch count is saved in its checkpoint), the unresolved branch is resolved early. Whether a mispredicted branch is resolved early (using a result FIFO) or normally (using local execution), the fetch counter is restored to the fetch count of the mispredicted branch, which was saved in its checkpoint. Wrap-around of the counters is not a problem because the maximum count value is larger than the maximum allowed distance between leading and lagging cores.

By leveraging the result FIFOs in this way, execution in the lagging cores will never fall too far behind. Thus, when the code phase changes, all the cores will be contested fairly in the new phase without the need for actually detecting the change of phase, and the core that is best suited will automatically be able to *take lead*.

How far behind a lagging core is depends on the physical propagation delay between cores. When the characteristics of the code change, it is this *lagging distance* that a core needs to catch-up on before it can become the head of the pack and commence effective execution. Note that it is not necessary for all the lagging cores to receive the result of a retired instruction in the same cycle. This issue is of convenience, as different cores may be at differing distances from each other.

Although the frequencies and retirement widths of the cores may differ, the peak rate at which instruction results are retired by any core – in instructions-per-second (IPS) – must be sustainable by all other cores. That is, the peak retirement rate (in IPS) of any core must be less than or equal to the peak rate (in IPS) at which instruction results can be written to the register file and memory in any other core. Without this condition, a lagging core may unboundedly fall behind, resulting in excessive catch-up time and therefore defeating the

purpose of architectural contesting. We refer to a lagging core that cannot keep-up with the leading core as a *saturated lagger*. This scenario can be dealt with simply by disabling contesting mode for the saturated lagger.

5.3.2. Handling Stores

Stores are redundantly performed in the private cache levels of the cores. The private cache levels are configured to use the write-through policy to simplify contesting (this does not preclude using the write-back policy in non-contesting modes). To prevent lagging cores from incorrectly observing future stores of less-lagging cores, stores stop short of writing through to the shared cache level. Here, we employ a synchronizing store queue similar to SRT's store queue [100].

In SRT, the store queue waits for both instances of a store (the leading and trailing threads' instances), before performing a single merged instance to the L1 cache, and loads from the leading thread search the store queue in addition to the L1 cache. Similarly, the synchronizing store queue employed for contesting buffers stores and keeps track of which cores have privately performed each store. When the oldest store has been privately performed by all cores, a single merged instance is performed to the shared cache level.

5.3.3. GRB Implementation

Each core's GRB is pipelined to sustain the maximum throughput of retired instructions from that core. Also, since different cores may have different clock frequencies, asynchronous buffering is employed at the receiving cores to synchronize communication. This asynchronous buffering, unlike that in a GALS architecture [93], will not affect fixed-

configuration performance as it is placed between processors not between different units of a single processor.

The GRBs form the main artery of a contesting system. They determine the point-to-point latency between cores, and therefore the lagging distance of cores depending on which one is the leader. Therefore, optimizing the design of the GRBs (and the core layout) is critical to the performance enhancement of contesting.

5.4. Measuring Up to the Very Best

5.4.1. Methodology

The *sim-mase* simulator from the Simplescalar V4.0 toolset has been modified to model the contesting implementation described in Section 5.3.

The simulator was modified to enable time-synchronous execution of multiple simulator instances that model different proportional clock periods. In order to model time-synchronous execution, the simulator instances perform handshaking in a round-robin arrangement. Receiving a handshaking signal signifies the passing of a base time-unit (specifically 0.01ns). Each simulator instance executes an iteration of its top-level simulation loop upon the passing of as many time-units as there are in the clock period it is modeling. For example, a simulator instance modeling a 3GHz core will execute one iteration of its top-level simulation loop every 33 time-units.

The benchmarks used throughout are the 100-million instruction Simpoints [74] of the SPEC2000 integer benchmarks (except for *eon*, which we were unable to compile with the Simplescalar compiler).

In this study, we consider the pool of prospective cores in the heterogeneous CMP to consist of cores that are customized for individual SPEC2000 integer benchmarks in 70nm technology. We used the XP-Scalar design-space exploration framework (described in Chapter 3), which employs a simulated-annealing exploration process, to arrive at the benchmark-customized cores. XP-Scalar varies multiple design parameters, including superscalar width, register-file/ROB size, issue-queue size, load-store queue size, L1 and L2 cache configurations, and clock frequency. The depth of pipelining of various architectural units/stages is consistent with the processor's frequency and the complexity of these units/stages. The customized core of each benchmark and its performance with respect to all benchmarks are the same as those documented in Table 3-5.

5.4.2. Results

We limit our evaluation to 2-way architectural contesting (contesting between two cores). An issue of importance is the considered core-to-core latency, or the time it takes for an instruction result to travel from one core to another. In this part of the study, a one nanosecond (three cycles of a 3 GHz processor) core-to-core latency is initially considered. The effect of scaling this latency is later evaluated.

Figure 5-3 shows the performance (instructions per time, IPT) of contesting, for each benchmark. For each benchmark, the two cores that are contested (from among all benchmark-customized cores) are those two which give the highest performance when contested; the pair of contesting cores used by a given benchmark is labeled above its bar in the graph. For comparison, the IPT of each benchmark on its own customized core is also shown. Contesting yields an average speedup of 15% over a benchmark's own customized

core. The largest speedup is attained for the benchmark *gcc* at 25%. Four out of the eleven studied benchmarks attain more than 18% speedup.

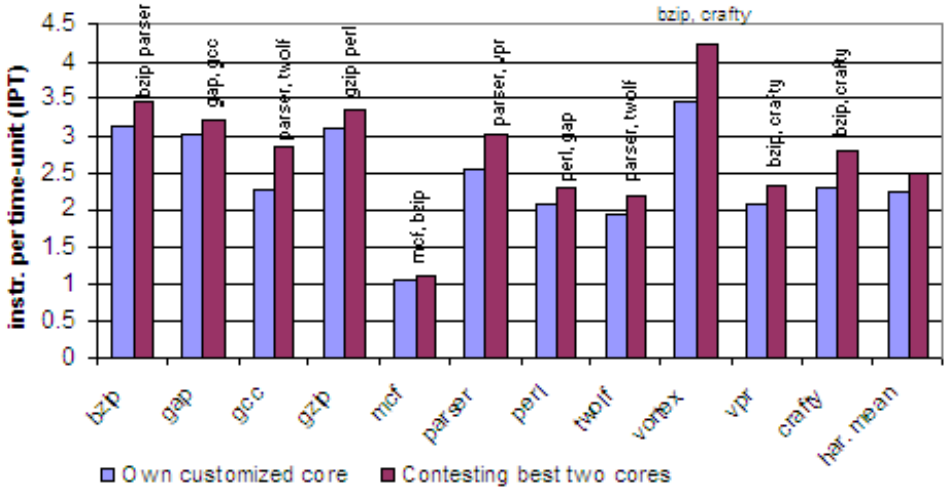


Figure 5-3: IPT of each benchmark for 1) execution on its own customized core and 2) contesting between two cores that maximize contested-execution performance (the two contested cores are shown above the bars).

The averaged results in Figure 5-1 show that achieving speedups in the range of 15% over a benchmark's own customized configuration requires the ability to switch execution between configurations at a rate of around 100 instructions. This number of instructions is proportional to the number of instructions in the pipeline of an average configuration at any instance. However, using previously proposed techniques to detect changes in workload behavior, determine a suitable configuration, and transfer execution to it, can probably be achieved at a rate of a few thousand instructions at the very best – which drastically diminishes the benefit of being able to adjust the microarchitecture.

5.4.2.1. The source of performance enhancement

A question that may arise is how integral heterogeneity in the microarchitecture of the cores is to this performance enhancement, and whether the origin is heterogeneity in the cache configurations. Differentiating the heterogeneity in the caches from that in the microarchitecture of the cores can provide insight into the origin of the performance enhancement. However, note that the best cache configuration for a workload is not independent of other microarchitectural design factors.

In order to address this question, each benchmark is executed with contesting between two cores that differ only in their L2 caches. One of the cores is one of the best two cores for contesting. The other is the same core, but with its L2 cache (configuration and access latency) replaced with that of the other best core for contesting. For example, *bzip* was originally contested between the customized cores of *bzip* and *parser* (contesting these two cores yielded the highest performance). For the modified experiment, *bzip* is contested between two *bzip* cores, except that one of these otherwise identical cores has the L2 cache of the *parser* core. This experiment is repeated with two *parser* cores, one of which has *bzip*'s L2 cache. The higher performing trial of these two trials is used.

Figure 5-4 shows the speedup of contesting. The total height of each bar represents the speedup of contesting in the original experiment (heterogeneity in both the microarchitecture and L2 cache). The bottom fraction of each bar represents the speedup of contesting in the modified experiment, isolating the performance enhancement due to heterogeneous L2 caches. These results show that, for most of the benchmarks (other than *gcc* and *parser*), only a minor portion of the performance enhancement can be attributed to only heterogeneity in

the L2 cache. All the same, it is contesting that enables this heterogeneity to be exploited at a fine granularity.

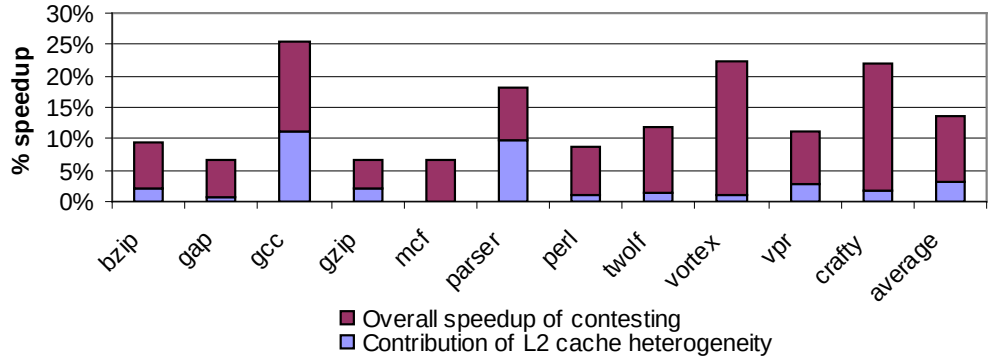


Figure 5-4: Isolating the contribution of L2 cache heterogeneity to the performance enhancement of contesting.

5.4.2.2. The effect of core-to-core latency

Figure 5-5 shows the effect that the latency of transmitting instruction results from core to core has on the average speedup of contesting between the best two cores for each benchmark over its performance on its own customized core. These results show a decrease in the performance enhancement of contesting as this latency increases. At a latency of around 100ns the performance benefit reduces to around 6%. These results show the importance of the speed of the GRB.

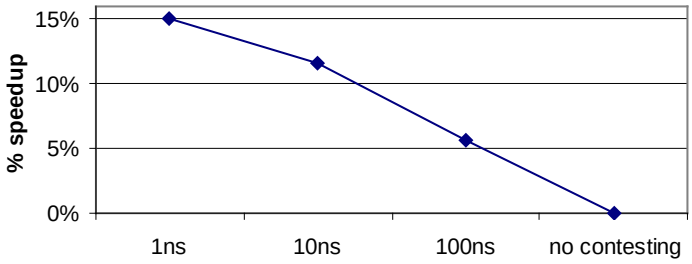


Figure 5-5: The average speedup of contesting over customized cores for different core-to-core latencies.

5.5. Evaluation with Limited Core Types

Section 5.4 evaluated the performance enhancement attainable from contesting between the best two core types for fine-grain switching, for each benchmark. Since the best pair of contesting cores differs from one benchmark to the next, the previous evaluation implies that the customized core types of all benchmarks are available in the heterogeneous CMP. However, there may be fewer core types in a realistic heterogeneous CMP. In Section 5.5.1, we first address general heterogeneous CMP design and what influences the best set of core types to employ when the number of core types is limited. In Section 5.5.2, we apply these principles to design heterogeneous CMPs with only two core types. Finally, in Section 5.5.3, we evaluate the performance enhancement of contesting between the cores of the systematically designed dual-core heterogeneous CMPs.

5.5.1. The Design Goal

The best combination of microarchitectural configurations to employ in a heterogeneous system depends on the design goal.

If the design goal is to minimize the total execution time of a set of benchmarks when submitted to the system one-by-one – as is customary in single-core microarchitecture evaluation – a representative figure of merit is the harmonic-mean of the performance (instructions per time unit, IPT) of all benchmarks when each is executed on the most suitable core available. This figure of merit is improved if the benchmarks are weighted by the frequency with which they occur in the system. Without these weights, benchmarks that run infrequently but have long run-times may have disproportionate influence on the perceived-best core types.

Benchmark weights may not be available, however. In this situation, it may be desirable to use the average (arithmetic-mean) of IPTs as the figure of merit. Average IPT focuses on raw throughput instead of total time, which may lead to more performance-robust core types in the face of uncertain benchmark frequencies.

Neither of these metrics (harmonic-mean IPT and average IPT) accounts for core-contention between jobs and may thus bring about imbalance in the number of benchmarks better suited for the different core types. For example, if the design goal is simply to maximize either the harmonic-mean IPT or average IPT for ten benchmarks on two core types, the ultimate CMP design may be guided toward one core type that is favored by nine benchmarks and one core type that is favored by one benchmark. While this CMP design is optimum when there is no contention between jobs, it is not necessarily optimal when there is contention.

If contention between jobs is a concern, the best combination of core types to employ in the system is influenced by 1) the rate and distribution of job submissions and 2) how jobs are scheduled. Below, a figure of merit is developed that accounts for contention based on two simplified assumptions regarding job distribution and scheduling, respectively. Firstly, we consider a uniform distribution of job submissions. That is, jobs have an equal probability of belonging to one of the considered workload types (*i.e.*, one of the benchmarks). Unevenness in the distribution can be modeled by assigning importance weights that are proportional to the probability of a workload type being submitted to the system. Burstiness in the arrival of jobs of the same workload type decreases the value of heterogeneity. Secondly, we consider a scheduling policy that directs a job to the core type for which it is best suited, even if all

cores of that type are currently busy and the job must be queued, instead of directing it to the best *available* core. This policy is reasonable if all cores are heavily loaded.

In this setting, the arrival rate of jobs at the job-queue of a specific core will be proportional to the number of job types that prefer that core. Therefore, according to Little's law, the average number of jobs in a job-queue will also be proportional to the number of job types that prefer the corresponding core. Therefore, a representative figure of merit can be attained by dividing the performance (IPT) of each benchmark when executed on the most suitable core type in the CMP design by the number of benchmarks that share the core type, and then taking the harmonic mean. We refer to this figure of merit as the *contention-weighted harmonic-mean IPT*.

5.5.2. Exploring Combinations of Core Types

The best combination of core types to employ in a heterogeneous CMP is determined by searching all the possible combinations of core types for one that maximizes the considered figure of merit. Once again we limit the pool of prospective core types to the customized cores for individual SPEC2000 integer benchmarks. In addition, the number of core types in the heterogeneous CMP is limited to only two. This does not limit the total number of cores, that is, conceivably there could be multiple instances of each core type.

Since we separately consider three different figures of merit – average IPT (*avg*), harmonic-mean IPT (*har*), and contention-weighted harmonic-mean IPT (*cw-har*), as discussed in the previous section – we arrive at three different heterogeneous CMP designs, HET-A, HET-B, and HET-C, respectively. These designs are displayed in the first three rows of Table 5-1. The table shows which two core types comprise each of HET-A, HET-B, and HET-C. A core

type is identified by the name of the benchmark for which it is customized, *e.g.*, the customized core type for the *gcc* benchmark is named "*gcc*". HET-A is comprised of the *parser* and *twolf* core types, HET-B is comprised of the *gcc* and *mcf* core types, and HET-C is comprised of the *bzip* and *crafty* core types.

Since this chapter is ultimately concerned with single-thread performance (the benefit of contesting), regardless of the figure of merit used to arrive at a CMP design, the last column of Table 5-1 shows the harmonic-mean of the IPTs of all benchmarks when each is run on the most suitable core of a given design. Since HET-B was designed using this figure of merit to begin with, as expected, it has the highest harmonic-mean IPT among HET-A, HET-B, and HET-C.

Two other designs are included in the last two rows of Table 5-1. The first of these, HOM, is a homogeneous CMP design with only one core type, namely, the core type that gives the best performance on average for all benchmarks. Of all the customized core types, the *gcc* core type is the best overall (it maximizes both average IPT and harmonic-mean IPT). The last design, HET-ALL, is a heterogeneous CMP comprised of all the customized core types, so that each benchmark runs on its customized core.

While the *gcc* core is the best individual performer among the benchmark-customized cores, it is possible for there to be an even better core that is explicitly customized for the benchmark suite as a whole. We conducted an exploration of the design space for the aggregate performance across all benchmarks using the XP-Scalar exploration process. The customized core that was attained provided negligible overall performance enhancement over that of the *gcc* core.

Table 5-1: Five CMP designs and their performance.

CMP design	Designed based on which figure of merit?	Constituent core types	Harmonic-mean of IPT
HET-A	avg	parser & twolf cores	1.76
HET-B	har	gcc & mcf cores	1.88
HET-C	cw-har	bzip & crafty cores	1.87
HOM	avg or har	gcc core	1.57
HET-ALL	Not applicable	customized cores of all benchmarks	2.1

The performance results in the last column of Table 5-1 show that, when exploited at the application level, unconstrained heterogeneity can provide up to a 34% increase in harmonic-mean IPT (HET-ALL compared to HOM). With only two core types, HET-C provides a 19% increase in harmonic-mean IPT (HET-C compared to HOM). In other experiments, not shown here, we determined that the overall performance of a heterogeneous CMP with four core types is within 2% of the performance of HET-ALL.

Figure 5-6 displays the IPTs of individual benchmarks on the five CMP designs of Table 5-1. For a given CMP design, a benchmark is run on the most suitable core type available in that design. These illustrate the importance of the available core types.

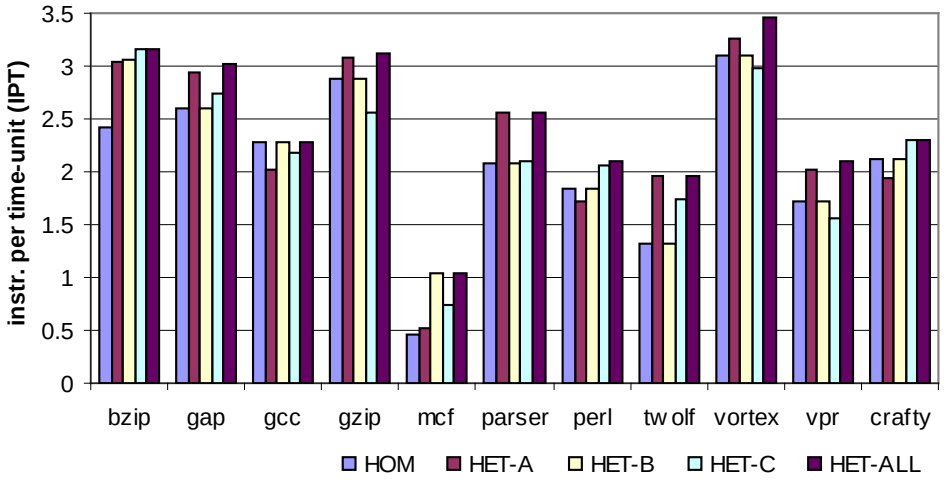


Figure 5-6: IPT for each benchmark when executed on the most suitable core type available in the CMP, for five CMP designs.

5.5.3. Evaluation of Contesting with Limited Core Types

In this section, we evaluate the single-thread performance enhancement of contesting on top of the HET-A, HET-B, and HET-C CMP designs from the previous section. The chief purpose of this exercise is to study contesting in the context of a heterogeneous CMP that was systematically designed to exploit application-level heterogeneity with a limited number of core types, and not explicitly for the purpose of fine-grain switching within an application. This is the expected setting in which contesting might be deployed.

Figure 5-7 shows the performance (instructions per time unit, IPT) of each benchmark, for three scenarios: 1) execution on HOM ("HOM"), 2) execution on the most suitable core type of HET-A ("HET-A, no contesting"), and 3) contested execution between the two core types of HET-A ("HET-A, contesting").

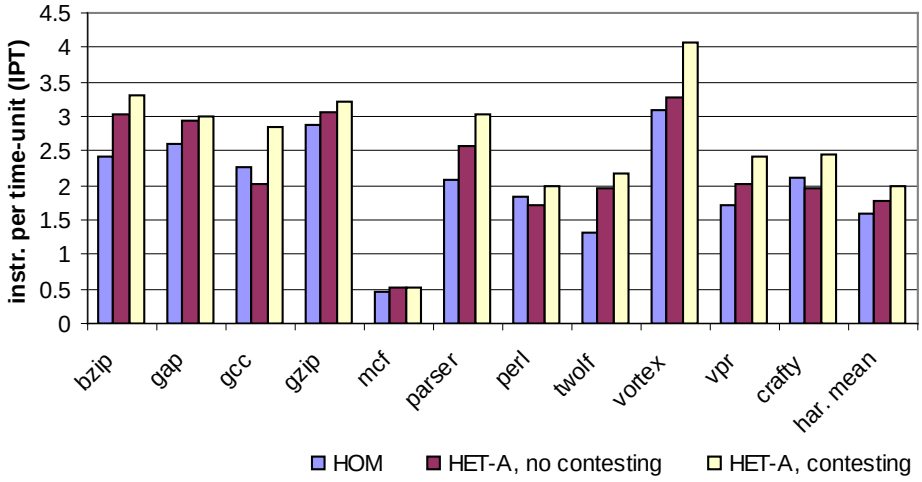


Figure 5-7: Performance of each benchmark on HOM, HET-A without contesting, and HET-A with contesting.

Contesting on HET-A yields an average speedup of 16% and a maximum speedup of 41% (for *gcc*) compared to not contesting. An interesting result is that, while the benchmarks *gcc, perl,* and *crafty* observe lower performance on the better of the two cores of HET-A compared to the overall best core provided by HOM, their contested execution with them more than compensates for the deficit.

Figure 5-8 shows the IPT of each benchmark under the same three scenarios, except that the HET-B CMP design is used instead of the HET-A CMP design. From Table 5-1, HET-B is comprised of the *gcc* and *mcf* core types. Due to the long clock period of the *mcf* core, it tends to become a saturated lagger in the contested execution of half of the benchmarks, resulting in little benefit for these. Nevertheless, contesting on HET-B yields an average speedup of 13% and a maximum speedup of 39% (for *twolf*) compared to not contesting.

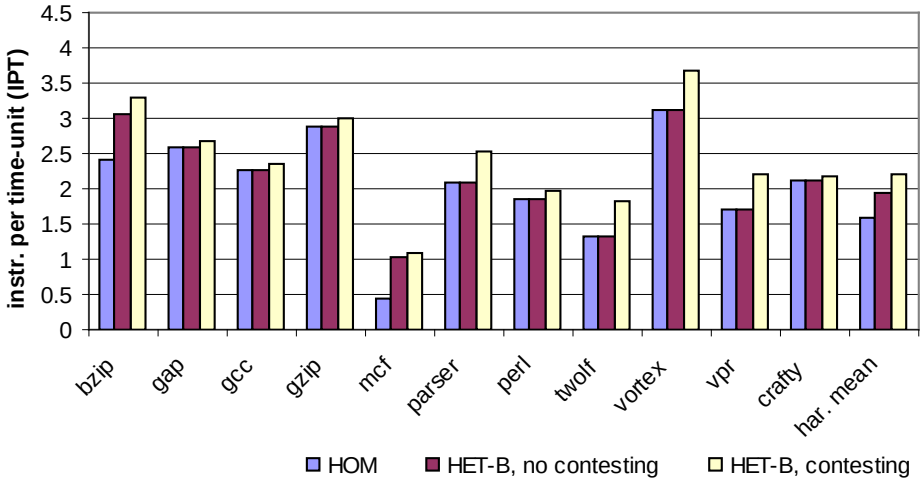


Figure 5-8: Performance of each benchmark on HOM, HET-B without contesting, and HET-B with contesting.

The highest overall speedup is attained for contesting between the two core types of the HET-C CMP design (featuring the *bzip* and *crafty* core types). Figure 5-9 shows the results

for HET-C. Contesting on HET-C yields an average speedup of 22% and a maximum speedup of 50% (for *vpr*) compared to not contesting. For the benchmarks *gzip*, *vortex*, and *vpr*, contesting prevents performance from dipping below that of the overall best core provided by HOM, and even boosts performance significantly above it. These speedups are attained through active participation of both cores in the effective computation.

With contesting, HET-C achieves an average speedup of 34% over HOM. In contrast, without contesting, HET-C achieves an average speedup of 11% over HOM. Therefore, contesting has roughly tripled the single-thread performance advantage of heterogeneity in this system. HET-C was primarily designed with heavy-loading of the system in mind. Thus, contesting can be viewed as a technique that provides robustness to heterogeneity – allowing a system to be primarily designed for heavy loading, yet not compromise single-thread performance when the system is lightly loaded (this issue is further addressed in the next section).

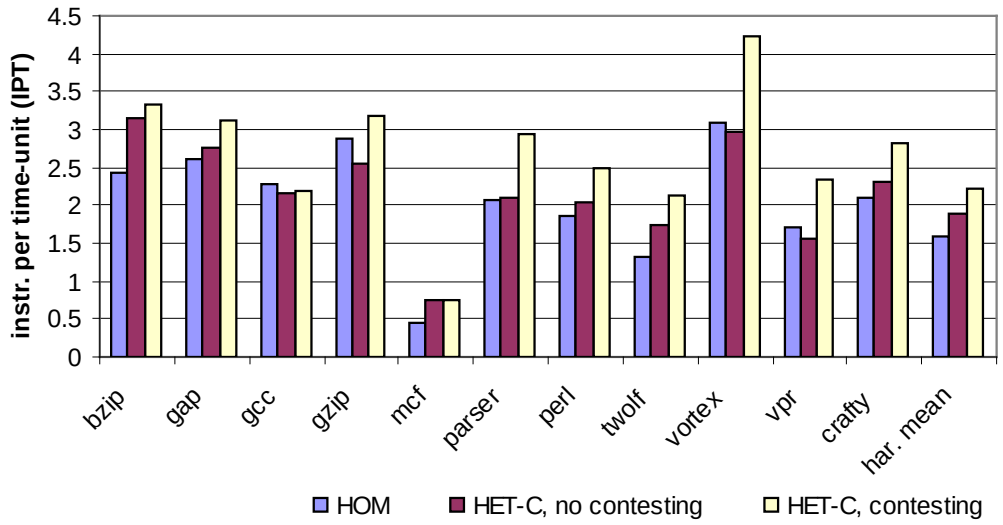


Figure 5-9: Performance of each benchmark on HOM, HET-C without contesting, and HET-C with contesting.

5.6. Discussion

5.6.1. Combination of Core Types for Contesting

The prior sections considered clear-cut figures of merit for the design of a heterogeneous CMP with a constrained number of core types. However, the truly best design goal may be more involved. For instance, an issue that may be of concern is certain benchmarks performing worse than they would have in a homogeneous system, despite heterogeneity improving single-thread performance on the whole. This can be reflected in the figure of merit by penalizing it when that is the case. Another issue may be the robustness of the design to perform well both when the system is loaded (throughput) and unloaded (single-thread performance). Combining simpler figures of merit can reflect such hybrid design goals.

The design of a constrained heterogeneous CMP involves compromises, by virtue of limiting the number of core types and optimizing for one figure of merit or the other. The above results showed that contesting provides a degree of performance robustness that compensates for certain side-effects of these compromises. Here we discuss two examples:

- Section 5.5 showed that a constrained heterogeneous CMP design improves single-thread performance for the benchmark suite as a whole compared to a homogeneous CMP design, but that specific benchmarks may nonetheless perform worse. The results showed that contested-execution of these benchmarks makes up for this performance deficit.

- The *cw-har* figure of merit, like the *avg* and *har* figures of merit, tries to steer the design of a constrained heterogeneous CMP towards higher single-thread performance, but it balances this goal with the need to distribute job types (benchmarks) evenly among the core types in

anticipation of a heavily loaded system. Thus, while this figure of merit does exploit application-level heterogeneity for higher single-thread performance, it may not do so to the same extent as the other narrower figures of merit. The results in 5.5 showed that contesting boosts single-thread performance of the heterogeneous CMP that was designed taking into account heavy loading (HET-C), compensating for any deficit with respect to the other designs (HET-A, HET-B). Thus, for systems that observe periods of heavy loading, HET-C with contesting as an available (but optional) mode of execution is a better design point than the other considered CMPs, because it is more robust, handling both periods of heavy and light loading well.

5.6.2. Customizing Cores for Contesting

Cores that are customized for application-level performance are not necessarily suitable for fine-grain regions of code. Architectural contesting will provide its greatest advantage when the cores are customized not for applications, but for fine-grain regions of code. In other words, the true single-thread performance potential of contesting can only be achieved when the cores are customized with contesting in mind. On the other hand, cores that are customized for fine-grain regions of code-will not necessarily be the best for application-level performance. Thus, a heterogeneous CMP consisting of such cores may hamper the benefit of “lower hanging fruit”: throughput improvement.

Determining the best core designs for contesting is much more complex than determining the best core design for an application, as the different core designs need to be explored together in contesting pairs (or contesting trios, etc.) – resulting in an explosion in an already vast overall design space. In addition, conducting design exploration across this design-space is a

slower process, as measuring the performance of design points involves simulation of contested execution (which is more time-consuming than simulation of conventional execution).

5.6.3. Contesting vs. More Core Types

We introduce a fourth constrained heterogeneous CMP design, HET-D, which is comprised of three core types instead of just two. The *har* figure of merit was used to select the three core types (maximizes harmonic-mean IPT of the benchmarks). HET-D is comprised of the customized cores of *twolf*, *crafty*, and *mcf*.

For each benchmark, Figure 5-10 compares the performance of contesting between the two core types of HET-C ("HET-C, contesting") to the performance of executing the benchmark on the most suitable core type of HET-D ("HET-D, no contesting"). In addition, the performance of executing the benchmark on its own customized core ("HET-ALL, no contesting") is shown for comparison.

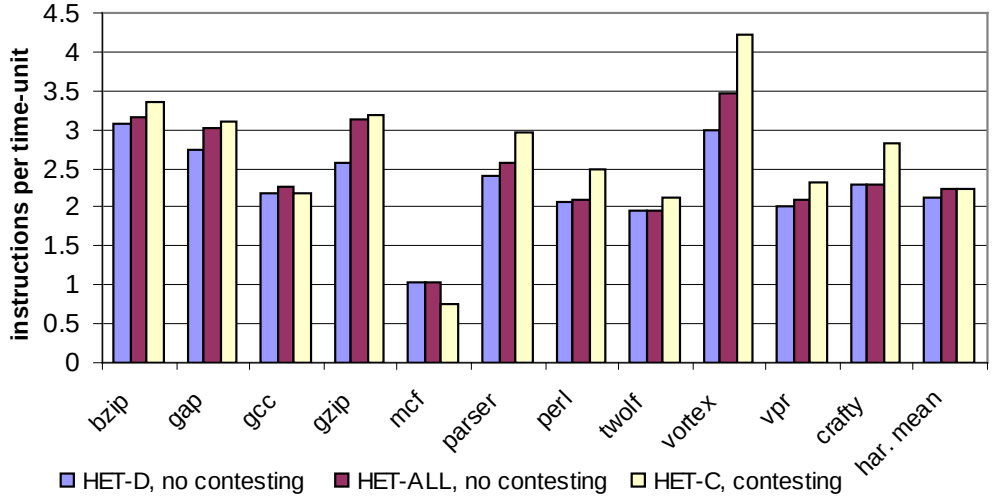


Figure 5-10: Comparison of dual-core contesting vs. employing more cores.

These results show that, on average, contesting between only two core types can yield as much single-thread performance enhancement as a heterogeneous system with all eleven core types (and typically more for a majority of benchmarks), and slightly more than a system with three core types. Therefore, in terms of maximizing single-thread performance, contesting may be a more cost-effective approach than increasing the number of core types in the system.

5.7. Chapter Summary

This chapter showed that workload behavior tends to vary considerably at fine granularities. Architectural contesting leverages the differently-designed cores in a heterogeneous multi-core to automatically and fluidly transfer effective execution to the most suitable core, exploiting workload variations that are too fine-grain to be handled by previous adaptational and migrational approaches. In addition to evaluating two-way contesting in a relatively unconstrained heterogeneous multi-core (as many core types as benchmarks), the chapter explored the interplay between contesting and the number of core types in more constrained heterogeneous multi-core designs. This exposed the broader issue of constrained heterogeneous multi-core design and how it influences, and may be influenced by, contesting.

CHAPTER 6: Conclusions and Future Work

This thesis has addressed the issue of providing microarchitecturally diverse core designs in a chip multiprocessor, how to do so efficiently, what needs to be considered when splitting up the workload space between the differently designed cores, and how fine-grain changes in application behavior can be exploited. There remains, however, considerable work to be done surrounding this issue.

6.1. Design Diversity over Circuit-Level Design Rigor

An important question that was not feasible to accurately address with available resources for this thesis is whether selectability between cores that are not as rigorously optimized at the circuit-level can outperform a regular homogeneous system with a rigorously designed core. If so, it would be indicative of the potential for a radical shift in microprocessor design methodology.

The current conventional approach to designing processors in effect favors rigorous circuit-level design, at the expense of sloppy high-level design (employing a single core design for all types of application behavior). The ability to efficiently incorporate numerous core designs in a microprocessor, however, may change this dynamic. In fact, under a fixed budget for design effort, an extreme approach would be to achieve feasibility in the incorporation of numerous differently microarchitected cores by employing core designs that are automatically synthesized, rather than custom designed. Such a design methodology, which can be labeled as exploiting “Sloppy Circuit-level Design”, would shift product

development effort from circuit-level custom design to the high-level design of different cores suited for different types of application behavior.

It should be noted that although the pretext of a fixed budget for design effort presents an intriguing angle for the analysis of core-selectability, it should not be viewed as the be-all and end-all context. A more relevant question might be whether core-selectability provides better return for the same *increase* in design effort. Ultimately, there are strong indications of there being little overall single-thread performance to be gained through tweaking any one general-purpose design, no matter how much design effort is put into it. And there are strong indications of there being notable single-thread performance to be gained through the incorporation of microarchitectural designs that are geared towards the behavior of the applications they execute.

6.2. Greater Microarchitectural Diversity

Another lacking aspect of the analysis presented in this thesis is the scope of microarchitectural diversity between the different cores. In the results presented in this thesis the diversity was limited to the sizing of microarchitectural features, while staying within the boundaries of a canonical superscalar processor design. However, there are indications that there is greater performance to be gained from more ingrained microarchitectural diversity. Examples of such differences would be in the incorporation of Checkpoint Processing and Recovery (CPR) [33], Value Prediction [71], Instruction Collapsing [114], Continual-Flow Pipelining (CFP) [103] and Trace Caching [29] – enabling these techniques to be employed when beneficial and their overhead to be completely removed from the active critical units

when not beneficial. Although such techniques may render individual core designs more complex, the point is that the complexities will be separated from each other.

It is important to realize that the benefit of adding such dimensions to the microarchitectural design space will not be limited to the potentially meager benefit that may have been documented for them in general purpose settings. This is a result of the fact that the incorporation of such techniques can shift how different design units scale relative to reach other – and thus the entire landscape of the design space. For instance, the benefit of value prediction [71] is not limited to reducing the average access time to data by means of speculation, at the expense of the overhead of the circuitry necessary to deal with misspeculations. It can also change the best issue-queue and ROB size of the design. Thus, for the true potential of such a technique to be exposed, the entire microarchitectural design needs to be geared towards application behavior for which it is beneficial – something that is infeasible when constrained to a single design solution.

In other words, the benefit of the incorporation of more ingrained microarchitectural diversity will be in the context of the application behavior for which each design solution is beneficial, when the entire design of the core is geared towards that type of application behavior.

6.3. Exploiting Detachability/Unpluggability

All in all, core-selectability is an approach to provide the ability to dynamically employ more suitable core designs depending on the behavior of the application(s) at hand. It achieves this with low overhead by moving the extra circuitry necessary for implementing changeability out of the critical structures of the cores. Nevertheless, it does entail a small overhead at the

point of port-sharing between the different cores in a node. Moreover, it does not provide a means to dynamically customize the non-core portion of the design (e.g. the interconnection network). Meanwhile, considerable performance gain has been shown to be achievable through customization of on-chip interconnection networks to the behavior of applications [76].

If it were possible to move the overhead of implementing changeability even further out of the design, for instance to the level of the entire Chip Multiprocessor, not only would core-selectability be achievable, so would selectability in the interconnection network. The only caveat in such an approach (which can be labeled “CMP-selectability”) is that, unlike the cores in Chip Multiprocessors, the interconnection networks are not consuming progressively smaller portions of the die area (i.e. the “shrinking factor” of Figure 1-5). However, if at this level of the design the frequency of the need for changeability could be limited (through the incorporation of mechanisms within the different CMPs that deal with finer-grained changeability, e.g. core-selectability), it may be extremely beneficial to implement CMP-selectability by moving the different CMP designs onto separate dies and employing unconventional technology to enable low-overhead change in which CMP is actively employed. As a long standing packaging feature of processors manufactured for desktop and server environments has been the potential to physically detach (or unplug) the chip from the computer in which they are employed, such an unconventional approach as “mechanical” replacement of the CMPs may be worth investigation [36].

Bibliography

[1] A. Dhodapkar and J. Smith. “Managing Multi-Configuration Hardware via Dynamic Working Set Analysis”, In Proceedings of the *Inter. Symp. on Comp. Arch. (ISCA)*, 2002.

[2] A. Hilton, A. Roth, “Ginger: Control Independence Using Tag Rewriting”, Proceeding of the *Intl. Symp. On Computer Architecture (ISCA)*, 2007.

[3] A. Joshi, A. Phansalkar, L. Eeckhout, L. John, "Measuring Benchmark Similarity Using Inherent Program Characteristics," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 769-782, Jun., 2006.

[4] A. Lungu, D. J. Sorin, “Verification-Aware Microprocessor Design”, In Proceedings of the *Int’l Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2007.

[5] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John, “Four Generations of SPEC CPU Benchmarks: What has changed and what has not”, *Tech Report TR-041026-01-1*. Oct. 2004.

[6] A. Phansalkar, A. Joshi, L. Eeckhout, L. K. John, “Measuring Program Similarity: Experiments with SPEC CPU Benchmark Suites”, In Proceedings of the *Int’l Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2005.

[7] A. S. Al-Zawawi, V. K. Reddy, E. Rotenberg, and H. Akkary. Transparent Control Independence (TCI). In Proceedings of the *International Symposium on Computer Architecture (ISCA)*, 2007.

[8] A. S. Dhodapkar and J. E. Smith, “Managing Multiconfiguration Hardware via Dynamic Working Set Analysis,” In Proceedings of the *Int’l Symposium on Computer Architecture (ISCA)*, 2002.

[9] B. C. Lee and D. Brooks, “Applied Inference: Case Studies in Microarchitectural Design”, In *ACM Transactions on Architectecture and Code Optimization*, 2010.

[10] B. C. Lee and D. Brooks, “Illustrative design space studies with microarchitectural regression models”, In Proceeinds of the *International Symp. on High-Performance Computer Architecture (HPCA)*, 2007.

[11] B. Greskamp, J. Torrellas, “Paceline: Improving Single-Thread Performance in Nanoscale CMPs through Core Overclocking”, In Proceedings of the *Inter. Conf. on Parallel Arch. and Comp. Tech. (PACT)*, 2007.

[12] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, Y. Solihin, “Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling”, In Proceedings of the *Int’l Symposium on Computer Architecture (ISCA)*, 2009.

[13] C. Cher, T. Vijaykumar, “Skipper: A Microarchitecture for Exploiting Control-Flow Independence”, Proceeding of the *Int’l Symposium on Microarchitecture (MICRO)*, 2001.

[14] C. Kim, S. Sethumadhavan, M.S. Govindan, N. Ranganathan, D. Gulati, D. Burger and S.W. Keckler, “Composable Lightweight Processors”, In *Proceedings of the Int’l Symposium on Microarchitecture (MICRO)*, 2007.

[15] C. Zilles, G. Sohi, “Execution-based Prediction Using Speculative Slices”, Proceedings of the *International Symposium on Computer Architecture (ISCA)*, 2001.

[16] D. Albonesi. “Dynamic IPC/clock rate optimization,” In Proceedings of the *Int’l Symposium on Computer Architecture (ISCA)*, 1998.

[17] D. Burger, S. Kaxiras, and J. R. Goodman, “DataScalar Architectures”, In Proceedings of the *Inter. Symp. on Comp. Arch. (ISCA)*, 1997.

[18] D. Folegnani and A. Gonzlez. “Energy-Effective Issue Logic”, In Proceedings of the *Inter. Symp. on Comp. Arch.* (ISCA), 2001.

[19] D. H. Albonesi et al. “Dynamically Tuning Processor Resources with Adaptive Processing”, In *IEEE Computer, December* 2003.

[20] D. Marculescu, A. Iyer, “Application-driven processor design exploration for power-performance trade-off analysis”, In Proceeding of the *International Conference on Computer Design (ICCD),* 2001.

[21] D. Ponomarev, G. Kucuk, O. Ergin, K. Ghose, P. Kogge, "Energy-Efficient Issue Queue Design", *IEEE Transactions on Very Large Scale Integration Systems,* 2003.

[22] D. Tarjan, S. Thoziyoor; N. P. Jouppi, “CACTI 4.0”, *Technical report HPL-2006-86,* 2006.

[23] D. Tullsen, S. Eggers, and H. Levy, “Simultaneous multithreading: Maximizing on-chip parallelism”, In *Proceedings of the Int’l Symposium on Computer Architecture (ISCA),* 1995.

[24] E. Grochowski, R. Ronen, J. Shen, and H. Wang, “Best of both latency and throughput”, In Proceeding of the *International Conference on Computer Design (ICCD),* 2004.

[25] E. Hao, P. Chang, and Y. Patt, “The Effect of Speculatively Updating Branch History on Branch Prediction Accuracy, Revisited”, In Proceedings of the *Inter. Symp. on Microarchitecture (MICRO),* 1994.

[26] E. Ipek, M. Kirman, N. Kirman, and J. F. Martinez, “Core Fusion: Accommodating software diversity in chip multiprocessors”, In Proceedings of the *Int’l Symposium on Computer Architecture (ISCA)*, 2007.

[27] E. Larson, S. Chatterjee, T. Austin, “The MASE Microarchitecture Simulation Environment”, In Proceedings of the *Int’l Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2001.

[28] E. Rotenberg, "AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessor", In Proceedings of *Fault-Tolerant Computing Systems (FTCS)*, 1999.

[29] E. Rotenberg, S. Bennett, J. E. Smith, “Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching”. In Proceedings of the *International Symposium on Microarchitecture (MICRO)*, 1996.

[30] E. Sohmaier, H. Shan, “Architecture Independent Performance Characterization and Benchmarking for Scientific Applications,” In *the Intl. Workshop on Modeling, Analysis, and Simulation of Computer and Telecom. Systems (MASCOTS’04)*, 2004.

[31] G. A. Abandah and E.S. Davidson, “Configuration Independent Analysis for Characterizing Shared-Memory Applications,” In Proceedings of the *Proc. Int’l Parallel Processing Symp.*, 1998.

[32] H. Akkary and M.A. Driscoll, "A Dynamic Multithreading Processor", In Proceedings of the *Int. Symp. on Microarchitecture*, 1998.

[33] H. Akkary, R. Rajwar, S. T. Srinivasan, “Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors”, In Proceedings of the *Inter. Symp. on Microarchitecture (MICRO)*, 2003.

[34] H. H. Najaf-abadi and E. Rotenberg, “Architectural Contesting”, In Proceedings of the *Int’l Symposium on High-Performance Computer Architecture (HPCA)*, 2009.

[35] H. H. Najaf-abadi, E Rotenberg, “Configurational Workload Characterization”, In Proceedings of the *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2008.

[36] H. H. Najaf-abadi, E. Rotenberg, “Exploiting Detachability: A Non-Silicon Approach to Polymorphism“, In the *Non-Silicon Computing Workshop (NSC-4), in conjunction with ISCA*, 2007.

[37] H. H. Najaf-abadi, E. Rotenberg, “The Importance of Accurate Task Arrival Characterization in the Design of Processing Cores”, In Proceedings of the *IEEE Int’l Symposium on Workload Characterization (IISWC)*, 2009.

[38] H. H. Najaf-abadi, N. K. Choudhary, E. Rotenberg, "Core-Selectability in Chip Multiprocessors", In Proceedings of the *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2009.

[39] H. Vandierendonck, K. De Bosschere, “Experiments with Subsetting Benchmark Suites”, In the *Workshop on Workload Characterization*, 2004.

[40] H. Vandierendonck, K. De Bosschere, “Many Benchmarks Stress the Same Bottlenecks”, In Proceedins of the *Workshop on Computer Arch. Eval. using Commercial Workloads (CAECW)*, 2004.

[41] http://www.crhc.illinois.edu/ACS/tools/ivm/about.html

[42] http://www.opencores.org

[43] http://www.opensparc.net

[44] http://www.virtutech.com

[45] J. Cong, A. Jagannathan, G. Reinman, and M. Romesis, "Microarchitecture Evaluation with Physical Planning", In Proc. of the *Design Automation Conference (DAC)*, 2003.

[46] J. Dujmovic and I. Dujmovic, “Evolution and Evaluation of SPEC benchmarks”, In *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 2-9, 1998.

[47] J. E. Smith. “Instruction-level distributed processing”, In *IEEE Computer*, 34(4):59.65, April 2001.

[48] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W.R. Davis, P.D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, R. Jenkal, "FreePDK: An Open-Source Variation-Aware Design Kit", In Proceedings of the *Int’l Conference on Microelectronic Systems Education (MSE'07)*, 2007.

[49] J. Emer et al., “Single-threaded vs. Multithreaded: Where should we focus?”, *IEEE Micro, vol. 27, no. 6*, Nov./Dec. 2007.

[50] J. Hennessy, D. Citron, D. Patterson, G. Sohi, “The Use and Abuse of SPEC: An ISCA Panel,” *IEEE Micro*, vol. 23, no. 4, pp. 73-77, Jul/Aug, 2003.

[51] J. Huh, D. Burger, and S. W. Keckler, “Exploring the Design Space of Future CMPs,” In Proceedings of the *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2001.

[52] J. Koppanalil, P. Ramrakhyani, S. Desai, A. Vaidyanathan, and E. Rotenberg. “A Case for Dynamic Pipeline Scaling”, In Proceedings of the *Int’l Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'02)*, 2002.

[53] J. Yi, D. Lilja, and D. Hawkins, "A Statistically Rigorous Approach for Improving Simulation Methodology," In *Proceedings of the Int'l Symp. High-Performance Computer Architecture (HPCA)*, 2003.

[54] J. Yi, R. Sendag, L. Eeckhout, A. Joshi, D. Lilja, and L. K. John, "Evaluating Benchmark Subsetting Approaches" In Proceedings of the *IEEE International Symposium on Workload Characterization (IISWC)*, 2006.

[55] K. Compton and S. Hauck. "Reconfigurable computing: A survey of systems and software", In *ACM Computing Surveys, 34(2)*, 2002.

[56] K. Hoste, A. Phansalkar, A. Georges, L. John, "Performance prediction based on inherent program similarity", In Proceedings of the *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2006.

[57] K. Skadron and P. S. Ahuja, "Hydrascalar: A multipath-capable simulator," In *Newsletter of the IEEE Tech. Committee on Computer Architecture*, Jan 2001.

[58] K. Sundaramoorthy, Z. Purser, and E. Rotenberg. "Slipstream Processors: Improving both Performance and Fault Tolerance", In Proceedings of the *Int'l Conference on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, 2000.

[59] L. Chen, D.H. Albonesi, and S. Dropsho, "Dynamically Matching ILP Characteristics Via a Heterogeneous Clustered Microarchitecture", In Proceedins of the *IBM Watson Conf. on the Intera. Between Arch., Circuits, and Compilers*, 2004.

[60] L. Eeckhout, H. Vandierendonck and K. De Bosschere "Quantifying the Impact of Input Data Sets on Program Behavior and its Applications." In *Jour. of Instruction-Level Parallelism*, Vol. 5, April 2003.

[61] L. Hammond, B. A. Nayfeh, and K. Olukotun, “A single-chip multiprocessor”, In *Computer, volume* 30, *no. 9,* Sept. 1997.

[62] M. A. Suleman, O. Mutlu, M. K. Qureshi, Y. N. Patt, “Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures”, In Proceedings of the *Int’l Conference on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, 2009.

[63] M. Andersson, J. Gudmundsson, C. Levcopoulos, G. Narasimhan, “Balanced Partition of Minimum Spanning Trees”, In Proceedings of the *International Conference on Computational Science*, 2002

[64] M. Annavaram, E. Grochowski, and J. Shen, “Mitigating Amdahl's Law Through EPI Throttling”, In Proceedings of the *International Symposium on Computer Architecture*, 2005.

[65] M. Chen, K. Olukotun, “TEST: A Tracer for Extracting Speculative Threads”, In Proceedings of the *Int’l Symposium on Code Generation and Optimization (CGO)*, 2003.

[66] M. Cintra, J. F. Mart´ınez, and J. Torrellas, “Architectural support for scalable speculative parallelization in shared-memory multiprocessors”, In Proceedings of the *Inter. Symp. on Comp. Arch. (ISCA)*, 2000

[67] M. Ekpanyapong, S. Kyu Lim, C. Ballapuram, and H. S. Lee, "Wire-driven Microarchitectural Design Space Exploration," In Proceedings of the *IEEE International Symp. on Circuits and Systems*, 2005.

[68] M. Frank, C. A. Moritz, B. Greenwald, S. Amarasinghe, A. Agarwal, “SUDS: Primitive Mechanisms for Memory Dependence Speculation”, In *MIT/LCS Technical Memo MIT-LCS-TM-591*, 1999.

[69] M. Frank, W. Lee, S. Amarasinghe, “A software framework for supporting general purpose applications on Raw computation fabric”, In *MIT-LCS Technical Memo MIT-LCS-TM-619*, 2001.

[70] M. Franklin, “Multiscalar Processors”, Kluwer Academic Publishers, 2002.

[71] M. H. Lipasti, C. B. Wilkerson, J. P. Shen, “Value locality and data speculation,” In Proceedings of the *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1996.

[72] M. Huang, J. Renau, and J. Torrellas. “Profile-based energy reduction for high-performance processors”, In the *4th Workshop on Feedback-Directed and Dynamic Optimization (FDDO4)*, 2001.

[73] M. Huang, J. Renau, J. Torrellas, “Positional Adaptation of Processors: Application to Energy Reduction”, In Proceedings of the *Inter. Symp. on Comp. Arch. (ISCA)*, 2003.

[74] M. Laurenzano, B. Simon, A. Snavely and M. Gunn, “Low Cost Trace-Driven Memory Simulation Using SimPoint”, In the *Workshop on Binary Instrumentation and Applications* (in conjunction with PACT) , 2005.

[75] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, D. A. Wood, “Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset”, In *SIGARCH Computer Architecture News*, v.33 n.4, 2005.

[76] M. M. Kim, J. D. Davis, M. Oskin, T. Austin, “Polymorphic On-Chip Networks”, In Proceedings of the *Int’l Symp. on Computer Architecture (ISCA)*, 2008.

[77] N. Choudhary et al., "FabScalar", In the *Workshop on Architecture Research Prototyping (WARP)*, 2009.

[78] P. J. Joseph, K. Vaswani, M. J. Thazhuthaveetil, "A Predictive Performance Model for Superscalar Processors," In Proceedings of the *Int'l Symposium on Microarchitecture (MICRO)*, 2006.

[79] P. Salverda and C. Zilles. "Fundamental performance constraints in horizontal fusion of in-order cores", In Proceedings of the *Int'l Symposium on High Performance Computer Architecture (HPCA)*, 2008.

[80] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, S. Dwarkadas, "Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures", In Proceedings of the *Int'l Symposium on Microarchitecture (MICRO)*, 2000.

[81] R. Berridge et. al. "IBM POWER6 microprocessor physical design and design methodology", In the *IBM Journal of Research and Development, Volume 51, Issue 6*, Nov. 2007.

[82] R. Calkin, R. Hempel, H. Hoppe, P. Wypior, "Portable programming with the PARMACS Message-Passing Library", In Proceedings of the *Parallel Comput., Special Issue on Message-Passing Interfaces*, 1994.

[83] R. Dolbeau and A. Seznec, "CASH: Revisiting hardware sharing in single-chip parallel processor", In the *Journal of Instruction-Level Parallelism (JILP), vol. 6,* April 2004. (www.jilp.org/vol6).

[84] R. E. Kessler, "The Alpha 21264 Microprocessor", In *IEEE Micro, v.19 n.2*, March 1999.

[85] R. H. J. M. Otten, L. P. P. P Van Ginneken, “Stop criteria in simulated annealing”, In Proceeding of the *International Conference on Computer Design (ICCD)*, 1988.

[86] R. Hum, “How to Boost Verification Productivity”, *EETimes*, January, 2005.

[87] R. Kumar, D. M. Tullsen, N. P. Jouppi “Core architecture optimization for heterogeneous chip multiprocessors”, In proceedings of the *Int’l Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2006.

[88] R. Kumar, D. M. Tullsen, P. Ranganathan, N. Jouppi, K. I. Farkas, "Single-ISA Heterogeneous Multicore Architectures for Multithreaded Workload Performance", In Proceedings of the *Int’l Symp. on Computer Architecture (ISCA)*, 2004.

[89] 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 of the *International Symp. on Microarchitecture (MICRO)*, 2003

[90] R. Kumar, N. P. Jouppi, D. M. Tullsen, “Conjoined-core Chip Multiprocessing” In Proceedings of the *Int’l Symposium on Microarchitecture (MICRO)*, 2004.

[91] S. Bird, A. Phansalkar, L. John, A. Mericasa, R. Indukuru, “Performance Characterization of SPEC CPU Benchmarks on Intel's Core Microarchitecture based processor”, In the *SPEC Benchmark Workshop*, 2007

[92] S. Chaudhry et al., “Rock: A high-performance SPARC CMT processor”, In the *IEEE Micro, vol. 29, no. 2*, Mar./Apr. 2009.

[93] S. G. Dropsho, G. Semeraro, D. H. Albonesi, G. Magklis, M. L. Scott, “Dynamically Trading Frequency for Complexity in a GALS Microprocessor”, In Proceedings of the *Int’l Symposium on Microarchitecture (MICRO)*, 2004.

[94] S. Ghiasi and D. Grunwald, “Aide de camp: Asymmetric dual core design for power and energy reduction”, In *University of Colorado Technical Report CU-CS-964-03*, 2003.

[95] S. Ghiasi, T. Keller, and F. Rawson, “Scheduling for heterogeneous processors in server systems”, In proceeding of *Computing Frontiers*, 2005.

[96] S. Gupta, S. Feng, A. Ansari, S. Mahlke, “Erasing Core Boundaries for Robust and Configurable Performance”, In Proceedings of the *Inter. Symp. on Microarchitecture (MICRO)*, 2010.

[97] S. Hauck, A. DeHon, “Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing,” *Morgan Kaufman*, 2008.

[98] S. J. E. Wilton, N. P. Jouppi, “CACTI: An enhanced cache access and cycle time model”, In the *IEEE Journal of Solid-State Circuits*, 31(5):677–688, 1996.

[99] S. K. Balakrishnan, R. Rajwar , M. Upton , K. Lai, “The Impact of Performance Asymmetry in Emerging Multicore Architectures”, In Proceedings of the *Int’l Symposium on Computer Architecture (ISCA)*, 2005.

[100] S. K. Reinhardt, S. S. Mukherjee, “Transient Fault Detection via Simultaneous Multithreading,” In Proceedings of the *Int’l Symp. On Computer Architecture (ISCA)*, 2000.

[101] S. Palacharla, N. P. Jouppi, and J. E. Smith, “Complexity-effective superscalar processors”, In Proceedings of the *Inter. Symp. on Comp. Arch. (ISCA)*, 1997.

[102] S. S. Navada, N. K. Choudhary, and E. Rotenberg, “Criticality-driven Superscalar Design Space Exploration”, In Proceedings of the *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2010.

[103] S. Srinivasan, R. Rajwar, H. Akkary, A. Gandhi, M. Upton, “Continual Flow Pipelines”, In Proceedings of the *Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2004.

[104] S.-H. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. Vijaykumar, “An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance caches”, In Proceedings of the *Inter. Symp. on High-Perf. Comp. Arch (HPCA)*, 2001.

[105] T Mitra, T Chiueh, “Dynamic 3D Graphics Workload Characterization and the Architectural Implications”, In Proceedings of the *Inter. Symp. on Microarchitecture (MICRO)*, 1999.

[106] T. Austin, E. Larson, D. Ernst, “SimpleScalar: An Infrastructure for Computer System Modeling,” In *Computer*, vol.35, no.2, Feb. 2002.

[107] T. Pering and R. Broderson, “Energy efficient voltage Scaling for Real-Time Operating Systems”, In Proceedings of the *IEEE Real-Time Technology and Applications Symposium (RTAS)*, 1998.

[108] T. Pering. et. al “The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms”, In Proceedings of the *Int’l Symposium on Low Power Electronics and Design, 1998*.

[109] T. Sherwood, E. Perelman, G. Hamerly, B. Calder, “Automatically Characterizing Large Scale Program Behavior,” In proceedings of the *Int’l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.

[110] T. Sherwood, S. Sair, and B. Calder, “Phase tracking and prediction”, In Proceedings of the *Inter. Symp. on Comp. Arch. (ISCA)*, 2003.

[111] U Gajanan, M. Hassan, K. C. Yen, A. Kumar, A. Ramachandran, D. Greenhill, “Implementation of an 8-core, 64-Thread, Power-Efficient SPARC Server on a Chip”, In the *IEEE Journal of Solid-State Circuits, Vol. 43, No. 1*, 2008.

[112] Y. Luo, V. Packirisamy, W. Hsu, A. Zhai, “Energy efficient speculative threads: dynamic thread allocation in Same-ISA heterogeneous multicore systems”, In Proceedings of the *Parallel Architectures and Compilation Techniques (PACT)*, 2010.

[113] Y. Ma, Z. Li, J. Cong, X. Hong, “Micro-architecture Pipelining Optimization with Throughput-Aware Floorplanning”, In Proceedings of the *ACM/IEEE Asia South Pacific Design Automation Conference*, 2007.

[114] Y. Sazeides, S. Vassiliadis, J. E. Smith, “The Performance Potential of Data Dependence Speculation & Collapsing”, In Proceedings of the *International Symp. on Microarchitecture (MICRO)*, 1996.