The Dissertation Committee for Vikas Agarwal
certifies that this is the approved version of the following dissertation:

Scalable Primary Cache Memory Architectures

Committee:

Lizy K. John, Supervisor

Stephen W. Keckler, Supervisor

Craig M. Chase

Nur A. Touba

Douglas C. Burger

Scalable Primary Cache Memory Architectures

by

Vikas Agarwal, M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2004

Dedicated to my parents.

Acknowledgments

I wish to thank the numerous people who helped and supported me. I would like to thank my advisor, Dr. Stephen W. Keckler, for the motivation and guidance that he has provided over the years.

Funding for this research was provided, in part, by The Department of Computer Sciences at The Univeristy of Texas at Austin, and in part by IBM Co-operative Fellowships for the years 2000-2003.

Scalable Primary Cache Memory Architectures

Publication No. _____

Vikas Agarwal, Ph.D.
The University of Texas at Austin, 2004

Supervisors: Lizy K. John
Stephen W. Keckler

For the past decade, microprocessors have been improving in overall performance at a rate of approximately 50–60% per year by exploiting a rapid increase in clock rate and improving instruction throughput. A part of this trend has included the growth of on-chip caches which in modern processor can be as large as 2MB. However, as smaller technologies become prevalent, achieving low average memory access time (AMAT) by simply scaling existing designs becomes more difficult because of process limitations.

The goal of this research is to achieve low AMAT and high instruction throughput at small feature sizes, using a combination of circuit and architectural techniques. Technology based models are developed to estimate cache access times that are used for architectural experiments. As a part of this research using a multi-banked design for a clustered processor in combination with selected data mapping techniques to achieve acceptable hit rate while

maintaining low cache access time is investigated. Additionally, predictive instruction steering strategies that help improve the performance of the clustered processor design are developed and evaluated.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Microprocessor performance improvements have been driven by progress in fabrication technology that has caused transistor as well as wires to get smaller and faster. The smaller feature sizes have provided the benefits of more complex microarchitecture as well as faster gate switching times. This research investigates the impact of technology scaling on processor performance. The increasing communication delay [8] between microarchitectural structures limits the improvement in performance that can be achieved as feature sizes decrease. This research proposes and evaluates using clustered caches in conjunction with clustered execution units as a means to improve the performance of processors in the face of increasing communication delays. The results show that 39% improvement in performance can be achieved by using a combination of dynamic data mapping and aggressive predictive instruction steering.

1.1 Scaling of Conventional Architectures

For the past decade, microprocessors have been improving in overall performance at a rate of approximately 50–60% per year. These substantial

performance improvements have been mined from two sources. First, designers have been increasing clock rates rapidly, both by scaling technology and by reducing the number of levels of logic per cycle. Second, designers have been exploiting the increasing number of transistors on a chip, plus improvements in compiler technology, to improve instruction throughput (IPC). Although designers have generally opted to emphasize one over the other, both clock rates and IPC have been improving consistently. Figure 1.1 shows that while some designers have chosen to optimize the design for fast clocks (Compaq Alpha), and others have optimized their design for high instruction throughput (HP PA-RISC), the past decade's performance increases have been a function of both.

If the cycle time of a processor is dominated by gate delay, then faster transistor contribute directly to higher performance. However, increasingly wire delay contributes a significant portion of the clock cycle time. This research develops an analytical wire model based on empirical results from a 3D field solver to study the scalability of interconnects as we move to smaller feature sizes. The results from the wire model show that only 10% of a 400mm^2 chip will be accessible at 35nm for an aggressively clocked microprocessor. The wire model is further used in a model to estimate the access time of various register file and cache like memory structures in the microprocessor. The model shows that with an aggressive clock rate it will take as much as 7 cycles to access a 64KB primary data cache at 35nm for an aggressive 13.5 GHz clock. These microarchitecture latency models are used to evaluate the impact

of technology scaling on the performance of a processor architecture.

Achieving high performance in future microprocessors will be a tremendous challenge, as both components of performance improvement are facing emerging technology-driven limitations. Designers will soon be unable to sustain clock speed improvements at the past decade's annualized rate of 50% per year. With detailed wire and component models, we show that today's designs scale poorly with technology, improving *at best* 12.5% per year over the next fourteen years [1]. The results show that designers must select among deeper pipelines, smaller structures, or slower clocks, and that none of these choices, nor the best combination, will result in scalable performance. Whether designers choose an aggressive clock and lower IPC, or a slower clock and a higher IPC, today's designs cannot sustain the performance improvements of the past decades.

One tremendously important contributor to the performance of modern microprocessors is the primary cache design. For maximal processor performance, the primary cache should be as large as possible, while at the same time reducing the average memory access time and sustaining reasonable hit rates. In the past, the trend in cache design has been to increase the capacity to improve the hit rate. However, recently there have been divergent approaches to the design of level-1 caches. One approach has been to design around the increased access latency of a level-1 data cache by increasing the pipeline depth. The other has been to reduce the capacity of the level-1 data cache in order to maintain a low access latency: for example the Pentium 4

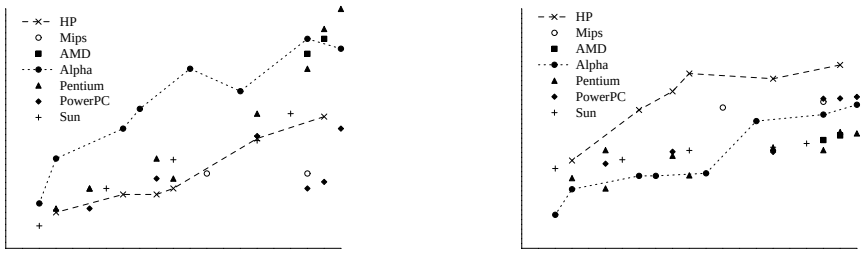


Figure 1.1: Processor clock rates and normalized processor performance (SpecInt/Clock rate), 1995-2000.

has a 8K DL1 as compared to the 16K DL1 in the Pentium III. However, as smaller technologies become prevalent, achieving low average memory access time by simply scaling existing designs becomes more difficult because of process limitations. The circuit issues that will need to be considered when designing high speed primary caches include stability, noise immunity, access latency, and power consumption.

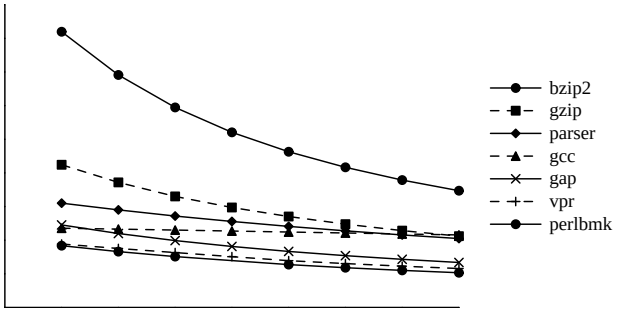
1.2 Clustered Primary Memory Systems

Wider issue processors with increasing numbers of execution units require low latency and high bandwidth data cache access. However, adding ports to a monolithic data cache increases both latency and area. In addition, increases in on-chip wire latency encourages partitioning so that the size and delay of critical structures, such as caches and register files, are limited. The simplest way to increase the available cache ports is to the partition the caches into independently accessible cache banks. This research proposes and evalu-

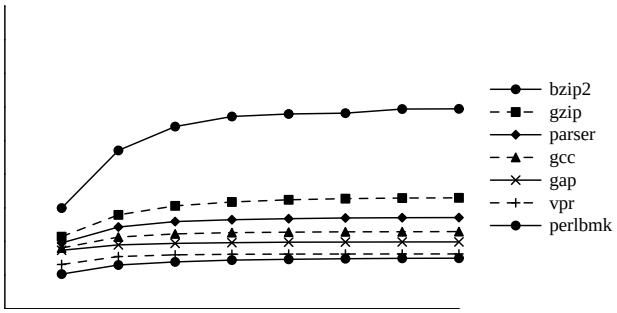
ates a group of clustered cache architectures connected to a clustered processor architecture.

While the Alpha 21264 was an early example of a commercial clustered microprocessor [32], an enormous amount of research has been done addressing clustered architectures. While most of the work on clustering has focused on the processor core, relatively little has addressed the effect on the memory system. Wider issue processors seek to increase issue-rate of all instruction, including loads and stores. Unfortunately, just adding load/store execution units merely increases bandwidth and latency pressures on traditional monolithic level-1 caches. Since clusters must be spread spatially across some fraction of the chip, the distance between a cluster and a centralized level-1 cache increases, thus increasing cache access latency. Figure 1.2a shows the effect of increasing level-1 cache latency in a 16-wide monolithic processor across a spectrum of the SPEC2000 benchmark suite. Reducing the level-1 cache latency from 3 cycles to 1 cycle increases average instructions per clock (IPC) by an average to 19%. Similar improvements result when the latency is reduced from 5 cycles to 3 cycles. Adding load/store execution units without increasing the level-1 cache bandwidth will elevate average memory access time because of level-1 cache contention. Figure 1.2b shows that increasing the number of level-1 cache ports from 1 to 3 results in an average IPC boost of 39%.

While a monolithic cache forces the design to trade off bandwidth for latency, a partitioned design allows for high bandwidth and higher capacity with low access latency. Placing a cache bank near a cluster improves average



(a)



(b)

Figure 1.2: Instructions per cycle (IPC) for SPEC2000 integer benchmarks as a function of (a) cache latency (b) number of cache ports.

cache bandwidth and the best-case memory latency when the needed data is found in the local bank. The goal of the architectures and algorithms presented in this research is to increase the likelihood that a load or a store finds its data in the local cache bank. We first discuss floor-planning and placement alternatives of processor and memory clusters that affect memory and inter-cluster communication latency, including a cluster centric topology that minimizes inter-cluster communication and a memory-centric topology that reduces access latency to remote cache banks.

1.3 Increasing Local Cache Bank Hits

In addition to the data placement policies, two methods to match load and store instructions with the cluster containing the data are evaluated: steering instructions to the clusters predicted to have the data or dynamically migrating data to the cluster with the instruction requiring the data. Two algorithms for steering memory instructions (i) dependence based steering, and (ii) predictive steering are examined. This research examines each of these steering policies with clustered caches that are either statically mapped or that allow dynamic data mapping. Our results show that static mapping provides little performance improvement over a slower monolithic cache, but that combining a dynamically mapped cache that allows data migration with the appropriate steering algorithm provides performance improvements of up to 51% over a monolithic cache. The results also indicate that some benchmarks do not respond well to cache bank prediction, which motivated the design

and evaluation of a hybrid steering scheme that dynamically selects between dependence-based steering and cache bank prediction steering.

Chapter 2 discusses topics related to this research and differentiates it from this work. Chapter 3 describes the technology models developed to estimate latencies of various memory structures in a microprocessor. Chapter 4 uses the latency models to evaluate the impact of shrinking feature sizes on the performance of a given processor design as well as the impact of increasing cache latency on the performance of the memory sub-system. Chapter 5 proposes and evaluates the design of clustered caches for a clustered microprocessor. Chapter 6 investigates the interaction between the steering policy used to map instructions to the execution clusters and the placement of data in the clustered cache. Finally, Chapter 7 summarizes the contributions of this research.

Chapter 2

Related Work

This research investigates how conventional processor architectures scale with feature size along with how to improve the performance of the primary data cache in future generations of processors using a combination of clustered caches and instruction steering policies. This chapter discusses prior work in the areas of processor scaling, design of clustered caches and instruction steering, and at the same time differentiating it from the work in this dissertation.

2.1 Scaling of Conventional Processors

The traditional approach to improving performance has been to improve both clock rate and instruction throughput. However at small feature sizes this research shows that for conventional architectures clock rate and IPC will become antagonistic limiting the performance growth achievable by conventional microarchitectures. The limits to which a conventional pipeline can be scaled at smaller feature sizes and aggressively scaled clock rates has already been covered elsewhere [24, 28, 29, 57]. Also, wider issue processors with increasing numbers of execution units require low latency register files with a large number of ports. However, adding a large number of ports to a

register increases the area and latency of the register file leading to the design of partitioned processors.

Several studies as well as a commercial microprocessor have suggested methods of partitioning conventional microarchitectures. Palacharla, et. al. studied critical paths in microprocessors and concluded that both the wake-up/selection logic for an instruction window and the data bypassing paths would scale poorly with future technologies [42]. To combat this problem, the authors propose a partitioning scheme that decomposes the execution units into clusters with separate register files. The centralized instruction window is replaced by a collection of distributed queues that hold streams of dependent instructions. The penalty for this partitioning is reported to be a 5% reduction in IPC. The Trace processor architecture is an extension to the complexity-effective superscalar architecture described above [50]. Instead of partitioning the instructions into different queues on the fly, traces are built dynamically and stored in a centralized trace cache. The trace scheduling logic fetches a trace and delivers it to one of many processing elements, where it is executed. Reinman, et. al. propose a scalable architecture for providing high instruction fetch bandwidth [45]. It includes a fetch target queue which allows the branch predictor to operate independently from the instruction cache, and a large fetch target buffer (FTB). To improve access time to the FTB, it is split into multiple levels, much like an L1 and L2 cache hierarchy. Finally, the Compaq Alpha 21264 actually implements a physical clustering scheme to reduce the register file size and the complexity of the bypass wires [31]. The integer units

are divided into two clusters, each with a copy of the physical register file. Instructions read operands from the cluster register file and write results to both the local and remote register files.

The Ultrascalar architecture suggests that existing microarchitectures are scalable to large numbers of execution units [26]. The Ultrascalar processing elements are connected using a hierarchical fat-tree network which is used to route all available register values to all of the instructions distributed throughout the processing elements at the end of each clock cycle. While the Ultrascalar II architecture reduces the global wiring requirements by only transmitting those values that are necessary, global communication is still required [36]. With substantial wire delays in future fabrication processes, this architecture must choose between slow clock rates or a substantial number of clock cycles to bypass data between remote processing elements. While the authors do not report IPC penalties, they will clearly be substantial enough to render the architecture uncompetitive with even conventional cores.

The Multicluster architecture suggested by Farkas, et. al. also partitions the execution units into clusters [19]. Each cluster has its own instruction dispatch queue, register file, and local bypass paths. Since the register namespace is split across the clusters, additional instructions are sometimes generated by the instruction distribution logic to move an operand to the other cluster. While programs can run unmodified, substantially better performance can be achieved with assistance from the compiler to balance the number of instructions that are delivered each cluster and to reduce the number of inter-

cluster communications. With two clusters, the authors report a reduction of 10-25% in IPC.

The more obvious partitioning of a large future chip is a single chip multiprocessor architecture (CMP) [23, 41]. This is a natural transition as long network latencies between processors in today's multiprocessors can be mapped directly onto the long on-chip wire delays. However, latency to access shared on-chip resources, such as the L1 or L2 cache must still be taken into account. Assigning a private L1 cache to each processor rather than sharing one across multiple processors will be more important in wire dominated technologies as the latency to access a globally shared structure will be substantial. Of course, like their larger ancestors, CMP architectures require parallel workloads to achieve speedup. The RAW architecture at MIT is an extension to a CMP that tightly couples an on-chip network to each processor [61]. The architecture scales well with technology as each processor has its own execution core, cache, and local memory. The compiler is responsible for discovering instruction and memory level parallelism, partitioning the program into instruction streams, and coordinating the communication through the tightly coupled and low-overhead network [7, 37]. Explicit communication between independent processors through an on-chip network is well matched to wire-delay constrained technologies. The Power4 [5, 17] is an example of a commercial CMP that also implements clustered functional units within the two processors that constitute the CMP. The Power5 [30] extends the Power4 design to support simultaneous multi threading [18] in addition to being a

CMP with clustered execution units.

While each of these architectures pay an IPC penalty for partitioning, each benefits from the higher clock rates enabled by smaller clusters. This research focuses on improving processor performance by using dynamic data mapping on partitioned primary data caches for a processor with clustered execution resources that require explicit communication to transfer values between clusters.

2.2 Design of Clustered Caches

Prior research has suggested that cache miss rates could be improved by partitioning the cache and directing different types of reference to each partition. Wolfe and Boleyn [63] proposed splitting the data cache and dedicating half to integer data and half to floating point data. The type of the load or store (integer versus floating-point) then determines which partition to access. Cho, et al. proposed splitting the data cache between the stack and the heap memory regions, with address ranges determining the cache partition [14]. Such techniques have the advantage that each sub-cache can be smaller and faster than a larger unified cache. However, these strategies do not necessarily use cache capacity effectively if the reference stream is strongly weighted toward one data type or the other. In addition, if the partitioning is not absolute in the sense that data could reside in either cache, additional coherence overhead must be paid. An interesting refinement is the cache architecture of the Itanium-2 in which all floating point data is delivered directly

from the level-2 cache and is not loaded into the level-1 cache [47]. This logical partitioning enable the level-1 cache to be smaller and faster without being polluted by floating point data which often exhibits poor temporal locality.

Many previous studies have suggested various forms of banked caches to provide better bandwidth to a monolithic processor architecture. Rivers, et al. studied the effect of cache banks as an alternative to adding ports and suggested bandwidth saving optimizations, such as re-ordering and combining references to reduce conflicts [48]. Parallel cachelets add a dynamic data mapping strategy to a banked to reduce bank conflicts exhibited by a statically mapped cache [38]. Loads are dynamically assigned in round-robing fashion to cachelets and subsequent instances of the load are assigned to the same cachelet. Data can be replicated on demand to provide higher bandwidth. Kim et. al. have proposed partitioning the level-2 cache as a means of reducing the average access latency of the level-2 cache [33]. The Access Region Cache partitions the data into stack and non-stack regions, and within each region statically map the data across a multi-banked cache [59]. A predictor uses the program counter to select the cache bank before the address is computed. Yoaz, et al. also describe a predictor preceding a statically mapped and banked cache to determine a cache bank before the address is known [64]. Early bank identification is used to prevent simultaneous issuing of instructions that must access the same cache bank. In addition, the predictor enables a faster partitioned cache architecture because in the common case an address generator must communicate only with a local cache bank. Partitioned caches

with static address interleaving have been examined in the context of a statically scheduled VLIW processors [21]. Here the compiler is responsible for for performing memory address disambiguation and cluster assignment.

The most relevant prior work is that of Racunas and Patt, which proposes a combined load/store steering algorithm and a migration scheme for an architecture with multiple address generation units [44]. Load and store instructions are tagged with partition assignments which are kept in a partition assignment table (PAT) and accessed at decode time. These instructions are steered to the assigned load/store units, each of which has its own ache partition. If a miss occurs in the level-1 cache, the data is serviced from the level-2 cache; if the data was available in another cache bank, the instruction can be reassigned to that partition, or the data can be migrated to the instructions partition. Our work differs from that of Racunas and Patt in several substantial ways. Most importantly, we explore the interaction (particularly in steering policies) between processor clustering and data cache clustering, where Racunas and Patt focus solely on the memory system and assume a monolithic execution unit architecture for non-memory instructions. Second, we examine the effect of the floor plan on the performance of a fully clustered architecture. Finally, we study the trade-off between static data placement and dynamic data migration with replication.

2.3 Instruction Steering

In addition to mechanisms that control the mapping of the data in the primary data cache, the steering policies used to map instructions to the clustered execution resources have a significant impact on performance. Prior work on steering instructions to clusters has been based on using either load balance [6, 20] or instruction dependencies [12, 13, 42] to map instruction to execution clusters. Round robin steering maps groups of three consecutive instructions to each cluster in succession [6]. The advantage of this policy is that each cluster executes an equal number of instructions leading to perfect load balance across the execution units. However, with round robing steering an average of 86% of the instructions require an operand from a remote cluster. The large number of remote operands are a consequence of dependent instructions being mapped to different execution clusters. Dependence based steering as proposed by Palacharla et. al. [42] reduces the number of remote operands required by instructions by mapping instructions to the cluster that contains the producer of the value required by the instruction. This causes a sharp drop in the number of instructions that require a remote operand from the 86% for round robin steering to 8% for dependence steering. This drop in the number in remote operands while reducing the inter-cluster communication disrupts the load balance across the execution clusters. Even though dependence based steering outperforms round-robin steering for the architecture under investigation, it is on the opposite end of the spectrum as far as load balance and inter-cluster communication are concerned. Chapter 6 shows

how the predictive and hybrid steering policies proposed in this research find the middle ground between the good load balance of the round robin steering policy and low communication overhead of dependence based steering in order to achieve better performance than either of these two basic steering policies.

Chapter 3

Technology Trends

Development in silicon technology and decreases in transistor sizes has driven microprocessor performance improvements. Reduced feature sizes have provided two benefits. First, smaller transistors allow more devices on a single die, providing area for more complex microarchitectures. Second, technology scaling reduces transistor gate length and hence transistor switching time. If gate delay dominates microprocessor cycle times, greater quantities of faster transistors contribute directly to higher performance. However, faster clock rates and slower wires will limit the number of transistors reachable in a single cycle to be a small fraction of those available on a chip. Reducing the feature sizes has caused wire width and height to decrease, resulting in larger wire resistance due to smaller wire cross-sectional area. Unfortunately, wire capacitance has not decreased proportionally. Even though wire surface area is smaller, the spacing between wires on the same layer is decreasing with each technology generation. Consequently, the increased coupling capacitance to neighboring wires offsets the decreased parallel-plate capacitance.

This chapter describes the development of an analytical wire delay model based on empirical capacitance results from a 3D field solver. The

effects of technology scaling on chip-wide communication delays and clock rate are demonstrated using simple first-order models. Since the latency in clock cycles is more relevant than the absolute delay along a segment of wire, trends in the clock rate and the impact clock rate can have on wire latency are analyzed using the delay model. The analytical wire model augments a cache-timing model to estimate the access latencies of memory structures like caches, register files and instruction windows. The models that are developed are used to reason about how designers can expect future microarchitectures will scale.

3.1 Analytical Wire Model

Since the delay of a wire is directly proportional to the product of its resistance and capacitance, developing models for these parameters across all of the technology generations of interest is essential. To compute wire resistance per unit length ($\Omega/\mu m$), the simple equation $R_{wire} = \frac{\rho}{W \times H}$ is used, where ρ is wire resistance, W is wire width, and H is wire height. Computing capacitance per unit length ($fF/\mu m$) is more complicated due to the interactions among multiple conductors. To model the capacitance, empirical results obtained from Space3D, a three-dimensional field solver [60] are used. Wire capacitance includes components for conductors in lower and higher metal layers as well as coupling capacitance to neighboring wires in the same layer. For each fabrication technology, Space3D uses the geometry for a given wire with other wires running parallel to it on the same layer and perpendicular

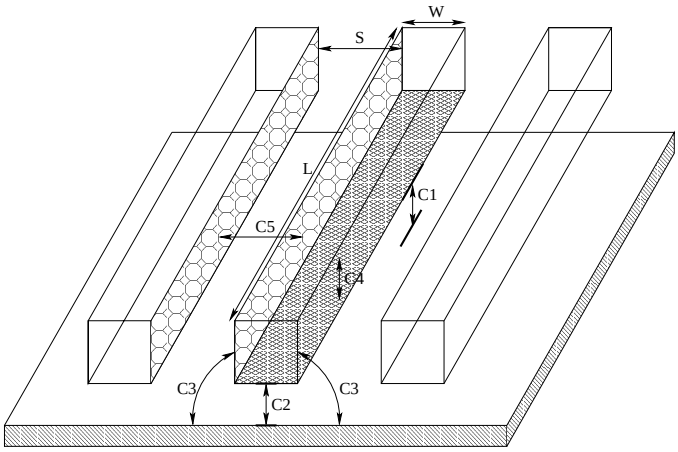


Figure 3.1: The components of the capacitance in our analytical capacitance model.

on the layers above and below. The layers above and below are assumed to have a dense distribution of wires on them. Space3D uses varying wire height, width, and spacing as inputs, and least-mean-squared curve fitting to derive the coefficients for the model. The capacitance model is optimistic compared to the worst-case environment of a wire in a real system because it assumes the grounding of all conductors other than the modeled wire, and thus not accounting for Miller-effect coupling capacitance.

The analytical capacitance model used the simple Equation 3.1 based on Figure 3.1 to generate the parasitic capacitance for a wire of length L, width W and having spacing S from the adjacent wires.

$$
\begin{aligned}
C_{wire} &= C_1 + C_2 + C_3 + C_4 + C_5 \\
&= L \times C_L + W \times C_W + S \times C_S + L \times W \times C_{LW} + \frac{L}{S} \times C_{LS} \quad (3.1)
\end{aligned}
$$

The first and second terms ($C_1 = L \times C_L$ and $C_2 = W \times C_W$) in this equation refer to the edge capacitance from the edges of the wire to the substrate. The third term ($C_3 = S \times C_S$) represents the increased fringing capacitance to ground as the wires are placed further and further apart. The fourth term ($C_4 = L \times W \times C_{LW}$) is the term representing the area capacitance to ground which goes up as the area of the wire increases. The last term ($C_5 = \frac{L}{S} \times C_{LS}$) represents the coupling capacitance to the adjacent wire, which increases as the length (L) of the wire increases and decreases as the wires a placed further apart (S). Table 3.1 shows the values of all five terms of our analytical model and the total capacitance for a $1\mu m$ wire in mid and top level metal from technologies from 250nm down to 35nm. As can be seen from the table capacitance per unit length of wire stays roughly constant across the various technology generations. The reason for the constant capacitance per unit length is that, as the capacitance to the substrate decreases as the wire dimensions get smaller, the coupling capacitance increases, as the placement of adjacent wires is closer to each other.

The derived values for R_{wire} and C_{wire} form the core of our wire delay model. Given the fabrication technology, and the wire length, width, and spacing, our model computes the end-to-end wire transmission delay. The

Gate (nm)	Level	$C_1(fF)$	$C_2(fF)$	$C_3(fF)$	$C_4(fF)$	$C_5(fF)$
250	mid	0.066	0.030	0.023	0.033	0.051
	top	0.066	0.042	0.032	0.046	0.036
180	mid	0.066	0.022	0.069	0.035	0.140
	top	0.065	0.034	0.063	0.033	0.154
130	mid	0.064	0.019	0.063	0.032	0.158
	top	0.061	0.024	0.062	0.029	0.182
100	mid	0.063	0.009	0.057	0.029	0.174
	top	0.061	0.021	0.054	0.026	0.198
70	mid	0.062	0.003	0.056	0.028	0.183
	top	0.060	0.013	0.058	0.026	0.204
50	mid	0.059	0.000	0.053	0.026	0.204
	top	0.058	0.004	0.058	0.025	0.213
35	mid	0.059	0.000	0.053	0.025	0.211
	top	0.056	0.001	0.052	0.023	0.233

Table 3.1: Terms in our capacitance model and their sum for a $1\mu m$ wire.

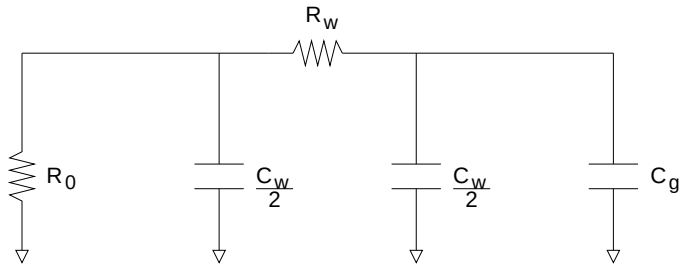


Figure 3.2: The RC network used to calculate wire delay in Equation 3.2 for a buffered wire

load on the remote end of the wire is a minimum-size inverter, which has a small gate capacitance relative to the wire capacitance. Optimal repeater placement is assumed in our model to reduce the delay's dependence on wire length from quadratic to linear. Each repeater is an inverter with PFET and NFET sizes chosen to minimize overall wire delay. A π circuit (as shown in figure 3.2) is used to model each wire segment in a repeated wire, as described in [4]. Equation 3.2 uses the delay of each segment to calculate overall delay as a function of wire length L.

$$D_{wire} = \frac{L}{l_0}(R_0(C_g + C_{wire}) + p + R_{wire}(\frac{C_{wire}}{2} + C_g)) \tag{3.2}$$

R_0 is the on-resistance of the repeater, C_g is the gate capacitance of the repeater, l_0 is the length of a wire segment between repeaters, p is the intrinsic delay of a repeater, and R_{wire} and C_{wire} are the resistance and capacitance of the wire segment between two repeaters. Using this equation, the transmission delay for a 5mm top-level wire more than doubles from 170ps to 390ps over the range of 250nm to 35nm technologies. When possible, increasing the wire width is an attractive strategy for reducing wire delay. Increasing the wire width and spacing by a factor of four for top level metal reduces the delay for a 5mm wire from 390ps to 210ps in a 35nm process, at a cost of four times the wire tracks for each signal. This study evaluates the wire widths shown in Table 3.2.

3.2 Wire Scaling

Our source for future technology parameters pertinent to wire delay is the 2001 ITRS [51]. Although the roadmap outlines the targets for future technologies, the parameters described within are not assured. Nonetheless, the model assumes that the roadmap's aggressive technology scaling predictions (particularly those for conductor resistivity ρ and dielectric permittivity κ) can be met. This research uses the roadmap's convention of subdividing the wiring layers into three categories: (1) *local* for connections within a cell, (2) *intermediate*, or mid-level, for connections across a module, and (3) *global*, or top-level, for chip-wide communication. To reduce communication delay, wires are both wider and taller in the mid-level and top-level metal layers. Our study of wire delay focuses on mid-level and top-level wires, using the wire width, height, and spacing projected in the roadmap.

Table 3.2 displays the wire parameters from 250nm to 35nm technologies, including the derived wire resistance per unit length (R_{wire}) and capacitance per unit length (C_{wire}) for both mid-level and top-level metal layers. The values are based on our analytical model as described in greater detail in Section 3.5. R_{wire} increases enormously across the technology parameters, with notable discontinuities at the transition to 180nm, due to copper wires, and 70nm, due to an anticipated drop in resistivity from materials improvements projected in the ITRS [51]. However, to limit the effect of shrinking wire width, wire aspect ratio (ratio of wire height to wire width) is predicted to increase up to a maximum of three. Larger aspect ratios increase the coupling

Gate Length (nm)	Dielectric Constant κ	Metal ρ ($\mu\Omega$-cm)	Mid-Level Metal Width (nm)	Aspect Ratio	R_{wire} ($m\Omega/\mu m$)	C_{wire} ($fF/\mu m$)
250	3.9	3.3	500	1.4	107	0.202
180	2.7	2.2	320	2.0	107	0.333
130	2.7	2.2	230	2.2	188	0.336
100	1.6	2.2	170	2.4	316	0.332
70	1.5	1.8	120	2.5	500	0.331
50	1.5	1.8	80	2.7	1020	0.341
35	1.5	1.8	60	2.9	1760	0.348

Gate Length (nm)	Dielectric Constant κ	Metal ρ ($\mu\Omega$-cm)	Top-Level Metal Width (nm)	Aspect Ratio	R_{wire} ($m\Omega/\mu m$)	C_{wire} ($fF/\mu m$)
250	3.9	3.3	700	2.0	34	0.222
180	2.7	2.2	530	2.2	36	0.350
130	2.7	2.2	380	2.5	61	0.359
100	1.6	2.2	280	2.7	103	0.361
70	1.5	1.8	200	2.8	164	0.360
50	1.5	1.8	140	2.9	321	0.358
35	1.5	1.8	90	3.0	714	0.366

Table 3.2: Projected fabrication technology parameters.

capacitance component of C_{wire}, which is somewhat mitigated by reductions in the dielectric constant of the insulator between the wires. Even with the advantages of improved materials, the intrinsic delay of a wire, $R_{wire} \times C_{wire}$, is increasing with every new technology generation. These results are similar to those found in other studies by Horowitz [27] and Sylvester [58].

3.3 Clock Scaling

While wires have slowed down, transistors have been getting dramatically faster. To first order, transistor switching time, and therefore gate delay, is directly proportional to the gate length. This research uses the *fanout-of-four* (FO4) delay metric to estimate circuit speeds independent of process technology technologies [27]. FO4 delay is the time for an inverter to drive four

copies of itself. Thus, a given circuit limited by transistor switching speed has the same delay measured in number of FO4 delays, regardless of technology. Reasonable models show that under typical conditions, the FO4 delay, measured in picoseconds (ps) is equal to $360 \times L_{drawn}$, where L_{drawn} is the minimum gate length for a technology, measured in microns. Using this approximation, the FO4 delay decreases from 90ps in a 250nm technology to 9ps in 50nm technology, resulting in circuit speeds improving by a factor of seven, just due to technology scaling.

The FO4 delay metric is important as it provides a fair means to measure processor clock speeds across technologies. The number of FO4 delays per clock period is an indicator of the number of levels of logic between on-chip latches. Microprocessors that have a small number of FO4 delays per clock period are more deeply pipelined than those with more FO4 delays per clock period. As shown by Kunkel and Smith [35], pipelining to arbitrary depth in hopes of increasing the clock rate does not result in higher performance. Overhead for the latches between pipeline stages becomes more significant as the number of levels of logic within a stage decreases too much. Pipelining in a microprocessor is also limited by dependencies between instructions in different pipeline stages. To execute two dependent instructions in consecutive clock cycles, the first instruction must compute its result in a single cycle. With current microarchitectures this requirement can be viewed as a lower bound on the amount of work that can be performed in a useful pipeline stage, and could be represented as the computation of an addition instruction. Under this

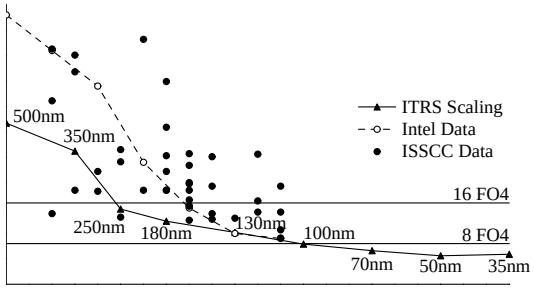


Figure 3.3: Clock scaling measured in FO4 inverter delays. The aggressive (8 FO4) and conservative (16 FO4) clocks are constant across technologies, but the ITRS roadmap projects less than 6 FO4 delays at 50nm and below.

assumption, a strict lower bound on the clock cycle time is 5.5 FO4 delays, which is the computation delay of a highly optimized 64-bit adder, as described by Naffziger [40]. When accounting for latch overhead and the time to bypass the output of the adder back to the input for the next instruction, reducing the clock period to 8 FO4 delays will present significant design challenges.

Figure 3.3 plots microprocessor clock periods (measured in FO4 delays) from 1992 to 2014. The horizontal lines represent the 8 FO4 and 16 FO4 clock periods. The clock periods projected by the ITRS shrink dramatically over the years and reach 5.6 FO4 delays at 50nm, before increasing slightly to 5.9 FO4 delays at 35nm. The Intel data represent five generations of x86 processors and show the reduction in the number of FO4 delays per pipeline stage from 53 in 1992 (i486DX2) to 15 in 2000 (Pentium III) to 10 in 2001 (Pentium 4) to 8.5 in 2004 (Pentium 4/Prescott), indicating substantially deeper pipelines. The isolated circles represent data from a wider variety of processors published in

Gate (nm)	Chip Area (mm^2)	16FO4 Clk f_{16} (GHz)	14FO4 Clk f_{14} (GHz)	12FO4 Clk f_{12} (GHz)	10FO4 Clk f_{10} (GHz)
250	400	0.69	0.79	0.93	1.11
180	450	0.97	1.10	1.29	1.54
130	567	1.34	1.53	1.78	2.14
100	622	1.74	1.98	2.31	2.78
70	713	2.48	2.83	3.31	3.97
50	817	3.47	3.97	4.63	5.56
35	937	4.96	5.67	6.61	7.94

Gate (nm)	Chip Area (mm^2)	8FO4 Clk f_8 (GHz)	6FO4 Clk f_6 (GHz)	ITRS Clk f_{ITRS} (GHz)
250	400	1.39	1.85	0.75
180	450	1.93	2.57	1.25
130	567	2.67	3.56	2.10
100	622	3.47	4.63	3.50
70	713	4.96	6.61	6.00
50	817	6.94	9.26	10.00
35	937	9.92	13.20	13.50

Table 3.3: Projected chip area and clock rate.

the proceedings of the International Solid State Circuits Conference (ISSCC) from 1994 to 2004. Both the Intel and ISSCC data demonstrate that clock rate improvements have come from a combination of technology scaling and deeper pipelining, with each improving approximately 15-20% per year. While the trend toward deeper pipelining will continue, reaching eight FO4 delays will be difficult, and attaining the ITRS projected clock rate is highly unlikely. Table 3.3 shows the resulting clock rates across the spectrum of technologies, assuming varying level of pipelining from six FO4 gate delay per pipeline stage to sixteen FO4 per pipeline stage.

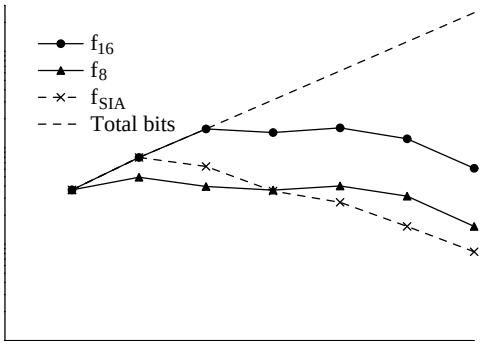


Figure 3.4: Reachable chip area in top-level metal, where area is measured in six-transistor SRAM cells.

3.4 Wire Delay Impact on Microarchitecture

The widening gap between the relative speeds of gates and wires will have a substantial impact on microarchitectures. With increasing clock rates, the distance that a signal can travel in a single clock cycle decreases. When combined with the growth in chip area anticipated for high-performance microprocessors, the time (measured in clock periods) to send a signal across one dimension of the chip will increase dramatically. Our analysis below uses the clock scaling described above and the projected chip areas from the ITRS, as shown in Table 3.3.

Based on the wire delay model, the chip area that is reachable in a single clock cycle is computed. Our unit of chip area is the size of a six-transistor SRAM cell, which shrinks as feature size is reduced. To normalize for different feature sizes across the technologies, the SRAM cell size is measured in λ,

which is equal to one-half the gate length in each technology. The SRAM cell is estimated to have an area of $700\lambda^2$, which is the median cell area from several recently published SRAM papers [9, 54, 65]. Our area metric does not include overheads found in real SRAM arrays, such as the area required for decoders, power distribution, and sense-amplifiers. Additionally, it does not reflect the size of a single-cycle access memory array; the area metric includes all bits reachable within a one-cycle, one-way transmission delay from a fixed location on the chip, ignoring parasitic capacitance from the SRAM cells.

Figure 3.4 shows the absolute number of bits that can be reached in a single clock cycle, which is termed *span*, using top-level wires for f_{16}, f_8, and f_{ITRS} clock scaling. The wire width and spacing are set to the minimum specified in the ITRS for top-level metal at each technology. Using f_{16} clock scaling, the span first increases as the number of bits on a chip increases and the entire chip can still be reached in a single cycle. As the chip becomes communication bound at 130nm, multiple cycles are required to transmit a signal across its diameter. In this region, decreases in SRAM cell size are offset equally by lower wire transmission velocity, resulting in a constant span. Finally, the span begins to decrease at 50nm when the wire aspect ratio stops increasing and resistance becomes more significant. The results for f_8 are similar except that the plateau occurs at 180nm and the span is a factor of four lower than that of f_{16}. However, in f_{ITRS} the span drops steadily after 180nm, because the clock rate is scaled superlinearly with decreasing gate length. These results demonstrate that clock scaling has a significant impact

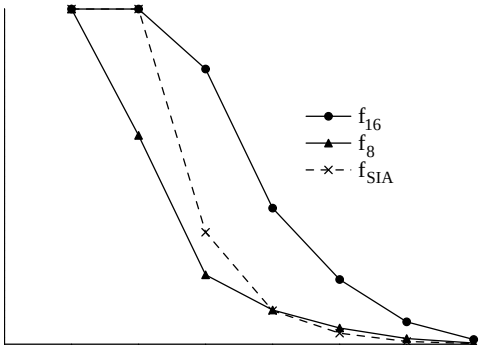


Figure 3.5: Fraction of total chip area reachable in one cycle.

on architectures as it demands a trade-off between the size and partitioning of structures. Using high clock rates to meet performance targets limits the size of pipeline stages and microarchitectural structures, while tightly constraining their placement. If lower clock rates can be tolerated, then microarchitects can give less consideration to the communication delay to reach large and remote structures.

Figure 3.5 shows the fraction of the total chip area that can be reached in a single clock cycle. Using f_8 in a 35nm technology, less than 0.4% of the chip area can be reached in one cycle. Even with f_{16}, only 1.4% of the chip can be reached in one cycle. Similar results have been observed in prior work [39]. If microarchitectures do not change over time, this phenomenon would be unimportant, since the area required to implement them would decrease with feature size. However, microarchitectures have become more complex because architects acquired more transistors with each new fabrication technology, and

used them to improve overall performance. In future technologies, substantial delay penalties must be paid to reach the state or logic in a remote region of the chip, so microarchitectures that rely on large structures and global communication will face more serious challenges in the future than they do today.

3.5 ECACTI Analytical Models

This research develops an analytical access time model in order to study the effect of the various technology trends on the access time of microarchitectural structures. To achieve this, we modify an existing cache model (ECACTI) using the analytical wire model described in section 3.4. This section explains the modifications to ECACTI used to incorporate the analytical wire model.

To model the various storage-oriented components of a modern microprocessor, an extended version of the original CACTI cache-modeling tool [62], called ECACTI [46] is used. Given the capacity, block size, associativity, number ports, and number of data and address bits, ECACTI considers a number of alternative cache organizations and computes the minimum access time. ECACTI automatically splits the cache into banks and chooses the number and layout of banks that incurs the lowest delay. When modeling large memory arrays, ECACTI presumes multiple decoders, with each decoder serving a small number of banks. For example with a 4MB array, ECACTI produces 16 banks and 4 decoders in a 35nm technology. Note that this model is op-

timistic, because it does not account for driving the address from a central point to each of the distributed decoders.

This research modifies ECACTI to include technology scaling, using the projected parameters from the ITRS. SRAM cell sizes and transistor parasitics, such as source and drain capacitances, scale according to their anticipated reduction in feature size for future technologies. The word-lines assume that they run from a decoder across its neighboring banks in mid-level metal, and that the world lines in mid-level metal do not affect the size of the SRAM cell. The model assumes that parameters that are not explicitly specified in the ITRS obey simple scaling rules e.g. load capacitance decreases linearly with feature size. Unlike Amrutur and Horowitz [4] this model makes the optimistic assumption that the sense-amplifier threshold voltage will decrease linearly with technology, which gives better sense times than if the threshold voltage does not scale.

The input parameters to ECACTI are the following (in order):

- Size of cache in bytes
- Block size in bytes
- Associativity (FA for fully associative)
- Number of read-write ports (optional, default 1)
- Number of extra read ports (optional)

- Number of extra write ports (optional)
- Number of sub-banks in the cache

An example of a typical ECACTI run is as follows:

```
cacti.05 65536 32 2 1 0 0 2
```

The above example is a simulation of a 64KB, 1-ported, 2-way set associative cache in a 50 nm technology with a block size of 32 bytes. Such a run will produce an output like:

```
Cache Parameters:
  Number of Subbanks: 2
  Total Cache Size: 65536
  Size in bytes of Subbank: 32768
  Number of sets: 512
  Associativity: 2
  Block Size (bytes): 32
  Read/Write Ports: 1
  Read Ports: 0
  Write Ports: 0
  Technology Size: 0.03um
  Vdd: 0.4V

Access Time: 0.207206
```

```
Cycle Time (wave pipelined) (ns):  0.0864705
Total Power all Banks (nJ): 0.052435
Total Power Without Routing (nJ): 0.052435
Total Routing Power (nJ): 0
Maximum Bank Power (nJ):  0.0262175

Best Ndwl (L1): 2
Best Ndbl (L1): 2
Best Nspd (L1): 1
Best Ntwl (L1): 1
Best Ntbl (L1): 4
Best Ntspd (L1): 2
Nor inputs (data): 3
Nor inputs (tag): 2

Area Components:

Aspect Ratio Total height/width: 2.118090

Data array (cm^2): 0.000642
Data predecode (cm^2): 0.000002
Data colmux predecode (cm^2): 0.000000
Data colmux post decode (cm^2): 0.000000
```

```
Data write signal (cm^2): 0.000000

Tag array (cm^2): 0.000069
Tag predecode (cm^2): 0.000001
Tag colmux predecode (cm^2): 0.000000
Tag colmux post decode (cm^2): 0.000000
Tag output driver decode (cm^2): 0.000001
Tag output driver enable signals (cm^2): 0.000000

Percentage of data ramcells alone of total area: 75.578653 %
Percentage of tag ramcells alone of total area: 5.609353 %
Percentage of total control alone of total area: 18.811993 %

Subbank Efficiency : 81.188007
Total Efficiency : 45.010768

Total area One Subbank (cm^2): 0.000716
Total area subbanked (cm^2): 0.002584

Time Components:
 data side (with Output driver) (ns): 0.205683
 tag side (with Output driver) (ns): 0.207206
 address routing delay (ns): 0
```

```
address routing power (nJ): 0
decode_data (ns): 0.0864705
            (nJ): 0.000536434
wordline and bitline data (ns): 0.0761042
           wordline power (nJ): 7.30985e-06
            bitline power (nJ): 0.00143317
sense_amp_data (ns): 0.0159375
               (nJ): 0.0184999
decode_tag (ns): 0.0301116
           (nJ): 0.000187212
wordline and bitline tag (ns): 0.0278823
          wordline power (nJ): 2.86444e-06
           bitline power (nJ): 0.000141301
sense_amp_tag (ns): 0.0159375
              (nJ): 0.00512602
compare (ns): 0.0319608
        (nJ): 1.75964e-05
mux driver (ns): 0.0694994
           (nJ): 3.56579e-05
sel inverter (ns): 0.00464361
             (nJ): 4.86469e-07
data output driver (ns): 0.0271711
                   (nJ): 0.000229532
```

```
total_out_driver (ns): 0
                (nJ): 0
total data path (without output driver) (ns): 0.178512
total tag path is set assoc (ns): 0.180035
```

Note that there exists a family of ECACTI simulators with one for each technology generation from 250nm down to 35nm. The example show above corresponds to using the simulator for the 50nm technology. Instead, to model the same cache in a 35nm technology, use cacti.03.

Aside from modeling direct-mapped and set associative caches, the extended version of ECACTI is used to explore other microarchitectural structures. For example, a register file is essentially a direct mapped cache with more ports, but fewer address and data bits than a typical L1 data cache. Content addressable memories (CAMs) are modeled as fully associative structures with the appropriate number of tag and data bits per entry. A similar methodology is used to examine issue windows, reorder buffers, branch prediction tables, and TLBs.

3.6 Microarchitectural Structures

In addition to reducing the chip area reachable in a clock cycle, both the widening gap between wire and gate delays and superlinear clock scaling has a direct impact on the scaling of microarchitectural structures in future

microprocessors. Clock scaling is more significant than wire delay for small structures, while both wire delay and clock scaling are significant in larger structures. The large memory-oriented elements, such as the caches, register files, instruction windows, and reorder buffers, will be unable to continue increasing in size while remaining accessible within one clock cycle. This section uses the analytical models to examine the access time of different structures from 250nm to 35nm technologies based on the structure organization and capacity. The results demonstrate the trade-offs between access time and capacity that are necessary for the various structures across the technology generations. The next chapter investigates the impact on processor performance of the trade-off between the access time and capacity of various microarchitectural structures.

3.6.1 Caches

The extended version of ECACTI, estimates memory structure access times, while varying cache capacity, block size, associativity, number of ports, and process technology. While cache organization characteristics do affect access time, the most critical characteristic is capacity. Figure 3.6 plots the access time versus capacity for a dual-ported, two-way set associative cache. The maximum cache capacities that can be reached in 3 cycles for the f_{16}, f_8 and f_{ITRS} clocks are also plotted as "isobars". Note that the capacity for a three cycle access cache decreases moderately for f_{16} and f_8, but falls off the graph for f_{ITRS}.

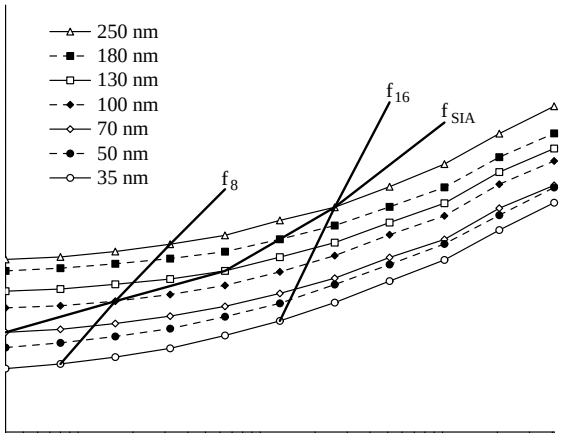


Figure 3.6: Access time for various L1 data cache capacities.

For each technology, the access time increases as the cache capacity increases. Even with substantial banking, the access time goes up dramatically at capacities greater than 256KB. For a given cache capacity, the transition to smaller feature sizes decreases the cache access time, but not as fast as projected increases in clock rates. In a 35nm technology, a 32KB cache takes one to six cycles to access depending on the clock frequency. One alternative to slower clocks or smaller caches is to pipeline cache accesses and allow each access to complete in multiple cycles. Due to the non-linear scaling of capacity with access time, adding a small number of cycles to the cache access time substantially increases the available cache capacity. For example, increasing the access latency from four to seven cycles increases the reachable cache capacity by about a factor of 16 in a 35nm technology. The results shown in Figure 3.6 apply to all of the cache-like microarchitectural structures that are

Gate (nm)	16FO4 Clk	14FO4 Clk	12FO4 Clk	10FO4 Clk	8FO4 Clk	6FO4 Clk	ITRS Clk
250	2	2	3	3	4	5	2
180	2	2	3	3	4	5	3
130	2	2	3	3	4	5	3
100	2	2	3	3	4	5	4
70	2	2	3	3	4	5	5
50	2	3	3	4	4	6	6
35	3	3	3	4	5	6	6

Table 3.4: Latency in cycles for a 64KB, 2-way set associative, 2 ported, 32B block cache in various technologies.

examined in this study, including L1 instruction and data caches, L2 caches, register files, branch target buffers, and branch prediction tables.

Table 3.4 shows the latency in clock cycles to access a 64KB, 2-way set associative, 2 ported, 32B block size cache for the various clock rates specified in Table 3.3. A comprehensive set of access times for various cache configurations and technology generations is shown in Tables A.1 to A.14 in Appendix A.

3.6.2 Register Files

While our cache model replicates current design methodologies, our register file model is more aggressive in design. Although register files have traditionally been built using single-ended full swing bit lines [49], larger capacity register files will need faster access provided by differential bit-lines and low-voltage swing sense-amplifiers similar to those in our model. For our register file modeling, the cache block size is set to the register width and

Gate (nm)	16FO4 Clk	14FO4 Clk	12FO4 Clk	10FO4 Clk	8FO4 Clk	6FO4 Clk	ITRS Clk
250	1	2	2	2	2	3	1
180	1	2	2	2	2	3	2
130	1	2	2	2	2	3	2
100	1	2	2	2	2	3	2
70	1	2	2	2	2	3	3
50	2	2	2	2	3	3	3
35	2	2	2	2	3	3	3

Table 3.5: Latency in cycles for a 128 entry, 10 ported, 80-bits per entry register file in various technologies.

the associativity is set to 1. The main difference between a register file and direct mapped cache is that the register file has a significantly higher number of ports. For multi-ported register files, the size of each cell in the register file increases linearly in both dimensions with the number of ports. Also, when modeling a register file, the number of output bits needs to match the size of each entry in the register file.

Our capacity results for the register file are similar to those seen in caches. Our results show that register files with many ports will incur larger access times. For example, in a 35nm technology, going from ten ports to 32 ports increases the access time of a 64-entry register file from 172ps to 274ps. Increased physical size and access time makes attaching more execution units to a single global register file impractical. Table 3.5 shows the latency in clock cycles to access a 128 entry, 10 ported, 80 bits per entry register file for the various clock rates specified in Table 3.3. A comprehensive set of access times

for various simple register file configuration from a 250nm technology down to a 35nm technology can be found in Tables A.15 to A.21 in Appendix A.

3.6.3 Content Addressable Memories

Lastly, we model components that require global address matching within the structure, such as the instruction window and the TLB. These components are typically implemented as content addressable memories (CAMs) and can be modeled using the tag array of a fully associative cache. Our initial model of the instruction window includes a combination of an eight-bit wide CAM and a 40-bit wide direct mapped data array for the contents of each entry. The issue window has eight ports, which are used to insert four instructions, write back four results, and extract four instructions simultaneously. Since the eight-ported CAM cell and corresponding eight-ported data array cell are assumed to be port-limited, the area of these cells is computed based on the width and number of bit-lines and word-lines used to access them. Note that only the structure access time is modeled and not the latency of the instruction selection logic.

Figure 3.7 shows the access time for this configuration as a function of the number of instructions in the window. As with all of the memory structures, the access time increases with capacity. The increase in access time is not as significant as in the case of the caches, because the capacities considered are small and all must pay an almost identical penalty for the fully associative match on the tag bits. Thus, in this structure, once the initial

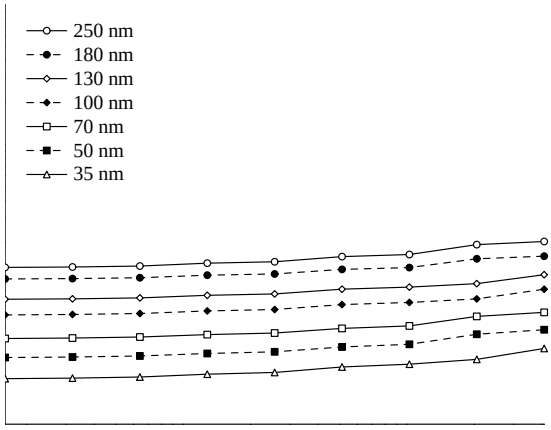


Figure 3.7: Access time vs. issue window size across technologies.

tag match delay has been computed, the delay for the rest of the array does not increase significantly with capacity. A 128-entry, eight-port, eight-bit tag instruction window has an access time of 227ps in a 35nm process, while a 12-bit tag raises the access time of a same size window to 229ps. A 128-entry, 32-port, eight-bit tag instruction window (as might be required by a 16-issue processor) has an access time of 259ps in a 35nm technology. Note that all of these results ignore the increase in complexity of the selection logic as the issue window size and the port count in the issue window increases. It is anticipated that the capacity and port count of the register file and the complexity of the selection logic will ultimately place a limit on the issue width of superscalar microarchitectures [42].

A comprehensive set of access times for various simple CAM configuration from a 250nm technology down to a 35nm technology can be found in

Tables A.22 to A.28 in Appendix A.

3.6.4 Validation

This research validates the modified ECACTI model in order to achieve greater confidence in the access times predicted by the model. We compare the latencies output by the analytical models to HSPICE simulations of corresponding circuits. The HSPICE simulations show that with simple linear technology scaling, below 100nm, the delay of an inverter simulated in HSPICE does not match expected delay from analytical models. In order to achieve a better match between the HSPICE and analytical models, we adjust the CMOS transistor parameters. The list of adjusted HSPICE model parameters includes the substrate doping, the threshold voltage and the mobility.

In order to better validate our analytical structure model based on the ECACTI code, the access times of various structures as predicted by the analytical model is compared to the latency output by a netlist representing a cross-section of the circuit simulated in ECACTI. The access times generated by ECACTI and the HSPICE deck for a 2-ported, 2-way associative cache are listed in Table 3.6. This level of accuracy is comparable to the accuracy of the original ECACTI model when compared to equivalent HSPICE results. The ECACTI and HSPICE models start diverging at small feature sizes and large capacities. The divergence is the result of the lack of accuracy of the *pi* model for long wires, used to estimate wire delay in the HSPICE netlist.

Our analytical model compares favorably to other models and related

Technology	Capacity	4	8	16	32	64	128	256	512
250nm	ECACTI	1.37	1.43	1.54	1.67	1.85	2.17	2.55	3.30
	HSPICE	1.06	1.10	1.35	1.43	1.69	2.28	2.70	3.52
180nm	ECACTI	1.17	1.21	1.29	1.40	1.54	1.72	2.01	2.58
	HSPICE	0.94	0.97	1.14	1.20	1.70	1.81	2.07	2.56
130nm	ECACTI	0.83	0.86	0.92	1.00	1.11	1.25	1.49	1.95
	HSPICE	0.67	0.70	0.85	0.90	1.09	1.49	1.73	2.12
100nm	ECACTI	0.63	0.66	0.70	0.77	0.85	0.99	1.19	1.60
	HSPICE	0.45	0.49	0.56	0.71	0.77	1.17	1.44	1.86
70nm	ECACTI	0.41	0.43	0.47	0.51	0.58	0.67	0.81	1.10
	HSPICE	0.31	0.34	0.34	0.47	0.55	1.00	1.25	1.67
50nm	ECACTI	0.31	0.32	0.35	0.49	0.46	0.54	0.68	0.94
	HSPICE	0.23	0.34	0.46	0.62	0.67	1.01	1.22	1.62
35nm	ECACTI	0.21	0.23	0.25	0.28	0.33	0.39	0.50	0.70
	HSPICE	0.28	0.29	0.25	0.43	0.47	1.12	1.23	1.57

Table 3.6: Comparison of access times (ns) generated from ECACTI versus access time generated by the SPICE deck for a 2-way set associative cache.

implementations. In a 250nm technology, the access time for a 64KB L1 data cache is computed to be 2.4ns. This access time is comparable to that of the 700MHz Alpha 21264 L1 data cache. Furthermore, for a 4MB cache in a 70nm technology, our model predicts an access time of 33 FO4 delays which matches the 33 FO4 access time generated by Amrutur and Horowitz for a similar cache [4].

3.7 Summary

While transistor speeds are scaling approximately linearly with feature size, wires are getting slower with each new technology. Even assuming low-

resistivity conductors, low-permittivity dielectrics, and higher aspect ratios, the *absolute* delay for a fixed-length wire in top-level metal with optimally placed repeaters is increasing with each generation. Only when the wire width and spacing is increased substantially can the wire delay be kept constant. Due to increasing clock frequencies, wire delays are increasing at an even higher rate. As a result, chip performance will no longer be determined solely by the number of transistors that can be fabricated on a single integrated circuit (capacity bound), but instead will depend upon the amount of state and logic that can be reached in a sufficiently small number of clock cycles (communication bound).

The argument made by Sylvester and Keutzer [58] that wire delays will not affect future chip performance holds only if wire lengths are reduced along with gate lengths in future technologies. Traditional microprocessor microarchitectures have grown in complexity with each technology generation, using all of the silicon area for a single monolithic core. Current trends in microarchitectures have increased the sizes of all of the structures, and added more execution units. With future wire delays, structure size will be limited and the time to bypass results between pipeline stages will grow. If clock rates increase at their projected rates, both of these effects will have substantial impact on instruction throughput.

Because of increasing wire delays and faster transistors, memory oriented microarchitectural structures are not scaling with technology. To access caches, register files, branch prediction tables, and instruction windows in a

Structure Name	f_{ITRS}	f_8	f_{16}
L1 cache 64K (2 ports)	7	5	3
Integer register file 64 entry (10 ports)	3	2	1
Integer issue window 20 entry (8 ports)	3	2	1
Reorder buffer 64 entry (8 ports)	3	2	1

Table 3.7: Projected access time (cycles) at 35nm.

single cycle will require the capacity of these structures to decrease as clock rates increase. In Table 3.7 shows the number of cycles needed to access the structures from the Compaq Alpha 21264, scaled to a 35nm process for each of the three methods of clock scaling. With constant structure capacities, the L1 cache will take up to seven cycles to access, depending on how aggressively the clock is scaled.

The next chapter reports experimental results that show how the projected scaling of microarchitectural components affects overall processor performance. Using the results of the analytical models discussed in this chapter as inputs to SimpleScalar-based timing simulation, the performance of current microarchitectures when scaled from 250nm to 35nm technology are investigated.

Chapter 4

Scaling of Conventional Architectures

As communication delays increase relative to computation delays, superlinear clock rate scaling and today's techniques for exploiting instruction level parallelism (ILP) work against one another. The wire model in the previous chapter demonstrated that aggressively scaling the clock reduces the amount of state that can be used to exploit ILP for a fixed pipeline depth. The designer is faced with two interacting choices: how aggressively to push the clock rate by reducing the number of levels of logic per cycle, and how to scale the size and pipeline depth of different microarchitectural structures. Using the results of the analytical models from the previous chapter as inputs to SimpleScalar-based timing simulation, we track the performance of current microarchitectures when scaled from 250nm to 35nm technology, using different approaches for scaling the clock and the microarchitecture. We measure the effect of three different clock scaling strategies on the microarchitecture: setting the clock at 16 *fanout-of-four* (FO4) delays, setting the clock at 8 FO4 delays, and scaling the clock according to the ITRS projections. For a given target frequency, we define two approaches for scaling the microarchitecture to smaller technologies:

- *Capacity scaling*: Shrink the microarchitectural structures such as caches, register files and re-order buffers, sufficiently so that their access penalties are constant across technologies. We define *access penalty* as the access time for a structure measured in clock cycles.

- *Pipeline scaling*: Hold the capacity of a structure constant and increase the pipeline depth as necessary to cover the increased latency across technologies.

While these trade-offs of clock versus structure size scaling manifest themselves in every design process, their importance will grow as communication delay creates more interference between clock speed and ILP optimizations.

This chapter explores the effect of capacity and pipeline scaling strategies upon instructions per cycle (IPC) and overall performance. For each scaling strategy, and at three different clock scaling rates (f_{16}, f_8, and f_{ITRS} projections), we measure IPC for our target microarchitecture at technologies ranging from 250nm to 35nm. The analysis will show that a monolithic microarchitecture does not scale well at high clock rates, motivating the design of clustered architectures.

4.1 Experimental Methodology

We perform our microarchitectural timing simulations with an extended version of the SimpleScalar tool set [10], version 3.0. We also use the Sim-

pleScalar memory hierarchy extensions, which simulate a one-level page table, hardware TLB miss handling, finite miss status holding registers (MSHRs) [34], and simulation of bus contention at all levels [11].

4.1.1 Target Microprocessor

The SimpleScalar tools use a Register Update Unit (RUU) [56] to track out-of-order issue of instructions. The RUU acts as a unified reorder buffer, issue window, and physical register file. Because of the large number of ports required for the RUU in a four-wide machine, implementing it is not feasible at high clock frequencies. We therefore split the structures logically in SimpleScalar, effectively simulating a separate reorder buffer, issue window, and physical register file.

We also modified SimpleScalar's target processor core to model a four-wide superscalar pipeline organization roughly comparable to the Compaq Alpha 21264 [31]. Our target is intended to model a microarchitecture typical of those found in current-generation processors; it is *not* intended to model the 21264 microarchitecture itself. However, we chose as many parameters as possible to resemble the 21264, including the pipeline organization and microarchitectural structure capacities. Our target processor uses a seven-stage pipeline for integer operations in our simulated base case, with the following stages: fetch, decode, map, queue, register read, execute, and writeback. As in the 21264, reorder buffer accesses occur off of the critical path, physical registers are read after instruction issue, and instructions are loaded into the

issue queues in program order.

Our target differs from the 21264 in the following respects. We assume that the branch target buffer is a distinct structure, as opposed to a line predictor embedded in the instruction cache. We simulate a two-level gshare predictor instead of the 21264's local history, global history, and choice predictors. We remove instructions from the issue queue immediately after they are issued, rather than implementing load-use speculation and waiting two cycles for its resolution. We simulate a combined floating-point and integer issue queue (they are split in the 21264), but model them from a timing perspective as if they were split. We do not implement the functional unit and register file clustering found in the 21264. Instead of the six instructions per cycle that the 21264 can issue (four integer and two floating-point), we permitted only four. We use the default SimpleScalar issue prioritization policy, which issues ready loads with highest priority, followed by the oldest ready instructions. The 21264 always issues the oldest ready instruction regardless of its type. Finally, we do not simulate the 21264 victim cache, which contains an eight-entry victim buffer.

4.1.2 Simulation Parameters

Our baseline processor parameters include the following: a four-way issue superscalar processor with a 40-entry issue window for both integer and floating point operations, a 64-entry load/store queue, commit bandwidth of eight instructions per cycle, an eight-entry return address stack, and a branch

	Capacity (bits)	# entries	Bits/entry	Ports
Branch pred.	32K	16K	2	1
BTB	48K	512	96	1
Reorder buffer	8K	64	128	8
Issue window	800/160	20	40	8
Integer RF	5K	80	64	10
FP RF	5K	80	64	10
L1 I-Cache	512K	1K	512	1
L1 D-Cache	512K	1K	512	2
L2 Cache	16M	16K	1024	2
I-TLB	14K	128	112	2
D-TLB	14K	128	112	2

Table 4.1: Capacities of structures used in delay calculations

mispredict rollback latency equal to the reorder buffer access delay plus three cycles. We set the number and delays of the execution units to those of the 21264 [31]. We show the default sizes of the remaining structures in Table 4.1.

In our baseline memory system, we use separate 64KB, two-way set associative level-one instruction and data caches, with a 2MB, four-way set-associative, unified level-two cache. The L1 caches have 64-byte lines, and the L2 cache has 128-byte lines. The L1/L2 cache bus is 128 bits wide, requires one cycle for arbitration, and runs at the same speed as the processing core. Each cache contains eight MSHRs with four combining targets per MSHR.

We assume a DRAM system that is aggressive by today's standards. The L2/memory bus is 64 bits wide, requires one bus cycle for arbitration, and runs at half the processor core speed. That speed ratio, assuming a tightly coupled electrical interface to memory, is similar to modern Rambus memory channels. We assumed that the base DRAM access time is 50ns, plus

a somewhat arbitrary 20ns for the memory controller. That access time of 70ns is typical for what was available in 1997. For each year beyond 1997, we reduce the DRAM access time by 10%, using that delay and the clock rate for a given year to compute the DRAM access penalty, in cycles, for the year in question. We model access time and bus contention to the DRAM array, but do not model page hits, precharging overhead, refresh cycles, or bank contention. The benchmarks we use exhibit low miss rates from the large L2 caches in our simulations. We simulate a subset of the SPEC CPU2000 [25] benchmarks for 500 million instructions using the `ref` input set.

4.1.3 Modeling Deeper Pipelines

To simulate pipeline scaling in SimpleScalar, we added the capability to simulate variable-length pipelines by specifying the access latency for each major structure as a command-line parameter. We assume that all structures can begin at most m accesses every cycle, where m is the number of ports. We do not account for any pipelining overheads due to latches or clock skew [35]. Furthermore, we assume that an n-cycle access to a structure will cause an $n-1$ cycle pipeline stall as specified below. We perform the following accounting for access delays in which $n > 1$:

- *Instruction-Cache*: Pipeline stalls affect performance only when a branch is predicted taken. We assume that fetches with no changes in control flow can be pipelined.

- *Issue window*: Additional cycles to access the issue window cause delays when instructions are removed from the queue (wakeup and select), not when instructions are written into the issue window.

- *Reorder buffer*: As in the 21264, writes to the reorder buffer and commits occur off the critical path. We therefore add reorder buffer delays only to the rollback latency, which in turn is incurred only upon a branch misprediction.

- *Physical register file*: Register access penalties are paid only on non-bypassed register file reads. Register file writes are queued and bypassed to dependent instructions directly.

- *Branch predictor and BTB*: Multi-cycle predictor accesses create pipeline bubbles only when a prediction must be made. Multi-cycle BTB accesses cause the pipeline to stall only when a branch is predicted taken. We assume that the two structures are accessed in parallel, so that pipeline bubbles will be the maximum, rather than the sum, of the two delays.

4.2 Pipeline versus Capacity Scaling

We use the simulated microarchitecture described above to measure IPC across the benchmark suite. The access latencies for the microarchitectural structures are derived from the models described in Chapter 3. From the access latencies, we compute the access penalties for each of the three clock scaling rates (f_{16}, f_8, and f_{ITRS}).

Table 4.1 shows both the number of bits per entry and the number of ports for each of the baseline structures. These parameters are used to compute the delay as a function of capacity and technology as well as the capacity as a function of access penalty and technology. In the issue window entry, we show two bit capacities, one for the instruction queue and one for the tag-matching CAM.

The third column of Table 4.1 shows the baseline structure sizes that we use for our pipeline scaling experiments. Table 4.2 shows the actual access penalty of the structures for the fixed baseline capacities. Note that as the feature size decreases, the access penalty to the fixed size structures increases, and is dependent on the clock rate. Consequently, deeper pipelines are required as a fixed-capacity microarchitecture is scaled to smaller technologies.

Table 4.3 shows the parameters for the capacity scaling experiments. Because the access penalties are held nearly constant, the capacities of the structures decrease dramatically as smaller technologies and higher clock rates are used. For each technology generation, we set each structure to be the maximum size possible while ensuring that it remains accessible in the same number of cycles as our base case (one cycle for most of the structures, and ten cycles for the L2 cache).

In future technologies, some of the structures become too small to be useful at their target access penalty. In such cases, we increase the access penalty slightly, permitting a structure that is large enough to be useful. The access penalties are shown in the subscripts in Table 4.3. Note that for tech-

Structure	250nm $f_{16}/f_8/f_{ITRS}$	180nm $f_{16}/f_8/f_{ITRS}$	130nm $f_{16}/f_8/f_{ITRS}$	100nm $f_{16}/f_8/f_{ITRS}$
Branch pred.	1/2/2	2/3/2	2/3/2	2/3/3
BTB	1/2/1	1/2/2	1/2/2	1/2/2
Reorder buffer	1/2/1	1/2/2	1/2/2	1/2/2
Issue window	1/2/1	1/2/2	1/2/2	1/2/2
Integer RF	1/2/1	1/2/2	1/2/2	1/2/2
FP RF	1/2/1	1/2/2	1/2/2	1/2/2
L1 I-Cache	2/3/2	2/3/2	2/3/3	2/3/3
L1 D-Cache	2/4/2	2/4/3	2/4/3	2/4/4
L2 Cache	11/21/11	10/19/12	11/21/17	11/23/23
I-TLB	2/3/2	2/3/2	2/3/3	2/3/3
D-TLB	2/3/2	2/3/2	2/3/3	2/3/3

Structure	70nm $f_{16}/f_8/f_{ITRS}$	50nm $f_{16}/f_8/f_{ITRS}$	35nm $f_{16}/f_8/f_{ITRS}$
Branch pred.	2/3/3	2/3/4	2/3/4
BTB	1/2/3	1/2/3	1/2/3
Reorder buffer	1/2/3	1/2/3	1/2/3
Issue window	1/2/3	1/2/3	1/2/3
Integer RF	1/2/2	1/2/3	1/2/3
FP RF	1/2/2	1/2/3	1/2/3
L1 I-Cache	2/3/4	2/4/5	2/4/5
L1 D-Cache	2/4/5	3/5/7	3/5/7
L2 Cache	12/24/29	17/34/49	19/38/52
I-TLB	2/3/4	2/3/4	2/3/4
D-TLB	2/3/4	2/3/4	2/3/4

Table 4.2: Access times (in cycles) using pipeline scaling with f_{16}, f_8, and f_{ITRS} clock scaling.

Structure	250nm $f_{16}/f_8/f_{ITRS}$	180nm $f_{16}/f_8/f_{ITRS}$
BPred	$8K_1/8K_2/8K_1$	$8K_1/8K_2/1K_1$
BTB	$1K_1/1K_2/512_1$	$512_1/512_2/4K_2$
ROB	$256_1/512_2/128_1$	$128_1/128_2/8K_2$
IW	$512_1/512_2/128_1$	$64_1/64_2/8K_2$
Int. RF	$256_1/256_2/128_1$	$256_1/256_2/35_1$
FP RF	$256_1/256_2/128_1$	$256_1/256_2/35_1$
L1 I$	$256K_2/64K_3/256K_2$	$256K_2/64K_3/64K_2$
L1 D$	$64K_2/32K_3/64K_2$	$64K_2/16K_3/32K_2$
$L2_{10}$	2M/256K/1M	2M/256K/1M
I-TLB	$32K_2/512_3/32K_2$	$32K_2/512_3/4K_2$
D-TLB	$32K_2/512_3/32K_2$	$32K_2/512_3/4K_2$

Structure	130nm $f_{16}/f_8/f_{ITRS}$	100nm $f_{16}/f_8/f_{ITRS}$
BPred	$4K_1/8K_2/256_1$	$4K_1/4K_2/4K_2$
BTB	$512_1/512_2/2K_2$	$512_1/512_2/512_2$
ROB	$128_1/128_2/2K_3$	$128_1/128_2/128_2$
IW	$64_1/64_2/2K_2$	$64_1/64_2/64_2$
Int. RF	$128_1/128_2/512_2$	$128_1/128_2/128_2$
FP RF	$128_1/128_2/512_2$	$128_1/128_2/128_2$
L1 I$	$256K_2/64K_3/16K_2$	$256K_2/64K_3/64K_3$
L1 D$	$64K_2/16K_3/2K_2$	$64K_2/16K_3/16K_3$
$L2_{10}$	1M/256K/1M	1M/256K/256K
I-TLB	$32K_2/512_3/4K_3$	$16K_2/512_3/2K_3$
D-TLB	$32K_2/512_3/4K_3$	$16K_2/512_3/2K_3$

Structure	70nm $f_{16}/f_8/f_{ITRS}$	50nm $f_{16}/f_8/f_{ITRS}$	35nm $f_{16}/f_8/f_{ITRS}$
BPred	$4K_1/4K_2/2K_2$	$4K_1/4K_2/256_2$	$4K_1/4K_2/512_2$
BTB	$512_1/512_2/128_2$	$256_1/256_2/512_3$	$256_1/256_2/512_3$
ROB	$128_1/128_2/2K_3$	$64_1/64_2/256_4$	$64_1/64_2/256_3$
IW	$64_1/64_2/2K_3$	$64_1/64_2/128_3$	$64_1/64_2/256_3$
Int. RF	$128_1/128_2/64_2$	$128_1/128_2/128_3$	$128_1/128_2/128_3$
FP RF	$128_1/128_2/64_2$	$128_1/128_2/128_3$	$128_1/128_2/128_3$
L1 I$	$128K_2/64K_3/16K_3$	$128K_2/32K_3/32K_4$	$128K_2/32K_3/32K_4$
L1 D$	$64K_2/16K_3/4K_3$	$32K_2/8K_3/4K_4$	$32K_2/8K_3/8K_4$
$L2_{10}$	1M/256K/128K	$512K/256K_{15}/128K_{15}$	$512K/256K_{15}/128K_{15}$
I-TLB	$16K_2/512_3/4K_4$	$16K_2/256_3/1K_4$	$16K_2/256_3/1K_4$
D-TLB	$16K_2/512_3/4K_4$	$16K_2/256_3/1K_4$	$16K_2/256_3/1K_4$

Table 4.3: Structure sizes and access times (in subscripts) using capacity scaling with f_{16}, f_8, and f_{ITRS} clock scaling.

nologies smaller than 130nm, no structure can be accessed in less than two cycles for the f_8 and f_{ITRS} frequency scales. Note that all of the L2 cache access penalties are ten cycles except for f_8 and f_{ITRS}, for which we increased the access penalty to 15 cycles at 50nm and 35nm.

4.3 Performance Measurements

The performance scaling is measured using two major evaluation techniques. The first uses Instructions per Cycle (IPC) which is a measure of the architectural performance to evaluate the impact of scaling. The second uses instruction throughput to measure performance, which incorporates the impact of both architecture as well as clock rate.

Scaling	Clock Rate	250nm	180nm	130nm	100nm	70nm	50nm	35nm
Pipeline	f_{16}	2.12	1.77	1.77	1.77	1.77	1.71	1.71
	f_8	1.24	1.06	1.06	1.06	1.06	0.94	0.94
	f_{ITRS}	1.77	1.44	1.28	1.06	0.86	0.70	0.70
Capacity	f_{16}	2.11	2.11	2.07	2.07	2.07	2.04	2.04
	f_8	1.32	1.30	1.30	1.27	1.28	1.25	1.25
	f_{ITRS}	2.12	1.47	1.23	1.28	1.03	0.79	0.84

Table 4.4: Geometric mean of IPC for each technology across the SPEC CPU2000 benchmarks.

4.3.1 Architectural Performance

Table 4.4 shows the geometric mean of the measured IPC values across the subset of the SPEC CPU2000 benchmarks used for the experiments. The results include both pipeline scaling and capacity scaling experiments at each of the three clock scaling targets.

In general, the IPC decreases as the technology is scaled from 250nm to 35nm. For linear clock scaling (f_{16} and f_8), this effect is caused by longer structure access penalties due to sub-linear scaling of the wires. For f_{ITRS}, the superlinear reduction in cycle time causes a larger increase in access penalty than f_{16} or f_8, resulting in a sharper drop in IPC. Detailed IPC values can be found in Tables C.1 to C.4 in Appendix C.

There are several cases in which IPC increases when moving from a larger technology to a smaller one. These small increases are caused by discretization limitations in our model: if the structure capacity at a given technology becomes too small to be useful, we increase the access penalty by one cycle. This increase occasionally results in a larger structure than in the pre-

vious technology, causing a small increase in IPC. One example of this effect is evident for capacity scaling at f_{ITRS} clock rates when moving from 130nm to 100nm. In Table 4.3, the branch predictor, L1 I-Cache, and L1 D-Cache all become noticeably larger but one cycle slower. A similar effect occurs for f_{ITRS} capacity scaling at the transition from 70nm to 50nm.

With the slower clock rates (f_{16} and f_{ITRS}), the IPCs at 250nm for capacity scaling are considerably higher than those for pipeline scaling. While the capacity scaling methodology permits structure sizes larger than chips of that generation could contain, the pipeline scaling methodology sets the sizes to be roughly equivalent to the 21264. The capacity scaling results at 250nm and 180nm thus show the IPCs if the chip was not limited by area. As the wires grow slower in the smaller technologies, and the core becomes communication bound rather than capacity bound, the capacity scaling strategy loses its advantage. The pipeline scaling shows higher IPC than capacity scaling for fast clocks at the smallest technologies. The highest IPC at 35nm, unsurprisingly, is for capacity scaling at the slowest clock available–that point is the one at 35nm for which the microarchitectural structures are the largest. Even with capacity scaling at the f_{16} clock scale, however, IPC decreases by 20% from 250nm to 35nm.

4.3.2 Instruction Throughput

For either scaling strategy, of course, IPC decreases as clock rates are increased for smaller technologies. However, performance estimates must in-

Scaling	Clock Rate	250nm	180nm	130nm	100nm	70nm	50nm	35nm	Speedup
Pipeline	f_{16}	1.46	1.72	2.38	3.09	4.41	5.96	8.52	5.80
	f_8	1.68	2.05	2.84	3.70	5.29	6.55	9.37	5.57
	f_{ITRS}	1.33	1.80	2.69	3.73	5.21	7.06	9.53	7.14
Capacity	f_{16}	1.45	2.05	2.77	3.60	5.13	7.09	10.14	6.95
	f_8	1.78	2.51	3.48	4.42	6.38	8.69	12.42	6.95
	f_{ITRS}	1.59	1.84	2.60	4.50	6.22	7.90	11.41	7.16

Table 4.5: Geometric mean of performance (billions of instructions per second) for each technology across the SPEC CPU2000 benchmarks.

clude the clock as well. In Table 4.5, we show the geometric mean performance of the SPEC CPU2000 benchmarks, measured in billions of instructions per second (BIPS), for each of the microarchitectural and clock scales. Most striking about the results is the similarity in performance improvements across the board, especially given the different clock rates and scaling methodologies. The small magnitudes of the speedups, ranging from five to seven for 35nm is remarkable (for all our normalized numbers, the base case is pipeline scaling performance at f_{16} and 250nm).

For all technologies through 50nm and for both scaling strategies, faster clocks result in uniformly greater performance. For capacity scaling at 35nm, however, the faster clocks show worse performance: f_{16} has the highest BIPS, f_8 is second, and f_{ITRS} has the lowest performance. This inversion is caused mainly by the memory system, since the 35nm cache hierarchy has considerably less capacity for the faster clocks.

For pipeline scaling, the caches all remain approximately the same size regardless of clock scaling, and the benefit of faster clocks overcome the set-

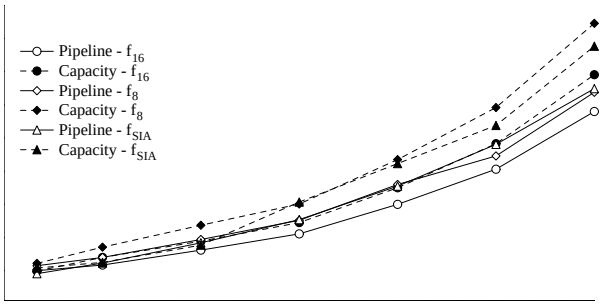


Figure 4.1: Performance increases for different scaling strategies.

backs of higher access penalties. Thus at 35nm, the best-performing clock scale is the most aggressive scale (f_{ITRS}).

4.4 Discussion

Figure 4.1 graphs total performance scaling with advancing technology. The six lines represent pipeline or capacity scaling for each of the three clock scaling rates. All performance numbers represent the geometric mean of the SPEC CPU2000 benchmarks, and are normalized to our baseline.

Although the various scaling strategies perform differently for the intermediate technologies, the overall performance at 35nm is remarkably consistent across all experiments. Capacity scaling at f_{ITRS} shows a high variance; this effect is due to the oscillating size of the L1 instruction cache, caused by the discretization issue described earlier. The pipeline scaling at f_{16} is the only strategy that performs qualitatively worse than the others. The f_{ITRS} and f_{16} clocks differ by nearly a factor of three at 35nm, and yet the overall perfor-

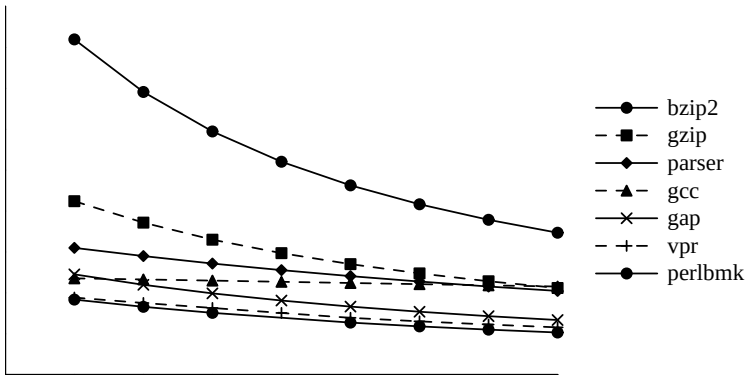
mance for both clocks at both scaling methodologies is nearly identical. The maximal performance increase is a factor of 8.6, which corresponds to a 13.5% annual improvement over that 17-year span.

4.4.1 Processor scaling

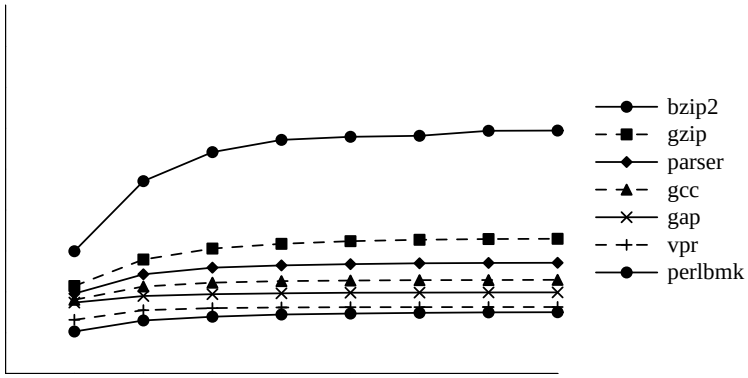
While careful selection of the clock rate, specific structure size, and access penalty would result in small performance improvements above what we have shown here, the consistency of these results indicates that they would not be qualitatively superior. For a qualitative improvement in performance growth, microarchitectures significantly different than those we measured will be needed. Wider issue processors with increasing numbers of execution units require low latency and high bandwidth cache access. However, adding ports to a monolithic cache increases both latency and area. In addition, increases in on-chip wire latency encourages partitioning so that the size and delay of critical structures, such as register files, are limited. While the Alpha 21264 was an early example of a commercial clustered microprocessor [32], an enormous amount of current research is addressing clustered architectures.

4.4.2 Clustered Caches

While most of the work on clustering has focused on the processor core, relatively little has addressed the effect on the memory system. Wider issue processors seek to increase issue-rate of all instruction, including loads and stores. Unfortunately, just adding load/store execution units merely



(a)



(b)

Figure 4.2: IPC for SPEC2000 integer benchmarks as a function of (a) cache latency (b) number of cache ports.

increases bandwidth and latency pressures on traditional monolithic level-1 caches. Since clusters must be spread spatially across some fraction of the chip, the distance between a cluster and a centralized level-1 cache increases, thus increasing cache access latency. Figure 4.2a shows the affect of increasing level-1 cache latency in a 16-wide monolithic processor across a spectrum of the SPEC2000 benchmark suite. Reducing the level-1 cache latency from 3 cycles to 1 cycle increases average instructions per clock (IPC) by an average to 19%. Similar improvements result when the latency is reduced from 5 cycles to 3 cycles. Adding load/store execution units without increasing the level-1 cache bandwidth will elevate average memory access time because of level-1 cache contention. Figure 4.2b shows that increasing the number of level-1 cache ports from 1 to 3 results in an average IPC boost of 39%.

While a monolithic cache forces the design to trade off bandwidth for latency, a partitioned design allows for high bandwidth and higher capacity with low access latency. In the rest of this research, we propose and evaluate clustered cache architectures connected to a clustered processor architecture. The next chapter details the architecture used to evaluate the steering algorithms and clustered caches that are proposed in this research. The architecture is then analyzed using dependence and round-robin steering to determine the optimal cache parameters to use with the various steering algorithms.

Chapter 5

Clustered Caches

The results from the previous chapter show that scaling a monolithic design to smaller feature sizes will only give us an average of 12.5% improvement per year. The results also show that reducing the latency and improving the bandwidth of the primary data cache lead to significant improvements in processor performance. In this chapter we describe the microarchitecture of a clustered primary data cache connected to a set of clustered execution units. Partitioning the cache into independently accessible banks and placing the banks close to the execution units is one way to reduce the average access latency as well as allowing multiple independent accesses to occur simultaneous, thus improving the bandwidth. The objective is to maximize the likelihood that a given load or store instruction finds the data that it needs in the bank closest to it. There are two aspects to matching load and store instructions to a cache bank: steering the instruction to the appropriate cluster and the physical organization of the cache. This chapter addresses the issues involved in the physical organization while the next chapter deals with the instruction steering issues.

5.1 Microarchitecture

The basic architecture examined in this research is shown in Figure 5.1. The processor has a unified front end with the instruction fetch, decode and map stages shared by all clusters. The map stage performs register renaming and steers the instructions to one of the four clusters according to the steering policy. Once instructions are mapped onto the cluster, they enter the instruction window of the cluster, and issue out-of-order as the operands are ready. Since dependent instructions potentially can be mapped on different clusters, transfer instructions are used to communicate values between clusters. The cluster that needs the data, sends an instruction requesting the data to the cluster with the data. The transfer instruction then executes on the remote cluster and forwards data back to the cluster that requested it. Transfer instructions consume execution resources and have finite latency to communicate the value from one cluster to another just as in the MultiCluster architecture [19].

The interconnect mechanism between the clusters and the primary data cache banks can either be a full crossbar allowing any cluster to access any cache bank or it can be a point-to-point connection only allowing access to the local cache bank. With the point-to-point interconnect, data in remote banks is only accessible by migrating it to the local cache bank. Data is mapped into the primary data cache banks using a variety of different address mapping and replacement policies. Data can either be statically mapped according to the address to a fixed location in a specific bank or it can be dynamically mapped

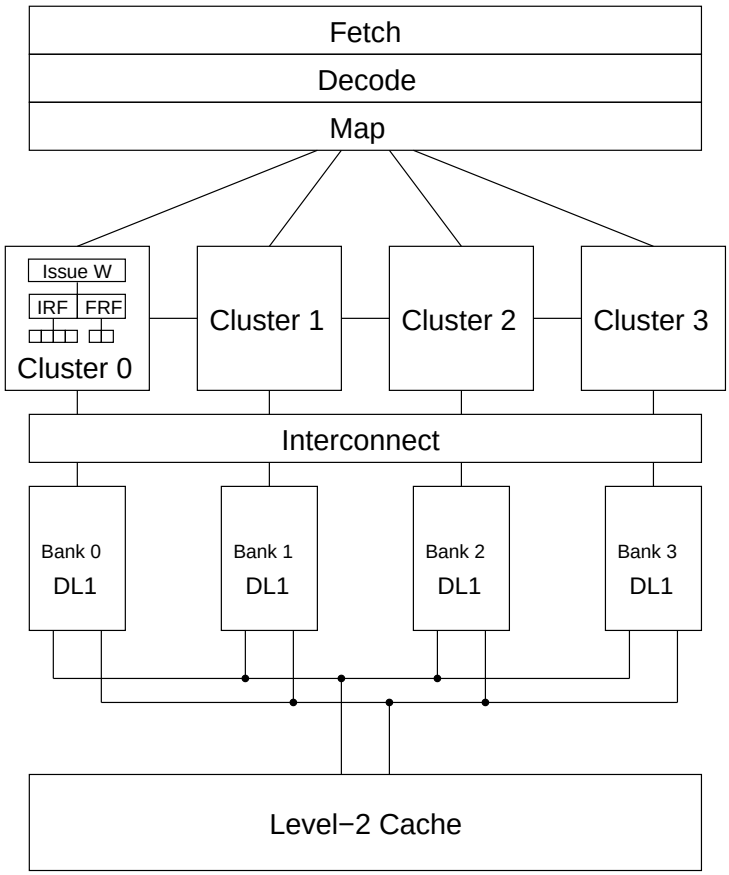


Figure 5.1: The baseline hardware that is considered for this research.

to any of the cache banks. The interconnect between the primary data cache and the unified L2 cache transfers data and maintains coherence.

This research investigates the performance of the processor while focusing on the primary data cache. Only load and store instructions access the data cache and of these, only loads are timing critical. Load instructions may encounter three scenarios:

- Local data, local consumer: The load, the instruction dependent on the load, and the data accessed by the load are all mapped to the same cluster. This is the optimal case, because it requires the least amount of communication and shortest interconnection.

- Local data, remote consumer: The load and the data required by the load are mapped on cluster i while the dependent instruction is mapped on cluster j. This requires the data to be loaded into cluster i from bank i and then transfered to the dependent instruction in cluster j. This transfer of the value from cluster i to cluster j adds latency as well a resource overhead to the execution of the instructions.

- Remote data: The load is mapped on cluster i while the data required by the by the load is in bank j. Regardless of where the dependent consumer instruction is mapped, this case has the highest overhead because cluster i must send the request to bank j and then wait until the data returns. Because this involves at least two inter-cluster transfers, this scenario has the longest latency and highest resource overhead. The first inter-cluster transfer is the remote request for the data to the cache bank that has the data. The second is the transfer of the data from the remote cache bank to the cluster requesting the data. Additionally, if the consumer of the data is located in a different cluster, additional inter-cluster communication is required to forward to the data to the consumer.

The rest of this chapter investigates different options for data placement, and topology that can reduce per access latency and increase the number of instructions in the first category. In this chapter we consider round robin and dependence based steering algorithms to map instructions to the respective clusters. The round robin algorithm steers groups of three instructions to each cluster in a round robin manner. On the other hand dependence based steering assigns dependent instructions to the same cluster. Further steering algorithms are described and evaluated in Chapter 6.

5.2 Simulation Environment

We use a modified version of the sim-alpha [15] simulator to run our performance modeling experiments, which is a validated simulator for the Alpha 21264 microprocessor. The simulator has been modified to support the following features:

- Processor clustering where resources are split into clusters and inter-cluster transfer instructions can be used to communicate values.
- Instruction steering algorithms including round-robin and dependence based steering.
- Multiple cache configuration including unified, clustered, and supporting several static and dynamic address mapping schemes.

The data cache is subdivided into a number of equal banks, based on the number of execution clusters in the processor. Each execution cluster can

Cache Capacity	Latency
32K	3
64K	4
128K	6
256K	10
512K	14

Table 5.1: Cache bank latency for the various cache configurations.

access the local cache bank directly or a remote cache bank through an interconnect. The memory subsystem uses a write through, write allocate, snooping policy to keep the data coherent across the various L1 cache banks. A cache line can either be present in only one cache bank at a time (no-replication), or it can be potentially present in multiple cache banks simultaneously (replication). On a L1 cache miss the L2 is probed and if the data is present in the L2, then the data is loaded from the L2. If the data is not present in the L2, then the main memory is accessed to load the data.

We use a technology scaled version of the ECACTI [55] simulator to estimate the cycle time of the various cache configurations used in this research at the 50nm technology node with an 8 fanout-of-four (FO4) clock (13GHz at 50nm). All the caches have 2 read/write ports and are fully pipelined to allow a new access to start every cycle. Table 5.1 lists the varying data cache capacities with their corresponding latencies. As can be seen from Table 5.1, the access latency increases significantly as the cache capacity increases beyond 128KB. Even the jump from 32K to 64K adds a single critical cycle to the primary cache access times. As shown in Chapter 4, these latencies will be more severe

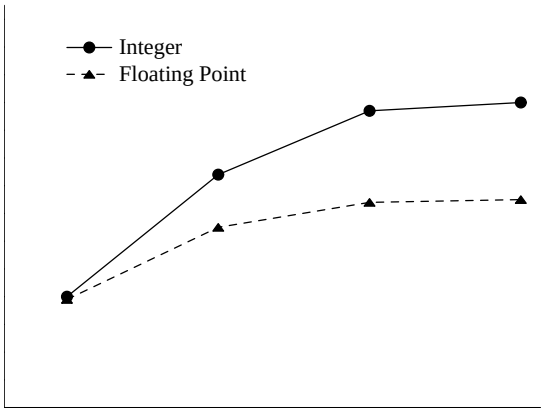


Figure 5.2: Mean IPC values as a function of the issue width.

at smaller feature sizes.

The SPEC CPU2000 [25] benchmarks with the `ref` input set represent the workload in this research. Table B.1 lists the integer benchmarks and Table B.2 lists the floating point benchmarks along with a brief descriptions of each benchmark. The benchmarks are compiled for Alpha EV6 ISA. The SimPoint toolkit [53] is used to generate multiple weighted regions of 100 million instructions each. The regions generated are representative of overall program behavior and thus allow simulation of overall program behavior in a much shorter execution time.

5.3 Baseline Processor Choices

Figure 5.2 shows the average IPC for integer and floating point benchmarks as a function of the issue width. The detailed IPC values for the in-

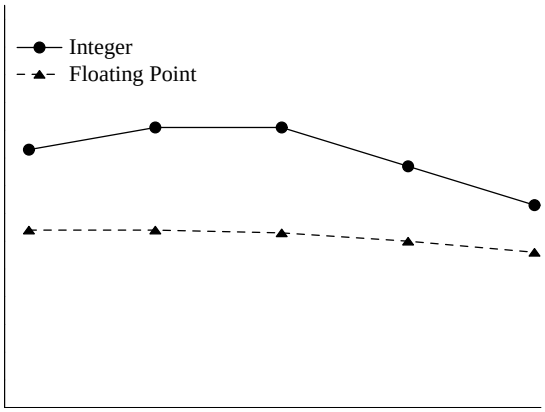


Figure 5.3: Mean IPC values for unified DL1 from 32KB to 512KB in capacity.

dividual benchmarks can be found in Tables D.1 and D.2. The issue width and functional resources are equally divided across four clusters for these experiments. The primary data cache consists of four statically mapped cache banks of 128KB each, backed up with a 2MB, 4-way set associative L2. The instructions are steered using dependence based steering. There is significant performance improvement from four wide to thirty-two wide. However, since there is negligible improvement in IPC from sixteen wide to thirty two wide, the sixteen wide configuration is used for the processor that is studied in this chapter.

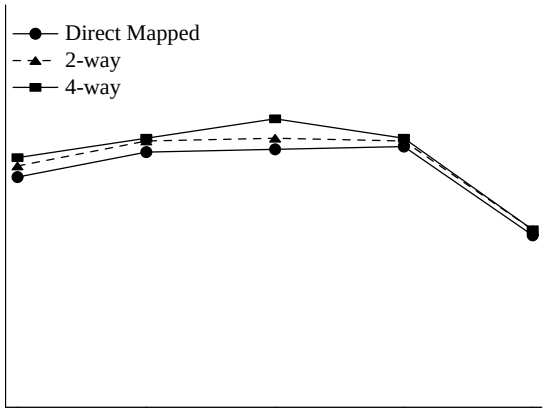
Figure 5.3 shows the mean IPC for integer and floating point capacity as a function of cache capacity for a unified data level-1 (DL1) cache. The detailed IPC values for the individual benchmarks can be found in Tables D.3 and D.4. The DL1 is 2 way set associative with 2 read/write ports. The ports

are shared across all 4 clusters. Based on the experiments on the issue width, these results are for a sixteen wide issue processor. Performance improves initially because the increased capacity leads to an improvement in the hit rate that offsets the increase in DL1 latency. However at capacities over 128KB, the improvement in hit rate cannot offset the increased latency and that leads to a decrease in the average IPC. This represents the baseline relative to which all performance is measured.

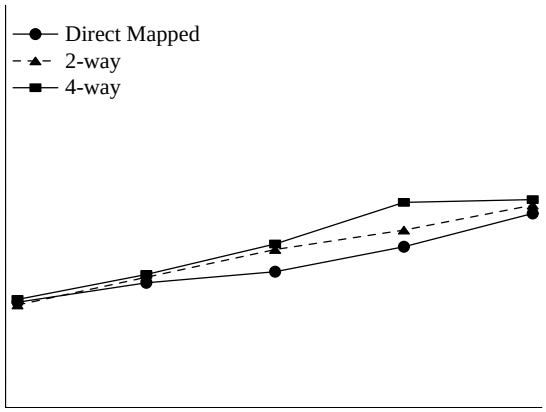
Figure 5.4 shows the average IPC as a function of L2 cache capacity corresponding to the individual benchmark IPC values listed in Tables D.5 and D.6. For the floating point benchmarks, performance steadily improves as a function of the cache capacity, because of a constantly improving hit rate. For the integer benchmarks, performance improves while the improvement in hit rate offsets the increase in L2 cache latency. However at capacities larger than 2MB, the performance decreases because the increased hit rate cannot offset the increase in latency. Based on these results, the 2MB 4-way set associative cache provides the best overall performance and is used as the L2 cache configuration for the experiments in this chapter as well as Chapter 6.

Perfect branch prediction is used to maximize the number of load and store instructions executed every cycle and to stress the memory sub-system. Table 5.2 shows the baseline machine configuration on a per cluster basis.

First we study the impact of bank capacity on the performance of the design. Next, we consider the impact of the topology on the performance of the processor. Finally we examine address mapping, and show that while



(a)



(b)

Figure 5.4: Average IPC for varying L2 configurations for (a) integer and (b) floating point benchmarks.

Instruction Fetch and Map
Perfect or hybrid branch prediction Fetch, decode and map 16 instructions per cycle
Execution Resources per Cluster
128 entry integer and floating-point register file 4 integer and 4 floating point units
Caches
Varying data cache configurations 1 cycle, 64KB direct mapped instruction cache 2MB 4-way, 2 port unified L2, 43 cycle latency deeply pipelined to allow one new access per cycle

Table 5.2: Processor Configuration

static mapping can yield performance comparable to a unified cache, using a dynamic mapping scheme can supply significant performance improvement.

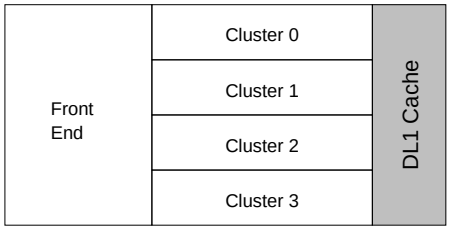
5.4 Statically Interleaved Caches

In a statically interleaved (mapped) cache, each address maps to a fixed location in a pre-determined bank, depending on the interleaving function. Cache-line interleaving in which an entire cache line is mapped to the same bank is convenient since a refill only requires communication between the level-2 cache and the level-1 cache bank that caused the miss. While this configuration does not allow multiple simultaneous accesses to the same cache line, it can improve bank locality. A particular instance of a load instruction in a tight loop would fetch multiple words from the same bank before proceeding to the next bank. Word-level interleaving could reduce contention in

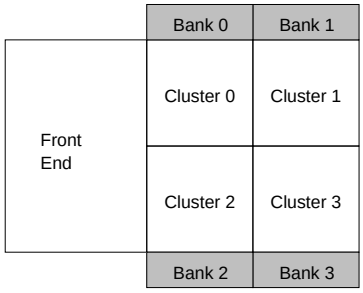
some circumstances, but would incur additional overhead because an eviction would require each banks to evict its part of the cache line. We found that the performance difference between word and cache-line interleaving was negligible and thus advocate cache-line interleaving for its simpler implementation.

5.5 Topology Variation

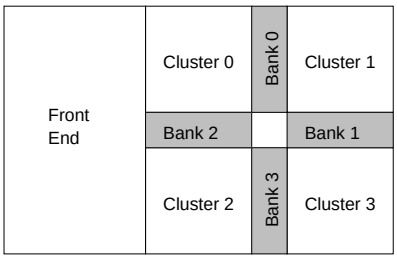
Since communication between two clusters, as well as between a cluster and a cache bank, is important, the physical layout of the clusters relative to each other as well as relative to the primary cache has an impact on overall performance. The traditional way to design partitioned processors has been to use a monolithic cache that is shared by all the clusters in the processor as shown in Figure 5.5a. This floorplan provides a constant but larger access latency to the cache from all the clusters. However, for banked caches in which each cluster has local access to one bank of the primary cache there are several possible floor plans. Figure 5.5b shows a "cluster centric" arrangement minimizing inter-cluster communication, and the latency of transfer instructions. The trade-off is increased memory latency as the cache banks are further from some functional units than from others. A "cache centric" organization, as shown in Figure 5.5c, minimizes the memory latency at the cost of increasing inter-cluster latency. The cache-centric topology works better for benchmarks that have a lot a remote memory instructions relative to the amount of inter-cluster communication. On the other hand, the cluster-centric topology works better for benchmarks that have more inter-cluster communication relative to



(a) Unified Cache



(b) Cluster centric



(c) Cache centric

Figure 5.5: Different physical topologies

Topology	Source Cluster	Destination Cluster			
		0	1	2	3
Unified cache	0	0	3	3	4
	1	3	0	4	3
	2	3	4	0	3
	3	4	3	3	0
Cluster centric	0	0	1	1	3
	1	1	0	3	1
	2	1	3	0	1
	3	3	1	1	0
Cache centric	0	0	3	3	4
	1	3	0	4	3
	2	3	4	0	3
	3	4	3	3	0

Table 5.3: Inter-cluster latencies in cycles for various topologies.

the number of remote memory operations.

We use an area model [22] and a wire delay model [2] to determine inter-cluster and cluster-cache latencies for different floor plans. The area model provides an estimated cluster area of $10^{11}\lambda^2$ ($14mm^2$ in a 50nm technology). The inter-cluster delay ranges from zero to four cycles, assuming an 8 fanout-of-four (FO4) clock (13GHz at 50nm). Table 5.3 shows the inter-cluster latencies for each of the topologies.

Figures 5.6 and 5.7 show the average IPC for the three topologies considered in this research when using a static address mapping scheme. The "ideal unified" line represents an unrealistic 512KB 8-port 1 cycle unified DL1. Since static mapping has cache hit characteristics similar to a monolithic cache the performance for the two cases is similar. Since some loads and stores to

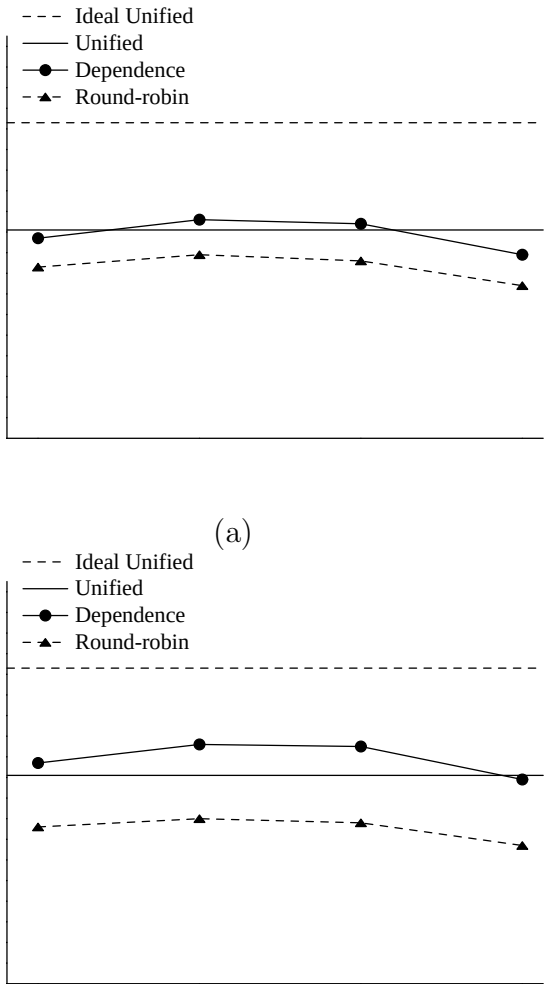
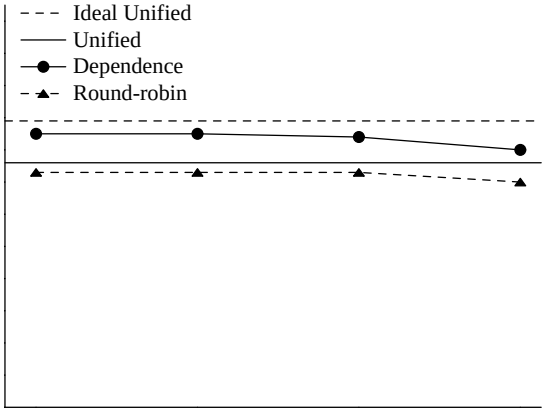
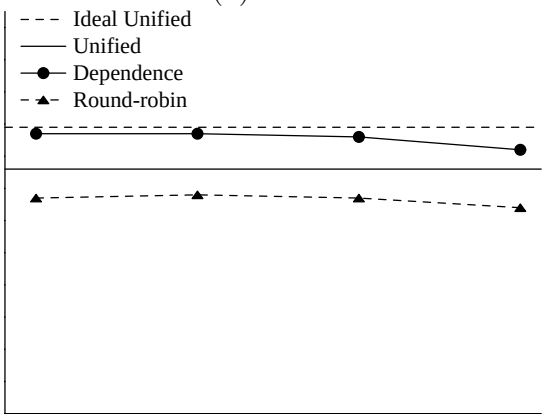


Figure 5.6: IPC values for integer benchmarks with statically mapped data for (a) cluster centric and (b) cache centric topologies.



(a)



(b)

Figure 5.7: IPC values for floating point benchmarks with statically mapped data for (a) cluster centric and (b) cache centric topologies.

remote banks can have a higher access latency for the partitioned design as compared to the monolithic design, the static partitioning has a lower average IPC. Integer benchmarks are more sensitive to cache capacity than the floating point benchmarks. As the capacity of a cache bank increases, performance initially improves as the hit rate gets better. However at cache capacities greater than 128KB the improvement in hit rate is offset by the increase in access latency. Finally, little differentiates the performance of the cluster centric topology from the performance of the cache centric topology. The cluster centric topology performs a little better because of lower transfer instruction latencies. However, as can be seen from Figure 5.6 there is a significant gap between the clustered caches and the ideal unified for the integer benchmarks. The reason for this gap is that with static mapping 75% of all memory instructions have to access a remote cache bank. Since both cluster-centric and cache-centric topologies have similar remote access latencies, the performance for both topologies is similar. The best way to reduce the number of remote loads is to map the data to the cache bank closest to the memory operation requesting the data as discussed in the next section.

5.6 Dynamically Mapped Caches

Dynamically mapped caches allow data to be placed in locations that vary during program execution. While the physical layout is the same as a static design, the placement of the data when it is first loaded, the location to look in when executing a load, whether to migrate data from one bank to

another, and how to maintain coherence across the banks are important design decisions.

5.6.1 Cache Parameters

Placement: When loading data into a primary cache that employs dynamic mapping, the bank into which the data is loaded is determined by the data placement strategy. The data may be loaded into the bank of the cluster that requested the value, or it can be loaded into a statically determined location. The latter has the advantage that data is always loaded into a well defined location making predicting the cluster to steer instructions to easier. However, if the consumer of the data is placed in a different cluster from the load, additional time to transfer the data across the clusters may be required. Loading the data into the cluster that requested it has the advantage of placing data close to where it is initially needed, but makes steering instructions to the cluster with the data more difficult.

Locating data: With static placement of the data, the bank to access on a load is obvious. However, with dynamic placement of data, since the location is not obvious, the local bank is accessed. In case of a miss in the local bank, the L2 is accessed as opposed to looking in the other DL1 banks for the data.

Data migration and Replication: In addition to the initial data placement policy, data can be migrated or replicated among the banks if needed

by more than one cluster. Migrating data means that the line is moved to the bank making the request and is removed from the bank that previously contained the data. Because the cache line can only be in one bank at a time, coherence is simple if the caches are write through. Replication implies that a cache line can be present in more than one DL1 cache bank at a time. With replication, on every store to a cache bank, the other cache banks must either invalidate their copies of the cache line that is being written or they must update their copy with the data being written. This research assumes that the cache lines must be invalidated and that all dependent memory instructions must stall till the invalidation is complete.

Coherence: Policies that allow data to be replicated must keep the data coherent across the cache banks. Because this is a primary data cache, the protocol to maintain coherence must be faster and cause less overhead than the protocols used to maintain L2 coherence. A long latency coherence protocol would cause all dependent memory instructions to stall reducing the effectiveness of dynamically mapping the data by increasing the average memory access time to levels comparable to that of the statically mapped case.

These four design aspects for a dynamically mapped cache affect one another. The coherence protocol depends strongly on the data migration and replication policy. Similarly, deciding where to look for data for a load is coupled to where data is placed when initially loaded. We consider all four aspects when designing the cache banks for a dynamically mapped primary

data cache.

5.6.2 Design of the Dynamically Mapped Cache

For our experiments, the cache line is loaded into the cache bank corresponding to the cluster that executed the load or store instruction that first accessed the data in order to exploit the spatial locality exhibited by groups of dependent instructions. Also, when a load or store executes, it always access the cache corresponding to the cluster on which it executes to reduce the cache access latency, because accessing a remote bank would be a longer latency operation. If the data is not found in the local cache bank, the L2 cache is accessed and so on, up the memory hierarchy. The only mechanism by which a cache line can move from one bank to another is if it is accessed by instructions executing on two different clusters. This architecture maintains coherence by employing a snooping, write through, write allocate policy for the primary data cache. All stores write through to the L2 cache and invalidate the corresponding line in the other cache banks. The invalidation latency is 10 cycles for the cluster-centric topology and 5 cycles for the cache-centric topology. When an invalidation occurs, all dependent memory instructions are held back for the appropriate number of cycles. An unmodified cache line can be present in multiple cache banks, while a modified cache line can only be in one cache bank. The number of write-through operations that can happen in a single cycle is limited by the two L2 write ports.

To focus on the interaction between steering and data mapping poli-

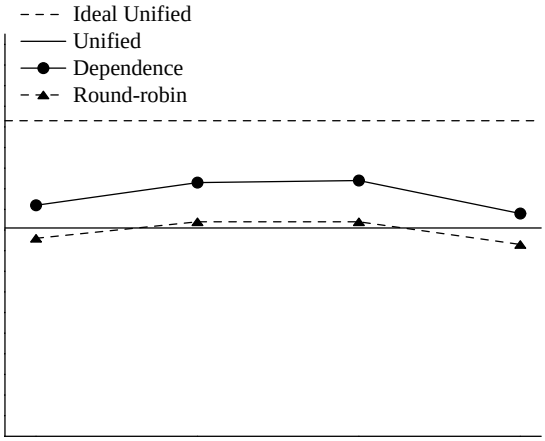
cies, we assume a centralized Load Store Queue (LSQ) that is off the critical path. A form of this could be implemented by speculating that no conflict occurs between a load and prior stores, and rolling back if one was later discovered during lookup in the LSQ. This approach is similar to processors such as the Pentium 4 that speculatively deliver data back to the processor from a virtually addressed cache and roll back only if a conflict is detected after translation. Such an approach is only feasible for the LSQ if load/store conflicts are uncommon. However, efforts to reduce the overhead of recovery would make this approach more efficient [16]. Another possible strategy partitions the load/store queue [52, 64], which is most amenable to a statically mapped and partitioned cache. These schemes could potentially be extended to dynamically mapped caches by performing two levels of disambiguation. The first level would be performed at the cache bank and would handle the common case of instructions accessing the same data. Conflicts detected at the second level LSQ would require a rollback recovery mechanism. Memory disambiguation is an active area of research [43, 52], and we expect that innovations in that domain would benefit the clustered architectures described in this research.

5.7 Performance of Dynamically Mapped Caches

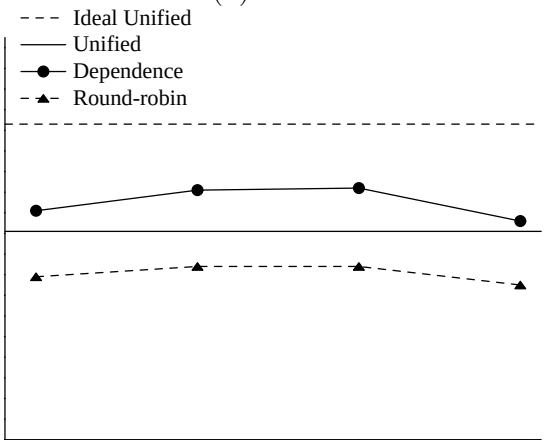
Figures 5.8 and 5.9 show the mean Instructions per Cycle (IPC) of the benchmarks for dynamic address mapping with migration (no replication) as a function of the DL1 bank size. A higher IPC implies a cache capacity that is

better suited to the data mapping policy. With dynamic mapping, the cache can store data closer to the cluster using it than with a monolithic centralized cache. Also, because lines that would conflict in a monolithic cache can map into different cache banks, the partitioned cache has a higher implicit associativity than the monolithic cache. The reason that the floating point IPC drops as a function of cache capacity is that the miss rate stays essentially constant with increasing capacity while the latency increases significantly. Figure 5.10 shows the mean miss rate for integer and floating point benchmarks as a function of DL1 bank size. Since the miss rate stay essentially constant for the floating point benchmarks, the mean IPC drops as the latency of the DL1 cache banks increases from 3 cycles for the 32KB banks to 14 cycles for the 256KB banks. The optimal dynamically mapped cache with a capacity of 128KB per bank outperforms the corresponding statically mapped design by 12%.

Figures 5.11 and 5.12 show the mean IPC for the benchmarks for dynamic address mapping with replication. In this scenario, on a cache write, the cache line must be invalidated in any cache bank that has this line. This invalidation latency causes all dependent memory instructions to be held back until the invalidation has been completed, leading to a reduction in the average IPC. Since the invalidation latency is significantly lower, the impact on performance is less for cache centric topology as can be seen from Figure 5.11b. Furthermore, dependence based steering of the instructions outperforms round robin steering because of lower inter cluster communication in the case of the



(a)



(b)

Figure 5.8: IPC values for integer benchmarks with dynamically mapped data without replication for (a) cluster centric and (b) cache centric topologies.

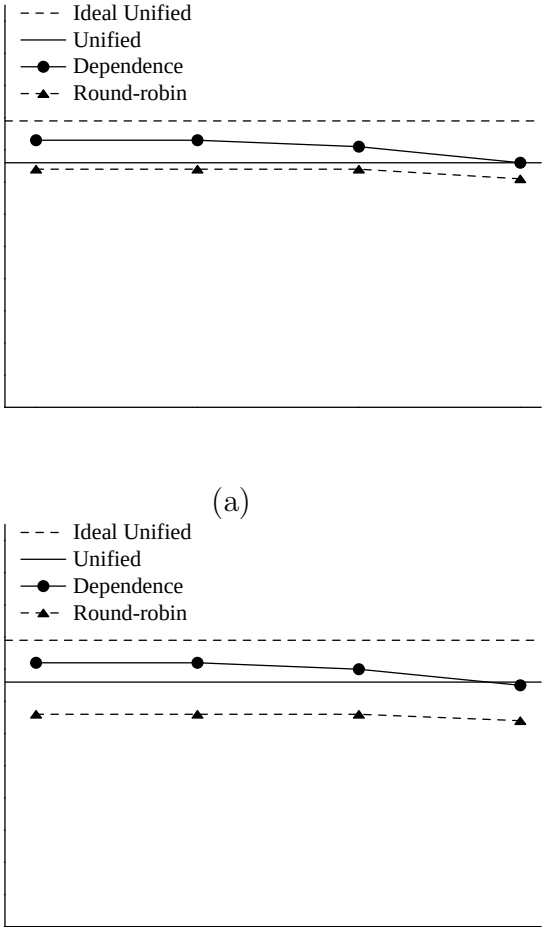


Figure 5.9: IPC values for floating point benchmarks with dynamically mapped data without replication for (a) cluster centric and (b) cache centric topologies.

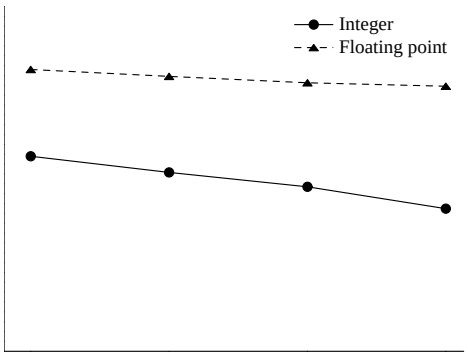


Figure 5.10: Mean miss rate as a function of DL1 bank size for the cluster centric topology.

former relative to the later. This difference is independent of the replication policy. Lastly, just like with the static data mapping as the cache capacity increases, performance initially improves as the improvement is cache hit rate outstrips the impact of the increased latency. However at larger capacities, the higher latency hurts performance more that the benefit gained from the improved hit rate. Similar to the case without replication, the mean IPC of the floating point benchmarks drops with an increase in cache capacity because the mean miss rate stays essentially constant.

5.8 Discussion

In this chapter, we examined different clustering topologies for assigning cache lines to cache banks. Our results show that with a 4-cluster, 16-wide processor and a projected 50nm technology, topology has little affect on overall

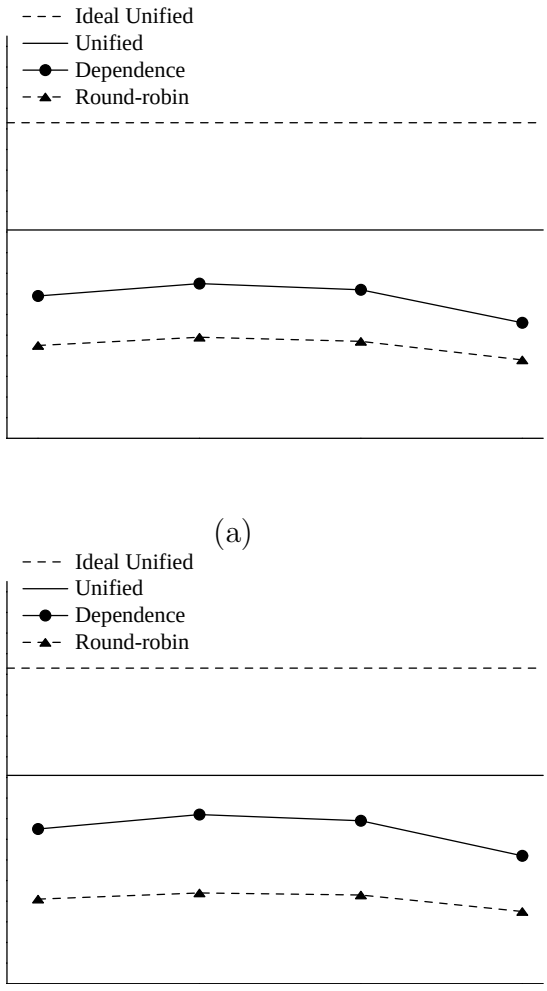
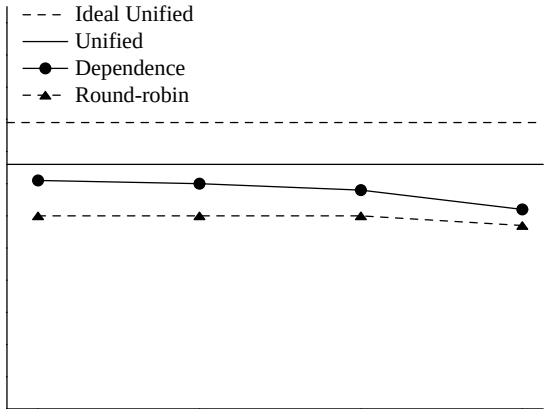
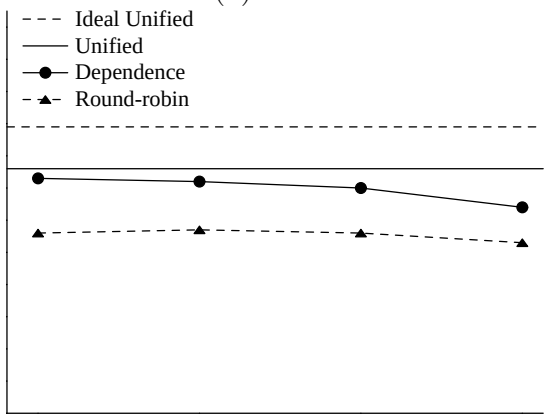


Figure 5.11: IPC values for integer benchmarks with dynamically mapped data with replication for (a) cluster centric and (b) cache centric topologies.



(a)



(b)

Figure 5.12: IPC values for floating point benchmarks with dynamically mapped data with replication for (a) cluster centric and (b) cache centric topologies.

performance, as long as both the processor and cache are clustered. Because of this little variation in performance between cluster centric and memory centric topologies, we only discuss the results from cluster centric topologies in the next chapter. Also, dynamically mapping the data to the cache banks leads to a 12% performance improvements relative to the statically mapped case. Allowing a cache line to be present in more than one bank tends to hurt performance because in the case of an invalidation, dependent load and store instructions must be restrained till the invalidation completes. Lastly, dependence based steering outperforms round robin steering by 12% because the former minimizes expensive inter-cluster communication.

So far we have only examined the performance from the cache organization point of view. From the results in this chapter we have noticed that the instruction steering policy does have a significant impact on performance. However, the ideal unified cache outperforms the best clustered design presented in this chapter by 16%. The next chapter examines how a more aggressive steering algorithm that maps the memory instruction to the appropriate cluster coupled with dynamic data mapping can close the gap between the performance of a dynamically mapped clustered cache with dependence steering and the ideal unified cache.

Chapter 6

Steering Policies for Clustered Cache Architectures

The results from the previous chapter show that there is a 16% gap between the performance of the clustered cache with dependence steering and the ideal unified cache. In addition to data placement policies discussed in the previous chapter, the steering algorithm used in the map stage to assign instructions to the clusters also affects how many load instructions find their data in the nearest cache bank. The objectives of a good steering algorithm are to minimize inter cluster communication, balance the execution load across all the clusters, and reduce the number of non-local loads executed on any given cluster.

The basic steering algorithms for mapping instructions on to the clusters that have been typically used are:

- Round robin: Steers groups of three instructions to each cluster in a round robin manner [6]. The round robin method has the advantage of balancing the load between the various clusters, but has the problem of frequently mapping dependent instructions to different clusters, thus increasing inter-cluster communication.

```
 1  LD    R2, R1
 2  LD    R4, R3
 3  ADD   R5, R1, R4
 4  MUL   R6, R2, R4
 5  SHFTR R6, 5
 6  LD    R9, R8
 7  SUB   R7, R5, R4
 8  LD    R2, R7
 9  SHFTL R9, 2
10  ADD   R8, R8, 4
11  ST    R9, R8
12  ADD   R9, R4, R1
```

Figure 6.1: Code fragment to show how different steering algorithms work.

Steering	Cluster 0	Cluster 1	Cluster 2	Cluster 3

Round Robin	`LD R2, R1`
	`LD R4, R3`
	`ADD R5, R1, R4`
		`MUL R6, R2, R4`
		`SHFTR R6, 5`
		`LD R9, R8`
			`SUB R7, R5, R4`
			`LD R2, R7`
			`SHFTL R9, 2`
				`ADD R8, R8, 4`
				`ST R9, R8`
				`ADD R9, R4, R1`

Dependence	`LD R4, R3`
	`ADD R9, R4, R1`
	–
	–
	–
	–
		`LD R2, R1`
		`ADD R5, R1, R4`
		`MUL R6, R2, R4`
		`SHFTR R6, 5`
		`SUB R7, R5, R4`
		`LD R2, R7`
			`LD R9, R8`
			`SHFTL R9, 2`
			`ADD R8, R8, 4`
			`ST R9, R8`
			–
			–
				–
				–
				–
				–
				–
				–

Figure 6.2: Example of instruction mapping for dependence and round robin steering.

- Dependence: Dependence steering reduces inter-cluster communication by assigning dependent instructions to the same cluster [42]. This leads to fewer transfer instructions, but more instructions are mapped to a single cluster.

For the sequence of instructions in Figure 6.1, the round robin algorithm would map instructions 1-3 to Cluster 0, instruction 4-6 to Cluster 1,

instructions 7-9 to Cluster 2 and instructions 10-12 to Cluster 3 as shown in Figure 6.2. For the same sequence of instructions, dependence based steering would map instructions 1,3,4,5,7 and 8 to the Cluster producing R1, instructions 2 and 12 to the Cluster producing R3 and instructions 6,9,10 and 11 to the Cluster producing R8 as shown in Figure 6.2. Dependence based steering requires only 1 transfer instruction, while round robin steering requires a total of 6 transfer instructions. Assuming that all operations require 1 cycle, the dependence based steering would execute in 5 cycles while the round robin steering example would take 7 cycles to execute.

Neither the dependence nor the round robin steering policy try to match the memory instructions with the cache bank that has the data needed by the instruction. An aggressive steering algorithm that matches memory instructions with the cache bank containing the data can significantly improve the performance of a dynamically mapped clustered cache.

6.1 Bank Predictive Steering

With dependence and round-robin steering a large number of memory instructions access addresses that are located in a remote cache bank. Accessing data not in the local cache bank is either a long latency operation (static address mapping) or achieved through accessing the L2 cache (dynamic address mapping). One possible approach to reduce the memory access latency is to map memory instructions to clusters that contain the data required by the instructions. We propose a steering algorithm that steers load and store

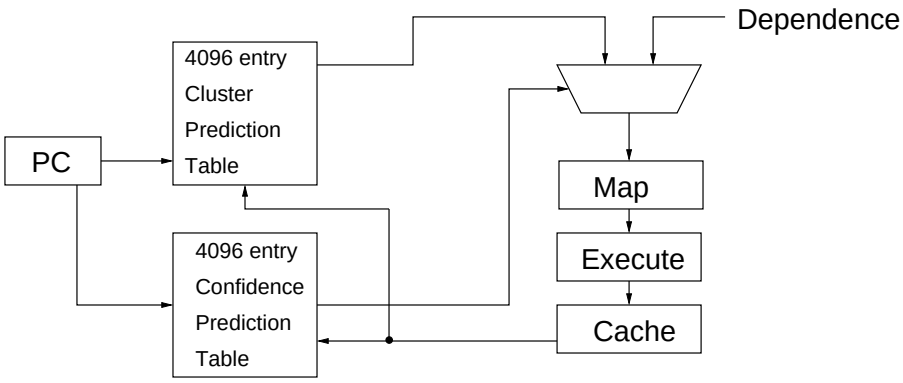


Figure 6.3: Microarchitecture structures used for Memory Predicted steering

instructions by predicting the cluster that contains the data required by the load or store. All other instructions are mapped using a dependence based steering policy.

Figure 6.3 shows a diagram of the predictor designed to steer load and store instructions in the proposed bank prediction steering policy. For loads and stores, the program counter indexes a 4096 entry prediction table that produces cache bank predictions. The PC also indexes a 4096 entry table consisting of 2-bit saturating confidence predictors. A confidence predictor value of 0 or 1, indicates low confidence in the prediction and the instruction is steered based on its dependence. A confidence value of 2 or 3 is high confidence, indicating that the bank predictor result should be used.

When the memory instruction executes and encounters a cache miss, the confidence predictor is decremented. If the load or store was steered using dependence based steering and it hits in the cache, the predictor updates the

Cluster 0	Cluster 1	Cluster 2	Cluster 3
–	LD R9, R8	LD R4, R3	LD R2, R1
LD R2, R7	SHFTL R9, 2	ADD R5, R1, R4	MUL R6, R2, R4
ST R9, R8	ADD R8, R8, 4	SUB R7, R5, R4	SHFTR R6, 5
–	–	ADD R9, R4, R1	–

Figure 6.4: Example of instruction mapping for bank predictive steering.

bank prediction table to point to the cache bank in which the instruction hit and sets the confidence level for the prediction to 2 (medium-high confidence). If the load or store had been was steered using the bank prediction and it hits in the primary data cache, the predictor increments the corresponding confidence predictor.

As an example, for the code fragment in Figure 6.1, when instruction 1 is mapped, the prediction table is examined and if the confidence is high, instruction 1 is mapped to the predicted cluster, else it is mapped to the cluster that produces R1. Similarly, instructions 2,6,8 and 11 are either mapped to the cluster predicted to have the cache line required by the memory operation or the cluster that produces R3, R8, R7 or R9 respectively. The remaining instructions are mapped to the cluster that has the producer instruction of the operand required by the instruction. Figure 6.4 shows an example mapping for the case where registers R1, R3, R7 and R8 are produced by clusters 2, 2, 3 and 1 respectively and instructions 1, 2, 6, 8 and 11 are predicted to be mapped to clusters 3, 2, 1, 0 and 0 with high, high, low, high and high confidence respectively. The predictive steering example requires 3 transfer instructions and takes 6 cycles to execute assuming that every operation takes 1 cycle.

Legend	Description
Dependence	Dependence based steering
Predictive	Bank prediction for memory instructions
Hybrid	Hybrid of the dependence and predictive memory steering
Oracle	Memory steering with oracle load/store mapping

Table 6.1: Legend of steering algorithms used in the graphs.

Cluster 0	Cluster 1	Cluster 2	Cluster 3
`LD R9, R8`	`LD R2, R7`	`LD R4, R3`	`LD R2, R1`
`SHFTL R9, 2`	`ADD R8, R8, 4`	`ADD R5, R1, R4`	`MUL R6, R2, R4`
`ST R9, R8`	–	`SUB R7, R5, R4`	`SHFTR R6, 5`
–	–	`ADD R9, R4, R1`	–

Figure 6.5: Example of instruction mapping for oracle steering.

6.2 Experimental Parameters

Table 6.1 summarizes the steering algorithms used in this chapter. Based on the results of the previous chapter, the physical design of the cache consists of 128KB 2-way set associative cache banks with non-replicated mapping of data. The rest of the processor configuration and simulation environment is the same as described in Section 5.2.

To maximize the chance of finding the data in the local cache bank, the oracle steering algorithm maps memory instructions to the cluster that is known to contain the data. This represents an upper bound on how much performance can be gained by steering load/store instructions to the appropriate cluster. All other instructions are mapped on a dependence basis.

For the code fragment in Figure 6.1, the oracle load/store steering algorithm would map instructions 1,2,6,8 and 11 to the cluster whose local cache

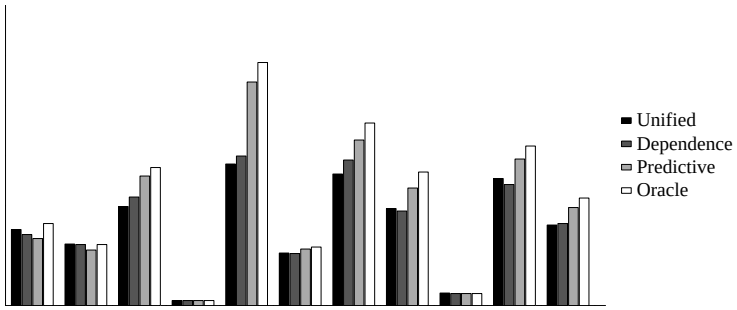


Figure 6.6: IPC values of integer benchmarks for statically mapped data and cluster centric topology.

has the cache line required by the memory operation. Of the remaining instructions, 3, 7 and 12 are mapped to the same cluster as instruction 2, instructions 4 and 5 are mapped to the same cluster as instruction 1, instruction 9 to the same cluster as instruction 6 and instruction 10 to the cluster that produces R8. Figure 6.5 shows an example mapping for the case where registers R1, R3, R7 and R8 are produced by clusters 2, 2, 3 and 1 respectively and instructions 1, 2, 6, 8 and 11 are known to be mapped to clusters 3, 2, 0, 1 and 0 respectively. The oracle steering example requires 3 transfer instructions and takes 6 cycles to execute assuming that every operation takes 1 cycle.

6.3 Performance of Bank Predictive Steering

Figures 6.6 and 6.7 shows IPC results for the experiments using statically mapped caches for a variety of steering algorithms. The dependence based steering outperforms the unified cache because of fewer overall misses

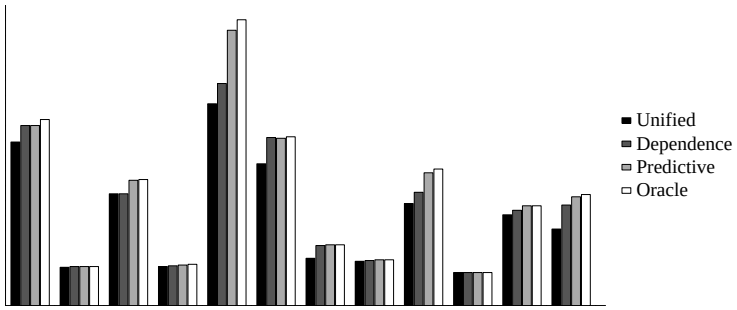


Figure 6.7: IPC values of floating point benchmarks for statically mapped data and cluster centric topology.

and higher available bandwidth to the primary data cache. The bank prediction steering algorithm that predicts which cluster has the data required by the load or store does better than dependence based steering because it is able to reduce the number of remote accesses from 75% to 38% for integer benchmarks. For floating point benchmarks, the number of remote accesses improves from 75% to 52%. Also, the bank predictive steering does a better job of load balancing the instructions across the clusters. For integer benchmarks, the load balance as measured by the variance in the number of instructions executed on a cluster improves from 13 million for dependence steering to a more balanced value of 4 million instructions. Similarly, for floating point benchmarks, the balance improves from 12 million instructions to 3 million instructions. The combination of the reduction in the number of remote loads and better load balance leads to a 19% improvement in mean IPC for integer benchmarks and an 8% improvement for floating point benchmarks.

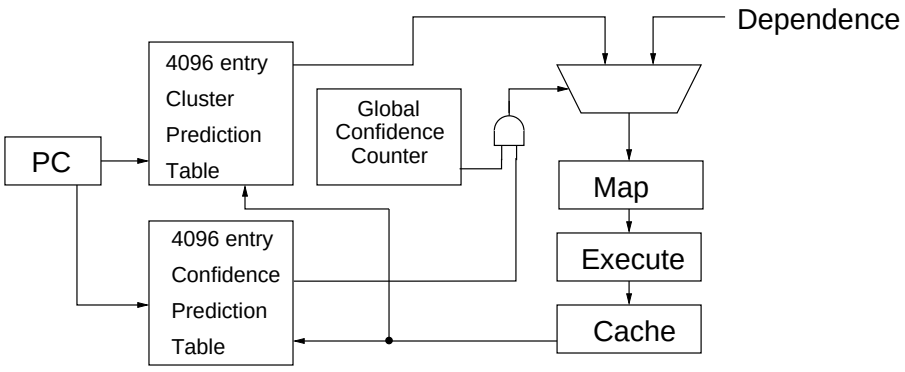


Figure 6.8: Microarchitecture structures used in Hybrid steering

6.4 Hybrid Steering

Some benchmarks like gzip, do better with dependence based steering rather than with the bank prediction steering because the predictive steering does not significantly improve the miss rate. For gzip, predictive steering only improves the DL1 miss rate from 4.8% to 2%. On the other hand, for gcc predictive steering improves DL1 miss rate from to 13% from the 21% DL1 miss rate for dependence steering, helping improve performance by 29%. The hybrid scheme tries to dynamically pick the better of prediction or dependence steering at every point in time. To achieve this, a global confidence counter is used to determine if the map stage should instantaneously use prediction or dependence based steering for a load or store instruction.

Figure 6.8 shows the proposed microarchitecture structures employed by hybrid steering for the mapping load and store instructions to the clusters. The difference from the bank prediction scheme is that there is the global

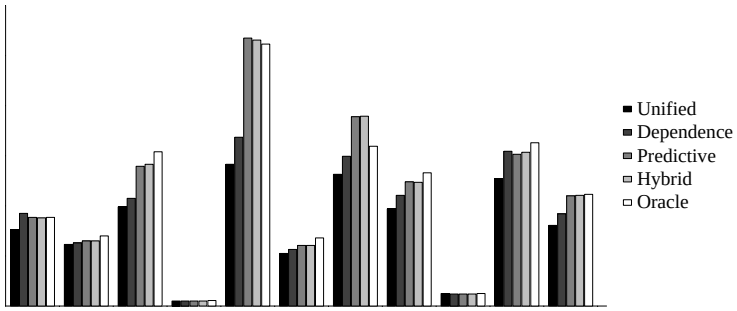


Figure 6.9: IPC of integer benchmarks for hybrid steering algorithm.

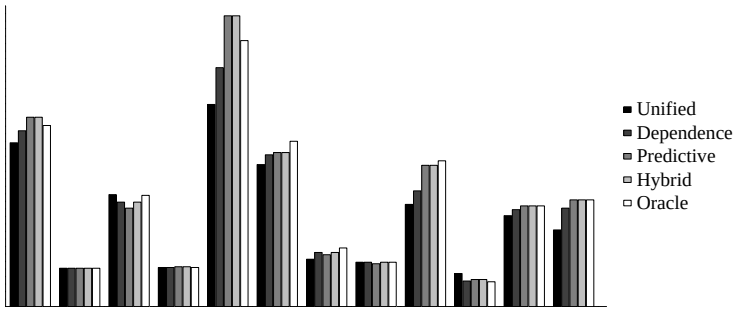


Figure 6.10: IPC of floating point benchmarks for hybrid steering algorithm.

confidence counter that maintains a count of the number of primary cache misses over the past 100,000 cycles. The count is updated every 100 cycles. If the value of the counter is greater than 1000, it indicates low confidence in the ability to correctly predict the cluster to which load/store instructions should be steered. In the case where the counter is over 1000, dependence steering is used to steer load and store instructions. The values picked represent the best average performance using hybrid steering.

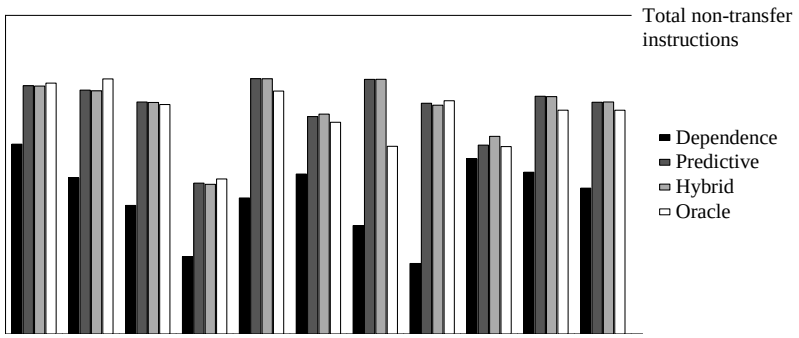


Figure 6.11: Total transfer instructions for integer benchmarks with dynamically mapped data.

Figures 6.9 and 6.10 compare the IPC for the hybrid steering to the IPC of the dependence and bank prediction steering. The hybrid algorithm is able to perform comparably to the better of the two for all of the benchmarks. Hybrid steering has better average IPC that either of the other two steering policies because it encapsulates the best features of dependence and bank prediction steering. Lastly, some of the integer benchmarks such as mcf which have a relative high cache miss rate (66%), show poor performance irrespective of the cache configuration. Benchmark like mcf benefit more from a larger slower cache that encapsulates a larger fraction of the working set.

The effectiveness of the hybrid steering algorithm is evaluated using the number of transfer instructions, the fraction of data in alternate cache banks and the inter-cluster balance. An increase in transfer instructions has the potential to reduce overall performance. In some cases, the increase in transfer instructions leads to a degradation in performance as is evident in

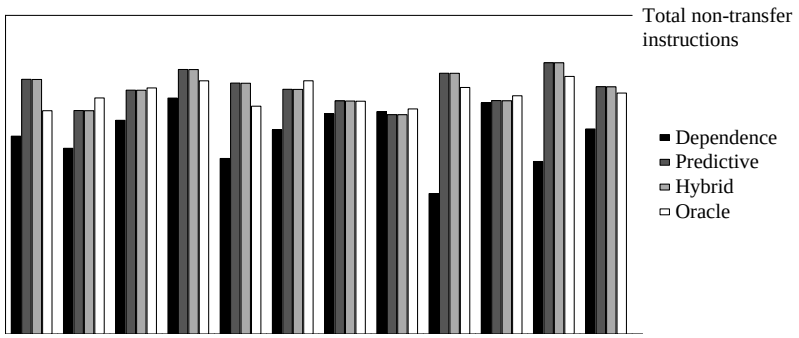
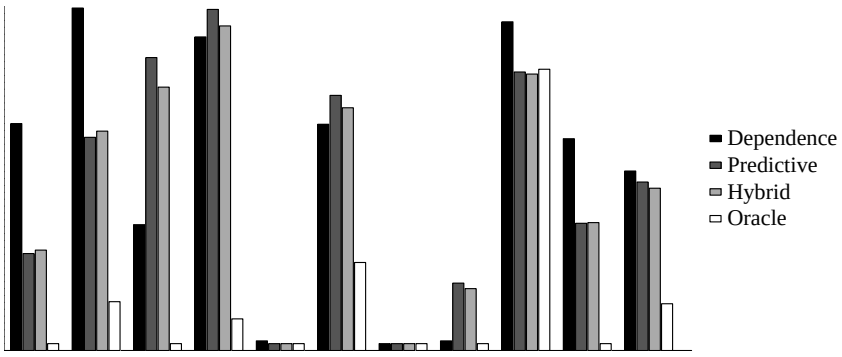


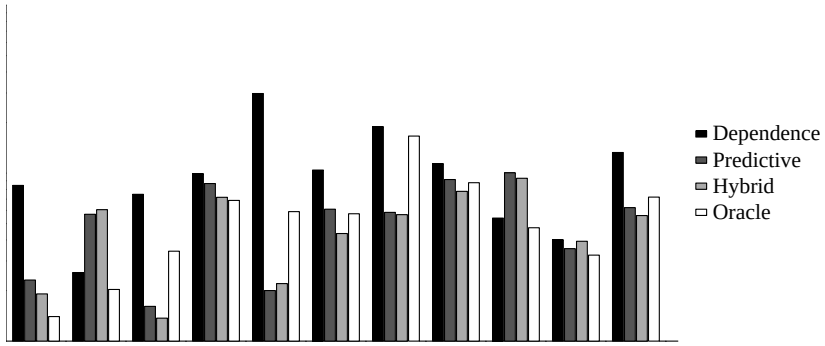
Figure 6.12: Total transfer instructions for floating point benchmarks with dynamically mapped data.

the case of parser, where predictive steering performs better than the hybrid prediction algorithms. On average, the hybrid prediction scheme does not significantly increase the number of transfer instructions as can be seen in Figures 6.11 and 6.12.

Figures 6.13 (a) and 6.14 (a) show the fraction of misses that would have hit in one of the other cache banks. The smaller the bar, the more effective the algorithm is in steering the load and store instructions to the cache bank that contains the data required by the instruction. The oracle steering does the best for most of the benchmarks because instructions are steered based on knowledge of what is in the cache. The reason for the oracle algorithm missing occasionally is that data can be evicted from the cache between the time an instruction is mapped to a cluster and the instruction accesses the local cache of the cluster. The bank predictive and hybrid steering algorithms do substantially better than the dependence algorithm because they intentionally

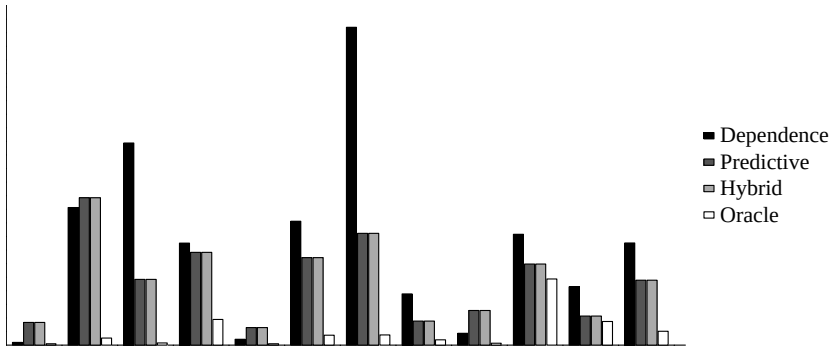


(a)

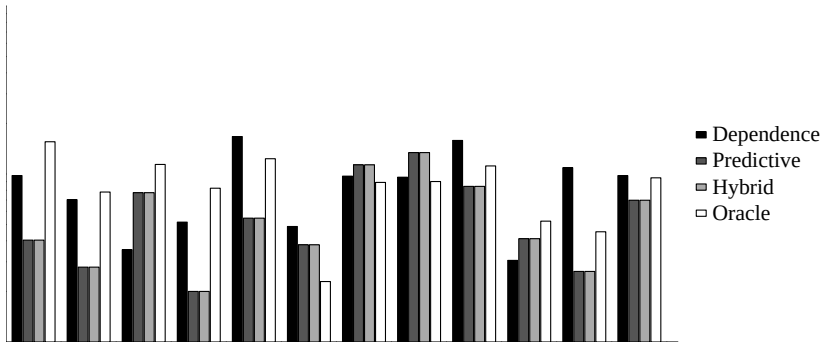


(b)

Figure 6.13: For integer benchmarks with dynamically mapped data (a) misses that could hit in other clusters (b) instruction balance across the clusters.



(a)



(b)

Figure 6.14: For floating point benchmarks with dynamically mapped data (a) misses that could hit in other clusters (b) instruction balance across the clusters.

steer instructions to the cluster where the data exists.

Figures 6.13 (b) and 6.14 (b) show the relative balance of the instructions across the clusters for the entire program execution. The balance is measured as the variance in the number on instructions mapped on each cluster. The lower the bar, the more balanced the instruction distribution across the clusters of the processor. The dependence based algorithm has a very high imbalance between the cluster executing the maximum number of instructions and the cluster executing the least number of instructions. The predictive, hybrid and oracle steering algorithms are able to achieve good balance for the benchmarks in consideration, but this leads to an increase in transfer instructions (as can be seen in Figures 6.11 and 6.12). The increase in transfer instructions does impact performance but with the predictive schemes, this impact on performance is more than offset by the improved matching of loads and stores to the caches banks containing the data required by the instructions.

6.5 Discussion

The results from this chapter show that a hybrid predictive steering algorithm can improve the performance of integer benchmarks by an average of 22% for dynamically mapped data relative to dependence based steering. Similarly, floating point benchmarks using hybrid improve by an average of 16% relative to dependence steering. The improvement in performance is achieved by a combination of reducing the number of misses that could hit in a remote cache bank and the instruction balance across the clusters. The hybrid steering

improve the mean percentage of data in alternate banks from 2.6% to 2.3% for the integer benchmarks and 7.5% to 4.7% for the floating point benchmarks. benchmarks. Simultaneously, the instruction balance improves from 13 million instructions to 6 million instructions for the integer benchmarks and from 10 million instructions to 6 million instructions for the floating point benchmarks.

Hybrid steering improves performance 37% relative to the unified cache for integer benchmarks and 39% for floating point benchmarks. This improvement in performance comes close to the performance of the oracle steering algorithm. Since the hybrid steering with dynamic data mapping is able to achieve performance comparable to the ideal unified cache with dependence based steering, it proves that the combination of the right steering policy coupled with dynamic data mapping can lead to a significant boost in performance for clustered microarchitectures.

Chapter 7

Conclusions

This research examined the effects of technology scaling on wire delays and clock speeds, and measured the expected performance of a modern aggressive microprocessor core in CMOS technologies down to 35nm. We found that communication delays will become significant for global signals. Even under the best conditions, the latency across the chip in a top-level metal wire will be 12–32 cycles, depending on clock rate. In advanced technologies, the delay (in cycles) of memory oriented structures increases substantially due to increased wire latencies and aggressive clock rates. Consequently, even a processor core of today's size does not scale well to future technologies. Figure 7.1 compares our *best* measured performance over the next 14 years with projected scaling at recent historical rates (55% per year). While projected rates for 2014 show performance exceeding one trillion instructions per second, our best-performing microarchitecture languishes at 6.5 BIPS. To reach a factor of thousand in performance improvement at an aggressive clock of 8 FO4 (10GHz in 35nm), a chip must sustain an execution rate of 150 instructions per cycle.

While our results predict that existing microarchitectures do not scale

with technology, we have in fact been quite generous to potential microprocessor scaling. The wire performance models conservatively assume very low-permittivity dielectrics, resistivity of pure copper, high aspect ratio wires, and optimally placed repeaters. The models for structure access time further assume a hierarchical decomposition of the array into sub-banks, word-line routing in mid-level metal wires, and cell areas that do not depend on word-line wire width. In the simulation experiments of sample microarchitectures, the assumption was made that all structures could be perfectly pipelined, that routing delay between structures is insignificant, and that latch and clock skew overheads are negligible.

With these assumptions, the best performance we were able to obtain was a speedup of 7.4 from a 250nm chip to 35nm chip, which corresponds to an annual gain of 12.5%. Over the same period, the clock improves by either 12%, 17%, or 19% annually, depending on whether a clock period at 35nm is 16, 8, or 5.9 FO4 delays, respectively. That result means that the performance of a conventional, out-of-order microprocessor is likely to scale *worse* than the clock rate, even when given an effectively unlimited transistor budget. If any of the optimistic scaling assumptions of our models are not met, actual microprocessor scaling will be poorer than we report.

The models show that dense storage structures will become considerably slower relative to projected clock rates, and will adversely affect instruction throughput. While structure access time remains effectively constant with the clock rate up to 70nm technologies, at 50nm and below, wire delays be-

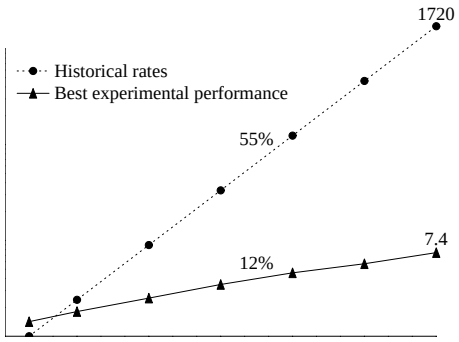


Figure 7.1: Projected performance scaling over a 17-year span for a conventional microarchitecture.

come significant. If clocks are scaled super-linearly relative to decreases in gate length, access times for these structures increases correspondingly. For example, when designing a level-one data cache in a 35nm technology, an engineer will be faced with several unattractive choices. First, the engineer may choose an aggressive target clock rate, and attempt to design a low access penalty cache. At the aggressive ITRS projection of 13.5 GHz (which is likely unrealistic), even a single-ported *512 byte* cache will require three cycles to access. Second, the designer may opt for a larger cache with a longer access time. Given our conservative assumptions about cache designs, a 64KB L1 data cache would require at least seven cycles to access at the ITRS projected clock rate. Finally, the designer may choose a slower clock but a less constrained cache. At 5 GHz (16 FO4 delays), a 32KB cache can be accessed in two cycles.

None of these choices are ideal. The first two alternatives reduce IPC

substantially, while the third incurs a 2.7-fold penalty in clock rate. Optimizing for any one of clock rate, pipeline depth, or structure size will force significant compromises in the other design points for future ultra-small gate-length technologies. While other work has proposed to vary the clock rate and effective structure capacity dynamically [3], those trade-offs are still within the context of a conventional microarchitecture, which is not scalable no matter which balance between clock rate and instruction throughput is chosen.

The results of this study paint a bleak picture for conventional microarchitectures. Clock scaling will soon slow precipitously to linear scaling, which will force architects to use the large transistor budgets to compensate. While it is likely that research innovations will allow conventional microarchitectures to scale better than our results show, we believe that the twin challenges—of recent diminishing returns in ILP and poor scaling of monolithic cores with technology—will force designers to consider more radical alternatives.

One key challenge is to design cores within each partition that can sustain high ILP at fast clock rates. While a monolithic cache forces the design to trade off bandwidth for latency, a partitioned design allows for high bandwidth and higher capacity with low access latency.

This research examined different clustered cache topologies, static and dynamic algorithms for assigning cache lines to cache banks, and several instruction steering algorithms designed to place load and store instructions near the cache banks holding their data. The results show that with 4-cluster, 16-wide processor and a projected 50nm technology, topology has little affect on

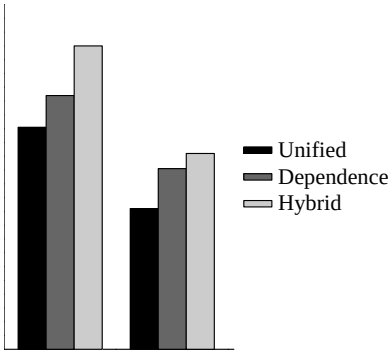


Figure 7.2: Mean IPC for various steering algorithms for integer and floating point benchmarks.

overall performance, as long as both the processor and cache are clustered. The results also show that even with advanced steering policies, statically mapping the cache lines to cache banks in an interleaved fashion produced little benefit over a slower monolithic cache. We proposed a predictive steering policy that predicts where where load/store instruction should be mapped and found that in some circumstances it produced better results over a more traditional dependence-based steering algorithm. As a result, we proposed and evaluated a hybrid policy that chooses between the predictive and dependence-based steering algorithms on the fly depending on the effectiveness of the predictor. Figure 7.2 shows the mean IPC for integer (INT) and floating point (FP) benchmarks for the unified cache, and the clustered cache with dependence and hybrid steering. Using the hybrid steering algorithm improved performance relative to the dependence-based algorithm by an average of 19% and 37% relative to the unified cache for the integer benchmarks. The correspond-

ing improvements in performance were 8% and 39% for the floating point benchmarks.

As feature sizes get smaller and pipeline stages get shorter, the disparity in access latency between the monolithic cache and a single bank will increase. Simultaneously, the inter-cluster communication latency will increase. The net result of both of these will be the need for even more complementary data mapping scheme and instruction steering algorithms that minimize both remote cache and register accesses. A further increase in communication latency may motivate hierarchical clustering in which each group of clusters shared a cache-centric design, but the global organization more closely resembles a cluster-centric architecture.

Appendices

Appendix A

Structure Access Times

A.1 Cache Access Times

Cache access times for various cache configurations in ns are listed in this appendix. The capacity, associativity, block size and number of ports is varied along with the technology.

Size (KB)	Ports	Direct mapped Block size (bytes) 32	64	128	256	2-way set associative Block size (bytes) 32	64	128	256
4	1	1.12	1.17	1.31	1.58	1.37	1.51	1.81	2.38
	2	1.19	1.26	1.45	1.85	1.45	1.63	2.03	2.88
8	1	1.23	1.27	1.36	1.65	1.43	1.55	1.84	2.50
	2	1.33	1.39	1.55	1.98	1.55	1.70	2.15	3.09
16	1	1.36	1.41	1.52	1.75	1.53	1.63	1.89	2.61
	2	1.50	1.58	1.76	2.19	1.71	1.85	2.25	3.34
32	1	1.53	1.59	1.69	1.94	1.67	1.76	2.04	2.80
	2	1.75	1.86	2.03	2.53	1.92	2.09	2.54	3.74
64	1	1.74	1.77	1.90	2.19	1.85	1.96	2.20	3.07
	2	2.08	2.14	2.38	2.96	2.18	2.39	2.97	4.50
128	1	2.07	2.13	2.26	2.57	2.14	2.28	2.58	3.43
	2	2.68	2.76	3.06	3.63	2.77	3.08	3.64	5.47
256	1	2.47	2.54	2.70	3.10	2.55	2.71	3.11	4.08
	2	3.36	3.46	3.79	4.62	3.47	3.80	4.63	6.77
512	1	3.22	3.29	3.45	4.09	3.30	3.47	4.10	5.08
	2	4.94	5.04	5.28	6.66	5.07	5.30	6.68	8.71
1024	1	4.51	4.60	4.80	5.53	4.62	4.81	5.55	6.93
	2	7.18	7.33	7.63	9.25	7.35	7.65	9.27	12.29
2048	1	6.55	6.64	6.87	7.35	6.66	6.89	7.37	9.86
	2	11.95	12.11	12.49	13.28	12.15	12.53	13.32	18.37
4096	1	10.11	10.24	10.53	11.15	10.26	10.56	11.17	14.49
	2	18.71	18.96	19.47	20.52	19.00	19.51	20.56	27.68
8192	1	15.24	15.49	16.09	17.31	15.53	16.13	17.35	19.51
	2	31.42	31.88	32.90	34.99	31.95	32.97	35.06	40.05

Table A.1: Cache access time in ns for various cache configurations in a 250nm technology.

Size (KB)	Ports	4-way set associative Block size (bytes) 32	64	128	256	8-way set associative Block size (bytes) 32	64	128	256
4	1	1.53	1.80	2.31	NA	1.87	2.34	NA	NA
	2	1.66	2.03	2.82	NA	2.12	2.87	NA	NA
8	1	1.57	1.82	2.42	3.74	1.90	2.46	3.74	NA
	2	1.73	2.14	3.03	5.79	2.24	3.08	5.79	NA
16	1	1.65	1.88	2.54	4.01	1.95	2.57	4.01	8.63
	2	1.89	2.25	3.31	6.30	2.34	3.33	6.30	15.59
32	1	1.78	2.02	2.72	4.34	2.10	2.76	4.34	9.26
	2	2.09	2.54	3.74	6.97	2.59	3.74	6.97	16.83
64	1	2.00	2.20	3.07	4.82	2.27	3.07	4.82	10.10
	2	2.42	2.97	4.50	7.99	2.97	4.50	7.99	18.56
128	1	2.29	2.58	3.43	5.62	2.63	3.43	5.62	11.32
	2	3.08	3.64	5.47	9.64	3.64	5.47	9.64	21.14
256	1	2.71	3.11	4.08	6.76	3.11	4.08	6.76	13.21
	2	3.80	4.63	6.77	12.13	4.63	6.77	12.13	25.10
512	1	3.47	4.10	5.08	8.51	4.10	5.08	8.51	15.92
	2	5.30	6.68	8.71	16.10	6.68	8.71	16.10	31.06
1024	1	4.81	5.55	6.93	10.80	5.55	6.93	10.80	20.06
	2	7.65	9.27	12.29	20.82	9.27	12.29	20.82	40.42
2048	1	6.89	7.37	9.86	13.84	7.37	9.86	13.84	26.60
	2	12.53	13.32	18.37	27.18	13.32	18.37	27.18	55.28
4096	1	10.56	11.17	14.49	19.54	11.17	14.49	19.54	33.90
	2	19.51	20.56	27.68	39.19	20.56	27.68	39.19	71.61
8192	1	16.13	17.35	19.51	28.61	17.35	19.51	28.61	44.22
	2	32.97	35.06	40.05	59.29	35.06	40.05	59.29	94.31

Table A.2: Cache access time in ns for various cache configurations in a 250nm technology.

Size (KB)	Ports	Direct mapped Block size (bytes) 32	64	128	256	2-way set associative Block size (bytes) 32	64	128	256
4	1	0.91	0.94	1.04	1.25	1.17	1.28	1.51	1.93
	2	0.96	1.00	1.13	1.42	1.22	1.36	1.64	2.22
8	1	1.01	1.03	1.09	1.29	1.22	1.32	1.54	2.00
	2	1.08	1.11	1.21	1.50	1.30	1.42	1.70	2.34
16	1	1.11	1.14	1.21	1.36	1.30	1.38	1.59	2.06
	2	1.21	1.25	1.38	1.62	1.41	1.52	1.79	2.47
32	1	1.24	1.27	1.36	1.52	1.40	1.47	1.70	2.17
	2	1.37	1.43	1.57	1.87	1.56	1.65	1.97	2.68
64	1	1.40	1.43	1.50	1.72	1.54	1.61	1.80	2.36
	2	1.63	1.67	1.80	2.22	1.75	1.85	2.23	3.05
128	1	1.66	1.70	1.80	1.99	1.72	1.81	2.02	2.63
	2	2.02	2.07	2.26	2.64	2.08	2.27	2.66	3.67
256	1	1.95	1.99	2.13	2.34	2.00	2.14	2.35	3.06
	2	2.52	2.57	2.82	3.24	2.58	2.83	3.25	4.76
512	1	2.53	2.57	2.66	3.04	2.58	2.67	3.06	3.76
	2	3.63	3.68	3.81	4.52	3.70	3.83	4.54	5.97
1024	1	3.39	3.44	3.55	4.06	3.45	3.56	4.07	4.91
	2	5.00	5.07	5.23	6.25	5.09	5.25	6.27	8.01
2048	1	4.94	4.98	5.11	5.37	5.00	5.12	5.39	6.67
	2	8.25	8.34	8.54	8.95	8.37	8.56	8.98	11.51
4096	1	7.18	7.24	7.39	7.72	7.25	7.41	7.74	9.52
	2	12.16	12.29	12.55	13.10	12.32	12.58	13.12	16.86
8192	1	10.66	10.78	11.09	11.75	10.80	11.12	11.78	13.42
	2	20.12	20.34	20.86	21.95	20.38	20.90	21.99	25.31

Table A.3: Cache access time in ns for various cache configurations in a 180nm technology.

Size (KB)	Ports	4-way set associative Block size (bytes) 32	64	128	256	8-way set associative Block size (bytes) 32	64	128	256
4	1	1.31	1.50	1.86	NA	1.59	1.92	NA	NA
	2	1.39	1.64	2.16	NA	1.74	2.23	NA	NA
8	1	1.34	1.53	1.93	2.87	1.62	1.99	2.87	NA
	2	1.45	1.69	2.27	4.22	1.80	2.34	4.22	NA
16	1	1.41	1.58	1.99	3.02	1.67	2.04	3.02	6.49
	2	1.56	1.80	2.41	4.50	1.90	2.47	4.50	10.27
32	1	1.50	1.69	2.09	3.20	1.78	2.15	3.20	6.83
	2	1.69	1.97	2.62	4.85	2.08	2.68	4.85	11.67
64	1	1.65	1.80	2.30	3.46	1.89	2.35	3.46	7.26
	2	1.90	2.23	3.04	5.38	2.31	3.06	5.38	12.55
128	1	1.81	2.02	2.57	3.91	2.07	2.64	3.91	7.90
	2	2.27	2.66	3.67	6.26	2.66	3.67	6.26	13.87
256	1	2.14	2.35	3.06	4.54	2.43	3.06	4.54	8.90
	2	2.83	3.25	4.76	7.57	3.25	4.76	7.57	15.91
512	1	2.67	3.06	3.76	5.54	3.06	3.76	5.54	10.33
	2	3.83	4.54	5.97	9.71	4.54	5.97	9.71	18.96
1024	1	3.56	4.07	4.91	7.45	4.07	4.91	7.45	12.51
	2	5.25	6.27	8.01	13.69	6.27	8.01	13.69	23.73
2048	1	5.12	5.39	6.67	9.61	5.39	6.67	9.61	16.16
	2	8.56	8.98	11.51	17.48	8.98	11.51	17.48	31.78
4096	1	7.41	7.74	9.52	12.94	7.74	9.52	12.94	22.50
	2	12.58	13.12	16.86	24.01	13.12	16.86	24.01	46.03
8192	1	11.12	11.78	13.42	18.02	11.78	13.42	18.02	29.14
	2	20.90	21.99	25.31	34.87	21.99	25.31	34.87	58.34

Table A.4: Cache access time in ns for various cache configurations in a 180nm technology.

Size (KB)	Ports	Direct mapped Block size (bytes) 32	64	128	256	2-way set associative Block size (bytes) 32	64	128	256
4	1	0.66	0.68	0.77	0.94	0.83	0.92	1.09	1.43
	2	0.69	0.73	0.84	1.09	0.87	0.98	1.20	1.68
8	1	0.73	0.75	0.80	0.97	0.87	0.94	1.12	1.49
	2	0.78	0.82	0.90	1.15	0.92	1.01	1.25	1.78
16	1	0.80	0.83	0.89	1.03	0.92	0.99	1.15	1.55
	2	0.88	0.91	1.02	1.25	1.00	1.09	1.33	1.91
32	1	0.89	0.92	1.00	1.15	1.00	1.05	1.23	1.64
	2	1.00	1.05	1.17	1.45	1.11	1.19	1.50	2.12
64	1	1.02	1.05	1.11	1.30	1.11	1.16	1.31	1.81
	2	1.20	1.23	1.34	1.69	1.26	1.35	1.70	2.49
128	1	1.22	1.25	1.33	1.51	1.25	1.34	1.52	2.05
	2	1.49	1.53	1.69	2.06	1.54	1.70	2.06	3.02
256	1	1.44	1.48	1.59	1.79	1.49	1.60	1.80	2.46
	2	1.89	1.94	2.14	2.52	1.95	2.15	2.53	3.85
512	1	1.91	1.94	2.03	2.40	1.95	2.04	2.41	3.00
	2	2.78	2.83	2.96	3.63	2.84	2.97	3.64	4.86
1024	1	2.55	2.60	2.70	3.19	2.61	2.71	3.20	3.91
	2	3.82	3.89	4.05	5.01	3.90	4.06	5.02	6.48
2048	1	3.80	3.85	3.97	4.22	3.86	3.98	4.23	5.49
	2	6.44	6.51	6.71	7.12	6.53	6.72	7.14	9.60
4096	1	5.53	5.59	5.75	6.07	5.61	5.76	6.08	7.82
	2	9.45	9.57	9.83	10.38	9.58	9.84	10.39	13.97
8192	1	8.38	8.50	8.81	9.46	8.52	8.83	9.48	10.89
	2	15.84	16.07	16.60	17.69	16.09	16.62	17.71	20.57

Table A.5: Cache access time in ns for various cache configurations in a 130nm technology.

Size (KB)	Ports	4-way set associative Block size (bytes) 32	64	128	256	8-way set associative Block size (bytes) 32	64	128	256
4	1	0.95	1.10	1.39	NA	1.16	1.44	NA	NA
	2	1.01	1.21	1.66	NA	1.29	1.70	NA	NA
8	1	0.96	1.12	1.45	2.31	1.19	1.49	2.31	NA
	2	1.04	1.26	1.78	3.48	1.34	1.80	3.48	NA
16	1	1.01	1.15	1.51	2.45	1.22	1.55	2.45	5.43
	2	1.12	1.34	1.91	3.73	1.41	1.91	3.73	9.14
32	1	1.07	1.22	1.59	2.62	1.30	1.64	2.62	5.76
	2	1.22	1.50	2.12	4.07	1.53	2.12	4.07	9.97
64	1	1.19	1.31	1.77	2.86	1.38	1.81	2.86	6.19
	2	1.38	1.70	2.49	4.57	1.70	2.49	4.57	10.84
128	1	1.34	1.52	2.01	3.27	1.55	2.06	3.27	6.81
	2	1.70	2.06	3.02	5.37	2.06	3.02	5.37	12.12
256	1	1.60	1.80	2.46	3.84	1.81	2.46	3.84	7.77
	2	2.15	2.53	3.85	6.54	2.53	3.85	6.54	14.07
512	1	2.04	2.41	3.00	4.73	2.41	3.00	4.73	9.13
	2	2.97	3.64	4.86	8.43	3.64	4.86	8.43	16.93
1024	1	2.71	3.20	3.91	6.37	3.20	3.91	6.37	11.18
	2	4.06	5.02	6.48	11.59	5.02	6.48	11.59	21.34
2048	1	3.98	4.23	5.49	8.03	4.23	5.49	8.03	14.52
	2	6.72	7.14	9.60	14.74	7.14	9.60	14.74	28.54
4096	1	5.76	6.08	7.82	10.78	6.08	7.82	10.78	20.04
	2	9.84	10.39	13.97	20.10	10.39	13.97	20.10	39.46
8192	1	8.83	9.48	10.89	15.47	9.48	10.89	15.47	25.26
	2	16.62	17.71	20.57	29.94	17.71	20.57	29.94	50.43

Table A.6: Cache access time in ns for various cache configurations in a 130nm technology.

Size (KB)	Ports	Direct mapped Block size (bytes) 32	64	128	256	2-way set associative Block size (bytes) 32	64	128	256
4	1	0.52	0.54	0.61	0.76	0.63	0.70	0.84	1.11
	2	0.55	0.57	0.67	0.88	0.66	0.75	0.92	1.38
8	1	0.57	0.59	0.64	0.78	0.66	0.72	0.86	1.16
	2	0.61	0.65	0.71	0.93	0.70	0.77	0.97	1.48
16	1	0.64	0.65	0.70	0.83	0.70	0.76	0.89	1.21
	2	0.69	0.72	0.81	1.01	0.75	0.83	1.03	1.59
32	1	0.70	0.72	0.79	0.92	0.77	0.80	0.96	1.28
	2	0.78	0.82	0.92	1.16	0.84	0.92	1.17	1.77
64	1	0.81	0.83	0.88	1.04	0.85	0.89	1.05	1.43
	2	0.96	0.98	1.07	1.37	0.99	1.07	1.37	2.06
128	1	0.96	0.99	1.05	1.23	0.99	1.06	1.24	1.65
	2	1.19	1.21	1.33	1.67	1.21	1.34	1.68	2.48
256	1	1.16	1.19	1.29	1.45	1.20	1.29	1.45	2.02
	2	1.52	1.56	1.73	2.04	1.57	1.74	2.05	3.19
512	1	1.56	1.59	1.66	1.97	1.60	1.67	1.98	2.50
	2	2.26	2.31	2.43	2.96	2.32	2.44	2.96	4.05
1024	1	2.08	2.12	2.21	2.62	2.13	2.21	2.63	3.24
	2	3.10	3.17	3.30	4.10	3.17	3.31	4.11	5.34
2048	1	3.17	3.20	3.30	3.52	3.21	3.31	3.53	4.61
	2	5.28	5.34	5.51	5.88	5.35	5.52	5.89	7.99
4096	1	4.59	4.64	4.78	5.06	4.65	4.78	5.06	6.56
	2	7.71	7.81	8.04	8.53	7.81	8.05	8.54	11.55
8192	1	7.04	7.15	7.42	7.98	7.16	7.42	7.98	9.27
	2	12.91	13.11	13.58	14.56	13.12	13.59	14.57	17.05

Table A.7: Cache access time in ns for various cache configurations in a 100nm technology.

Size (KB)	Ports	4-way set associative Block size (bytes) 32	64	128	256	8-way set associative Block size (bytes) 32	64	128	256
4	1	0.73	0.85	1.08	NA	0.91	1.13	NA	NA
	2	0.77	0.93	1.38	NA	1.00	1.38	NA	NA
8	1	0.74	0.86	1.13	1.95	0.93	1.18	1.95	NA
	2	0.79	0.97	1.48	2.98	1.04	1.48	2.98	NA
16	1	0.78	0.89	1.18	2.07	0.95	1.22	2.07	4.79
	2	0.86	1.04	1.59	3.20	1.11	1.59	3.20	7.83
32	1	0.82	0.96	1.27	2.22	1.01	1.30	2.22	5.08
	2	0.93	1.17	1.77	3.49	1.22	1.77	3.49	8.84
64	1	0.91	1.05	1.43	2.43	1.08	1.44	2.43	5.46
	2	1.07	1.37	2.06	3.92	1.37	2.06	3.92	9.61
128	1	1.06	1.24	1.65	2.77	1.24	1.65	2.77	6.02
	2	1.34	1.68	2.48	4.58	1.68	2.48	4.58	10.72
256	1	1.29	1.45	2.02	3.25	1.45	2.02	3.25	6.86
	2	1.74	2.05	3.19	5.55	2.05	3.19	5.55	12.40
512	1	1.67	1.98	2.50	4.00	1.98	2.50	4.00	8.04
	2	2.44	2.96	4.05	7.07	2.96	4.05	7.07	14.83
1024	1	2.21	2.63	3.24	5.35	2.63	3.24	5.35	9.81
	2	3.31	4.11	5.34	9.73	4.11	5.34	9.73	18.48
2048	1	3.31	3.53	4.61	6.95	3.53	4.61	6.95	12.63
	2	5.52	5.89	7.99	12.60	5.89	7.99	12.60	24.31
4096	1	4.78	5.06	6.56	9.23	5.06	6.56	9.23	17.33
	2	8.05	8.54	11.55	16.87	8.54	11.55	16.87	34.10
8192	1	7.42	7.98	9.27	13.30	7.98	9.27	13.30	22.48
	2	13.59	14.57	17.05	25.15	14.57	17.05	25.15	43.76

Table A.8: Cache access time in ns for various cache configurations in a 100nm technology.

Size (KB)	Ports	Direct mapped Block size (bytes) 32	64	128	256	2-way set associative Block size (bytes) 32	64	128	256
4	1	0.37	0.39	0.45	0.56	0.41	0.46	0.57	0.79
	2	0.39	0.41	0.49	0.66	0.43	0.50	0.66	1.04
8	1	0.40	0.42	0.46	0.58	0.43	0.47	0.59	0.84
	2	0.43	0.46	0.52	0.70	0.46	0.53	0.70	1.13
16	1	0.44	0.46	0.51	0.62	0.47	0.51	0.62	0.89
	2	0.48	0.51	0.58	0.76	0.51	0.58	0.77	1.23
32	1	0.49	0.51	0.56	0.68	0.51	0.56	0.68	0.97
	2	0.54	0.57	0.65	0.88	0.58	0.65	0.88	1.38
64	1	0.56	0.58	0.63	0.75	0.58	0.63	0.75	1.11
	2	0.66	0.68	0.76	0.99	0.68	0.76	0.99	1.63
128	1	0.65	0.68	0.74	0.87	0.68	0.74	0.88	1.28
	2	0.80	0.83	0.94	1.20	0.83	0.94	1.20	1.99
256	1	0.78	0.81	0.89	1.04	0.81	0.89	1.04	1.47
	2	1.03	1.07	1.20	1.49	1.07	1.21	1.49	2.34
512	1	1.05	1.08	1.15	1.41	1.08	1.15	1.41	1.86
	2	1.50	1.55	1.68	2.12	1.56	1.69	2.12	3.06
1024	1	1.38	1.42	1.50	1.83	1.42	1.50	1.84	2.38
	2	2.06	2.12	2.26	2.92	2.13	2.26	2.93	4.00
2048	1	2.10	2.14	2.24	2.45	2.14	2.24	2.45	3.50
	2	3.48	3.57	3.74	4.14	3.57	3.75	4.14	6.05
4096	1	2.99	3.04	3.17	3.45	3.05	3.18	3.45	4.89
	2	5.08	5.19	5.42	5.90	5.19	5.42	5.90	8.78
8192	1	4.57	4.68	4.95	5.49	4.69	4.95	5.49	6.58
	2	8.49	8.70	9.16	10.12	8.70	9.17	10.12	12.31

Table A.9: Cache access time in ns for various cache configurations in a 70nm technology.

Size (KB)	Ports	4-way set associative Block size (bytes) 32	64	128	256	8-way set associative Block size (bytes) 32	64	128	256
4	1	0.48	0.58	0.79	NA	0.62	0.80	NA	NA
	2	0.51	0.66	1.04	NA	0.71	1.04	NA	NA
8	1	0.49	0.60	0.84	1.48	0.63	0.84	1.48	NA
	2	0.53	0.70	1.13	2.27	0.74	1.13	2.27	NA
16	1	0.51	0.62	0.89	1.59	0.64	0.89	1.59	3.64
	2	0.58	0.77	1.23	2.47	0.80	1.23	2.47	6.33
32	1	0.56	0.68	0.97	1.72	0.68	0.97	1.72	3.91
	2	0.65	0.88	1.38	2.74	0.88	1.38	2.74	6.85
64	1	0.63	0.75	1.11	1.92	0.75	1.11	1.92	4.28
	2	0.76	0.99	1.63	3.13	0.99	1.63	3.13	7.59
128	1	0.74	0.88	1.28	2.22	0.88	1.28	2.22	4.80
	2	0.94	1.20	1.99	3.73	1.20	1.99	3.73	8.65
256	1	0.89	1.04	1.47	2.65	1.04	1.47	2.65	5.58
	2	1.21	1.49	2.34	4.60	1.49	2.34	4.60	10.21
512	1	1.15	1.41	1.86	3.30	1.41	1.86	3.30	6.68
	2	1.69	2.12	3.06	5.93	2.12	3.06	5.93	12.47
1024	1	1.50	1.84	2.38	4.09	1.84	2.38	4.09	8.30
	2	2.26	2.93	4.00	7.42	2.93	4.00	7.42	15.81
2048	1	2.24	2.45	3.50	5.30	2.45	3.50	5.30	10.80
	2	3.75	4.14	6.05	9.78	4.14	6.05	9.78	20.98
4096	1	3.18	3.45	4.89	6.98	3.45	4.89	6.98	13.47
	2	5.42	5.90	8.78	13.01	5.90	8.78	13.01	26.25
8192	1	4.95	5.49	6.58	10.50	5.49	6.58	10.50	17.65
	2	9.17	10.12	12.31	20.11	10.12	12.31	20.11	34.70

Table A.10: Cache access time in ns for various cache configurations in a 70nm technology.

Size (KB)	Ports	Direct mapped Block size (bytes) 32	64	128	256	2-way set associative Block size (bytes) 32	64	128	256
4	1	0.27	0.29	0.34	0.44	0.31	0.35	0.45	0.68
	2	0.29	0.32	0.39	0.55	0.32	0.39	0.56	0.96
8	1	0.30	0.32	0.36	0.47	0.32	0.36	0.47	0.72
	2	0.32	0.35	0.42	0.60	0.35	0.42	0.60	1.05
16	1	0.33	0.35	0.39	0.50	0.35	0.39	0.50	0.78
	2	0.37	0.40	0.47	0.67	0.40	0.47	0.67	1.16
32	1	0.37	0.39	0.43	0.57	0.39	0.44	0.57	0.86
	2	0.42	0.45	0.53	0.78	0.46	0.53	0.78	1.32
64	1	0.44	0.45	0.50	0.63	0.46	0.50	0.63	1.00
	2	0.53	0.56	0.64	0.89	0.56	0.65	0.89	1.58
128	1	0.52	0.54	0.60	0.74	0.54	0.60	0.74	1.19
	2	0.66	0.68	0.80	1.08	0.68	0.80	1.09	1.97
256	1	0.65	0.67	0.75	0.91	0.68	0.75	0.91	1.39
	2	0.89	0.93	1.09	1.39	0.94	1.09	1.40	2.36
512	1	0.90	0.94	1.02	1.24	0.94	1.02	1.24	1.77
	2	1.29	1.34	1.50	1.95	1.34	1.51	1.95	3.06
1024	1	1.21	1.25	1.33	1.70	1.25	1.33	1.70	2.28
	2	1.90	1.96	2.10	2.86	1.96	2.10	2.86	4.04
2048	1	1.93	1.98	2.09	2.35	1.98	2.09	2.35	3.31
	2	3.07	3.16	3.34	4.18	3.16	3.35	4.18	5.91
4096	1	2.78	2.84	2.97	3.25	2.84	2.98	3.25	4.81
	2	4.87	4.99	5.23	5.74	4.99	5.24	5.74	8.90
8192	1	4.54	4.66	4.92	5.48	4.66	4.93	5.48	6.69
	2	8.68	8.91	9.41	10.17	8.91	9.41	10.18	12.92

Table A.11: Cache access time in ns for various cache configurations in a 50nm technology.

Size (KB)	Ports	4-way set associative Block size (bytes) 32	64	128	256	8-way set associative Block size (bytes) 32	64	128	256
4	1	0.36	0.45	0.68	NA	0.49	0.68	NA	NA
	2	0.41	0.56	0.96	NA	0.57	0.96	NA	NA
8	1	0.37	0.47	0.72	1.39	0.51	0.72	1.39	NA
	2	0.43	0.60	1.05	2.30	0.61	1.05	2.30	NA
16	1	0.39	0.50	0.78	1.50	0.53	0.78	1.50	3.71
	2	0.47	0.67	1.16	2.52	0.67	1.16	2.52	6.83
32	1	0.44	0.57	0.86	1.65	0.57	0.86	1.65	4.00
	2	0.53	0.78	1.32	2.81	0.78	1.32	2.81	7.41
64	1	0.50	0.63	1.00	1.86	0.63	1.00	1.86	4.40
	2	0.65	0.89	1.58	3.24	0.89	1.58	3.24	8.22
128	1	0.60	0.74	1.19	2.19	0.74	1.19	2.19	4.98
	2	0.80	1.09	1.97	3.90	1.09	1.97	3.90	9.39
256	1	0.75	0.91	1.39	2.65	0.91	1.39	2.65	5.83
	2	1.09	1.40	2.36	4.85	1.40	2.36	4.85	11.11
512	1	1.02	1.24	1.77	3.34	1.24	1.77	3.34	7.04
	2	1.51	1.95	3.06	6.29	1.95	3.06	6.29	13.60
1024	1	1.33	1.70	2.28	4.19	1.70	2.28	4.19	8.81
	2	2.10	2.86	4.04	7.94	2.86	4.04	7.94	17.29
2048	1	2.09	2.35	3.31	5.43	2.35	3.31	5.43	11.50
	2	3.35	4.18	5.91	10.31	4.18	5.91	10.31	22.94
4096	1	2.98	3.25	4.81	7.13	3.25	4.81	7.13	14.48
	2	5.24	5.74	8.90	13.71	5.74	8.90	13.71	28.98
8192	1	4.93	5.48	6.69	10.84	5.48	6.69	10.84	19.01
	2	9.41	10.18	12.92	20.69	10.18	12.92	20.69	37.89

Table A.12: Cache access time in ns for various cache configurations in a 50nm technology.

Size (KB)	Ports	Direct mapped Block size (bytes) 32	64	128	256	2-way set associative Block size (bytes) 32	64	128	256
4	1	0.19	0.20	0.24	0.31	0.21	0.24	0.31	0.48
	2	0.20	0.22	0.28	0.40	0.23	0.28	0.40	0.70
8	1	0.21	0.22	0.25	0.33	0.23	0.25	0.33	0.52
	2	0.23	0.25	0.30	0.44	0.25	0.30	0.44	0.77
16	1	0.23	0.25	0.28	0.36	0.25	0.28	0.36	0.57
	2	0.26	0.29	0.34	0.49	0.29	0.34	0.49	0.87
32	1	0.26	0.28	0.31	0.41	0.28	0.31	0.41	0.64
	2	0.30	0.33	0.39	0.58	0.33	0.39	0.58	1.01
64	1	0.32	0.32	0.36	0.46	0.33	0.36	0.46	0.75
	2	0.39	0.41	0.48	0.66	0.41	0.48	0.66	1.23
128	1	0.38	0.39	0.44	0.54	0.39	0.44	0.54	0.89
	2	0.49	0.51	0.60	0.82	0.51	0.60	0.82	1.52
256	1	0.48	0.49	0.55	0.68	0.50	0.55	0.68	1.03
	2	0.67	0.70	0.81	1.07	0.70	0.81	1.07	1.79
512	1	0.67	0.70	0.76	0.94	0.70	0.76	0.94	1.34
	2	0.98	1.03	1.15	1.52	1.03	1.15	1.53	2.36
1024	1	0.91	0.94	1.00	1.27	0.94	1.00	1.28	1.75
	2	1.46	1.51	1.62	2.18	1.51	1.62	2.18	3.17
2048	1	1.47	1.51	1.60	1.79	1.51	1.60	1.79	2.58
	2	2.41	2.48	2.63	3.23	2.48	2.63	3.24	4.72
4096	1	2.12	2.17	2.28	2.51	2.17	2.28	2.51	3.73
	2	3.80	3.89	4.09	4.51	3.89	4.10	4.51	6.98
8192	1	3.50	3.59	3.81	4.26	3.59	3.81	4.26	5.15
	2	6.81	6.99	7.40	8.14	7.00	7.40	8.14	10.07

Table A.13: Cache access time in ns for various cache configurations in a 35nm technology.

Size (KB)	Ports	4-way set associative Block size (bytes) 32	64	128	256	8-way set associative Block size (bytes) 32	64	128	256
4	1	0.25	0.32	0.48	NA	0.35	0.48	NA	NA
	2	0.29	0.40	0.70	NA	0.41	0.70	NA	NA
8	1	0.26	0.33	0.52	1.01	0.36	0.52	1.01	NA
	2	0.30	0.44	0.77	1.71	0.44	0.77	1.71	NA
16	1	0.28	0.36	0.57	1.10	0.37	0.57	1.10	2.72
	2	0.34	0.49	0.87	1.89	0.49	0.87	1.89	5.07
32	1	0.31	0.41	0.64	1.23	0.41	0.64	1.23	2.97
	2	0.39	0.58	1.01	2.14	0.58	1.01	2.14	5.56
64	1	0.36	0.46	0.75	1.41	0.46	0.75	1.41	3.31
	2	0.48	0.66	1.23	2.51	0.66	1.23	2.51	6.25
128	1	0.44	0.54	0.89	1.68	0.54	0.89	1.68	3.80
	2	0.60	0.82	1.52	3.06	0.82	1.52	3.06	7.24
256	1	0.55	0.68	1.03	2.07	0.68	1.03	2.07	4.52
	2	0.81	1.07	1.79	3.86	1.07	1.79	3.86	8.70
512	1	0.76	0.94	1.34	2.65	0.94	1.34	2.65	5.54
	2	1.15	1.53	2.36	5.04	1.53	2.36	5.04	10.80
1024	1	1.00	1.28	1.75	3.17	1.28	1.75	3.17	7.03
	2	1.62	2.18	3.17	6.06	2.18	3.17	6.06	13.92
2048	1	1.60	1.79	2.58	4.18	1.79	2.58	4.18	9.21
	2	2.63	3.24	4.72	8.03	3.24	4.72	8.03	18.48
4096	1	2.28	2.51	3.73	5.58	2.51	3.73	5.58	11.00
	2	4.10	4.51	6.98	10.86	4.51	6.98	10.86	22.15
8192	1	3.81	4.26	5.15	8.60	4.26	5.15	8.60	14.70
	2	7.40	8.14	10.07	16.64	8.14	10.07	16.64	29.56

Table A.14: Cache access time in ns for various cache configurations in a 35nm technology.

A.2 Register File Access Time

Register File access times for various register files are listed in this appendix. The capacity, entry size and number of ports is varied along with the technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	0.80	0.86	0.94	1.02	0.87	0.93	1.03	1.13	0.89	0.96	1.06	1.15
64	0.83	0.91	1.01	1.13	0.92	1.00	1.11	1.22	0.94	1.04	1.15	1.27
128	0.91	1.00	1.14	1.28	1.00	1.11	1.27	1.44	1.03	1.15	1.31	1.48
256	1.00	1.12	1.30	1.49	1.09	1.22	1.43	1.66	1.12	1.27	1.49	1.73

Table A.15: Register file access time in ns for various configurations in a 250nm technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	0.68	0.71	0.77	0.82	0.71	0.77	0.82	0.87	0.74	0.78	0.83	0.90
64	0.70	0.75	0.81	0.87	0.75	0.80	0.87	0.95	0.76	0.82	0.91	0.98
128	0.75	0.81	0.90	0.99	0.81	0.87	0.97	1.07	0.83	0.90	1.01	1.13
256	0.82	0.90	1.02	1.13	0.88	0.98	1.11	1.25	0.90	1.00	1.15	1.30

Table A.16: Register file access time in ns for various configurations in a 180nm technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	0.48	0.50	0.53	0.57	0.51	0.54	0.57	0.61	0.52	0.55	0.59	0.64
64	0.49	0.52	0.56	0.60	0.53	0.56	0.61	0.67	0.54	0.58	0.64	0.70
128	0.53	0.57	0.62	0.68	0.57	0.62	0.69	0.76	0.59	0.65	0.72	0.80
256	0.58	0.63	0.71	0.78	0.63	0.69	0.78	0.88	0.65	0.72	0.82	0.93

Table A.17: Register file access time in ns for various configurations in a 130nm technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	0.36	0.37	0.39	0.41	0.39	0.41	0.43	0.46	0.40	0.42	0.45	0.48
64	0.37	0.39	0.42	0.45	0.41	0.43	0.46	0.50	0.42	0.45	0.49	0.53
128	0.41	0.43	0.47	0.50	0.45	0.48	0.52	0.57	0.46	0.50	0.55	0.60
256	0.45	0.48	0.53	0.58	0.49	0.54	0.60	0.66	0.51	0.56	0.63	0.70

Table A.18: Register file access time in ns for various configurations in a 100nm technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	0.24	0.25	0.26	0.27	0.27	0.28	0.30	0.32	0.28	0.29	0.31	0.33
64	0.25	0.27	0.28	0.30	0.28	0.30	0.32	0.35	0.29	0.31	0.34	0.37
128	0.28	0.30	0.32	0.34	0.31	0.33	0.37	0.40	0.32	0.35	0.38	0.42
256	0.31	0.33	0.36	0.39	0.35	0.37	0.41	0.46	0.36	0.39	0.44	0.49

Table A.19: Register file access time in ns for various configurations in a 70nm technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	0.17	0.18	0.20	0.21	0.20	0.21	0.23	0.24	0.21	0.22	0.24	0.25
64	0.18	0.20	0.21	0.22	0.21	0.22	0.24	0.26	0.21	0.23	0.25	0.28
128	0.21	0.22	0.24	0.25	0.23	0.25	0.28	0.31	0.24	0.26	0.29	0.33
256	0.23	0.25	0.27	0.30	0.26	0.28	0.32	0.36	0.27	0.30	0.34	0.39

Table A.20: Register file access time in ns for various configurations in a 50nm technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	0.12	0.13	0.14	0.15	0.14	0.15	0.16	0.17	0.14	0.15	0.17	0.18
64	0.13	0.14	0.15	0.16	0.14	0.16	0.17	0.19	0.15	0.16	0.18	0.20
128	0.14	0.15	0.17	0.18	0.16	0.18	0.20	0.22	0.17	0.19	0.21	0.24
256	0.16	0.17	0.19	0.21	0.18	0.20	0.23	0.26	0.19	0.21	0.25	0.29

Table A.21: Register file access time in ns for various configurations in a 35nm technology.

A.3 Content Addressable Memory Access Time

Content Addressable Memory (CAM) access times for various CAMs are listed in this appendix. The capacity, entry size and number of ports is varied along with the technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	1.39	1.41	1.44	1.46	1.46	1.49	1.53	1.58	1.48	1.52	1.57	1.62
64	1.43	1.42	1.45	1.48	1.47	1.50	1.55	1.59	1.50	1.53	1.58	1.64
128	1.44	1.48	1.52	1.51	1.52	1.56	1.62	1.62	1.54	1.59	1.66	1.67
256	1.47	1.50	1.55	1.59	1.55	1.59	1.65	1.72	1.57	1.62	1.69	1.77
512	1.55	1.61	1.69	1.65	1.63	1.70	1.80	1.77	1.65	1.73	1.85	1.83
1024	1.60	1.66	1.74	1.83	1.68	1.75	1.86	1.97	1.71	1.79	1.90	2.03
2048	1.77	1.90	1.84	1.93	1.85	2.00	1.96	2.08	1.88	2.04	2.01	2.14
4096	1.86	1.99	2.18	2.37	1.95	2.10	2.31	2.54	1.98	2.14	2.37	2.62
8192	2.29	2.63	3.10	3.59	2.39	2.75	3.26	3.80	2.42	2.80	3.33	3.89

Table A.22: CAM time in ns for various configurations in a 250nm technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	1.13	1.15	1.17	1.19	1.19	1.21	1.24	1.27	1.22	1.24	1.27	1.31
64	1.14	1.16	1.18	1.20	1.21	1.23	1.25	1.28	1.23	1.25	1.29	1.32
128	1.18	1.20	1.23	1.22	1.24	1.27	1.31	1.31	1.26	1.29	1.34	1.34
256	1.20	1.22	1.25	1.28	1.26	1.29	1.33	1.37	1.28	1.32	1.36	1.41
512	1.26	1.30	1.35	1.31	1.33	1.37	1.44	1.41	1.35	1.40	1.47	1.45
1024	1.29	1.33	1.39	1.44	1.36	1.41	1.47	1.54	1.39	1.44	1.51	1.58
2048	1.43	1.51	1.45	1.50	1.50	1.59	1.54	1.61	1.52	1.62	1.58	1.66
4096	1.49	1.57	1.68	1.80	1.56	1.66	1.78	1.92	1.59	1.69	1.83	1.97
8192	1.81	2.02	2.31	2.60	1.89	2.11	2.42	2.74	1.92	2.15	2.47	2.80

Table A.23: CAM time in ns for various configurations in a 180nm technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	0.81	0.82	0.84	0.85	0.86	0.87	0.89	0.91	0.87	0.89	0.91	0.94
64	0.82	0.83	0.84	0.86	0.87	0.88	0.90	0.92	0.88	0.90	0.92	0.95
128	0.85	0.86	0.88	0.88	0.89	0.91	0.94	0.94	0.91	0.93	0.96	0.97
256	0.86	0.88	0.90	0.92	0.91	0.93	0.96	0.99	0.93	0.95	0.98	1.02
512	0.91	0.94	0.98	0.95	0.96	0.99	1.04	1.02	0.97	1.01	1.06	1.05
1024	0.94	0.97	1.00	1.04	0.99	1.02	1.07	1.12	1.00	1.04	1.10	1.15
2048	1.04	1.10	1.06	1.10	1.09	1.16	1.13	1.18	1.11	1.18	1.16	1.22
4096	1.09	1.15	1.24	1.33	1.14	1.22	1.32	1.43	1.16	1.24	1.35	1.47
8192	1.34	1.51	1.75	1.99	1.40	1.58	1.83	2.09	1.42	1.61	1.87	2.14

Table A.24: CAM time in ns for various configurations in a 130nm technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	0.63	0.63	0.64	0.65	0.66	0.67	0.69	0.70	0.68	0.69	0.70	0.72
64	0.64	0.64	0.65	0.66	0.67	0.68	0.69	0.71	0.69	0.70	0.71	0.73
128	0.65	0.66	0.68	0.67	0.69	0.71	0.72	0.72	0.70	0.72	0.74	0.74
256	0.67	0.68	0.69	0.71	0.70	0.72	0.74	0.76	0.72	0.73	0.76	0.78
512	0.70	0.73	0.76	0.73	0.74	0.77	0.80	0.79	0.76	0.78	0.82	0.81
1024	0.73	0.75	0.78	0.81	0.77	0.79	0.83	0.87	0.78	0.81	0.85	0.89
2048	0.81	0.86	0.82	0.85	0.85	0.91	0.88	0.92	0.87	0.93	0.90	0.95
4096	0.85	0.90	0.98	1.05	0.89	0.96	1.04	1.12	0.91	0.97	1.06	1.15
8192	1.07	1.22	1.43	1.63	1.12	1.28	1.49	1.71	1.13	1.30	1.52	1.75

Table A.25: CAM time in ns for various configurations in a 100nm technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	0.43	0.43	0.44	0.44	0.45	0.46	0.47	0.48	0.46	0.47	0.48	0.49
64	0.44	0.44	0.44	0.45	0.46	0.47	0.47	0.48	0.47	0.48	0.49	0.50
128	0.44	0.45	0.46	0.47	0.47	0.48	0.49	0.51	0.48	0.49	0.51	0.52
256	0.47	0.46	0.47	0.48	0.49	0.49	0.50	0.52	0.50	0.50	0.52	0.54
512	0.47	0.49	0.51	0.50	0.50	0.52	0.55	0.54	0.51	0.53	0.56	0.56
1024	0.49	0.51	0.53	0.55	0.52	0.54	0.57	0.59	0.53	0.55	0.58	0.61
2048	0.54	0.58	0.57	0.59	0.57	0.61	0.61	0.64	0.59	0.63	0.62	0.66
4096	0.58	0.62	0.67	0.72	0.61	0.65	0.71	0.78	0.62	0.67	0.73	0.80
8192	0.72	0.82	0.96	1.10	0.75	0.86	1.01	1.16	0.76	0.87	1.03	1.18

Table A.26: CAM time in ns for various configurations in a 70nm technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	0.31	0.31	0.32	0.32	0.33	0.34	0.34	0.35	0.34	0.35	0.36	0.37
64	0.32	0.32	0.32	0.33	0.34	0.34	0.35	0.36	0.35	0.35	0.36	0.37
128	0.32	0.33	0.34	0.34	0.34	0.35	0.36	0.38	0.35	0.36	0.38	0.39
256	0.33	0.34	0.34	0.35	0.35	0.36	0.37	0.39	0.36	0.37	0.38	0.40
512	0.35	0.36	0.38	0.37	0.37	0.38	0.40	0.40	0.38	0.39	0.42	0.42
1024	0.36	0.37	0.39	0.41	0.38	0.40	0.42	0.45	0.39	0.41	0.44	0.46
2048	0.40	0.43	0.42	0.44	0.43	0.46	0.46	0.49	0.43	0.47	0.47	0.50
4096	0.43	0.46	0.50	0.55	0.45	0.49	0.54	0.60	0.46	0.50	0.56	0.62
8192	0.54	0.62	0.73	0.84	0.57	0.65	0.77	0.90	0.58	0.67	0.79	0.92

Table A.27: CAM time in ns for various configurations in a 50nm technology.

Number of Entries	32 bit entries No. of Ports				64 bit entries No. of Ports				80 bit entries No of. Ports			
	3	6	10	14	3	6	10	14	3	6	10	14
32	0.22	0.22	0.23	0.23	0.23	0.24	0.24	0.25	0.24	0.24	0.25	0.26
64	0.22	0.22	0.23	0.23	0.24	0.24	0.25	0.25	0.24	0.25	0.25	0.26
128	0.23	0.23	0.24	0.24	0.24	0.25	0.26	0.26	0.25	0.26	0.27	0.27
256	0.23	0.24	0.25	0.25	0.25	0.26	0.27	0.28	0.25	0.26	0.27	0.29
512	0.25	0.26	0.26	0.26	0.26	0.27	0.28	0.29	0.27	0.28	0.30	0.30
1024	0.26	0.27	0.28	0.30	0.27	0.29	0.31	0.33	0.28	0.29	0.32	0.34
2048	0.29	0.31	0.30	0.32	0.31	0.33	0.33	0.36	0.31	0.34	0.34	0.37
4096	0.31	0.33	0.37	0.41	0.33	0.36	0.40	0.45	0.33	0.37	0.41	0.46
8192	0.40	0.46	0.56	0.65	0.42	0.49	0.59	0.69	0.42	0.50	0.61	0.71

Table A.28: CAM time in ns for various configurations in a 35nm technology.

Appendix B

SPEC CPU2000 Benchmark Description

Benchmark	Description
164.gzip	Compression
175.vpr	FPGA Circuit Placement and Routing
176.gcc	C Programming Language Compiler
181.mcf	Combinatorial Optimization
186.crafty	Game Playing: Chess
197.parser	Word Processing
252.eon	Computer Visualization
253.perlbmk	PERL Programming Language
254.gap	Group Theory, Interpreter
255.vortex	Object-oriented Database
256.bzip2	Compression
300.twolf	Place and Route Simulator

Table B.1: List of Integer Benchmarks used in this research.

Benchmark	Description
168.wupwise	Physics / Quantum Chromodynamics
171.swim	Shallow Water Modeling
172.mgrid	Multi-grid Solver: 3D Potential Field
173.applu	Parabolic / Elliptic Partial Differential Equations
177.mesa	3-D Graphics Library
178.galgel	Computational Fluid Dynamics
179.art	Image Recognition / Neural Networks
183.equake	Seismic Wave Propagation Simulation
187.facerec	Image Processing: Face Recognition
188.ammp	Computational Chemistry
189.lucas	Number Theory / Primality Testing
191.fma3d	Finite-element Crash Simulation
200.sixtrack	High Energy Nuclear Physics Accelerator Design
301.apsi	Meteorology: Pollutant Distribution

Table B.2: List of Floating Point Benchmarks used in this research.

Appendix C

Scaling IPC

Scaling	Clock	Tech	164.gzip	175.vpr	176.gcc	181.mcf	197.parser
Pipeline	f_8	35	0.89	0.94	0.83	0.89	0.78
Pipeline	f_8	50	0.89	0.94	0.83	0.89	0.78
Pipeline	f_8	70	0.99	1.07	0.92	1.02	0.88
Pipeline	f_8	100	0.99	1.07	0.92	1.02	0.88
Pipeline	f_8	130	0.99	1.07	0.92	1.02	0.88
Pipeline	f_8	180	0.99	1.07	0.92	1.02	0.88
Pipeline	f_8	250	1.07	1.28	1.03	1.32	1.00
Pipeline	f_{16}	35	1.56	1.74	1.42	1.76	1.41
Pipeline	f_{16}	50	1.56	1.74	1.41	1.76	1.41
Pipeline	f_{16}	70	1.64	1.80	1.48	1.78	1.50
Pipeline	f_{16}	100	1.64	1.80	1.48	1.78	1.50
Pipeline	f_{16}	130	1.64	1.80	1.48	1.78	1.50
Pipeline	f_{16}	180	1.64	1.80	1.48	1.78	1.50
Pipeline	f_{16}	250	1.82	2.21	1.63	2.55	1.66
Pipeline	f_{ITRS}	35	0.64	0.69	0.63	0.66	0.58
Pipeline	f_{ITRS}	50	0.64	0.69	0.63	0.66	0.58
Pipeline	f_{ITRS}	70	0.77	0.87	0.76	0.86	0.71
Pipeline	f_{ITRS}	100	0.99	1.07	0.92	1.02	0.88
Pipeline	f_{ITRS}	130	1.12	1.30	1.07	1.36	1.04
Pipeline	f_{ITRS}	180	1.21	1.47	1.15	1.61	1.14
Pipeline	f_{ITRS}	250	1.64	1.80	1.48	1.78	1.50

Scaling	Clock	Tech	253.perlbmk	256.bzip2	300.twolf	Mean IPC	BIPS
Pipeline	f_8	35	0.91	0.89	0.86	0.87	8.65
Pipeline	f_8	50	0.91	0.89	0.86	0.87	6.05
Pipeline	f_8	70	1.05	0.99	0.93	0.98	4.86
Pipeline	f_8	100	1.05	0.99	0.93	0.98	3.40
Pipeline	f_8	130	1.05	0.99	0.93	0.98	2.61
Pipeline	f_8	180	1.05	0.99	0.93	0.98	1.89
Pipeline	f_8	250	1.34	1.11	0.99	1.13	1.53
Pipeline	f_{16}	35	1.80	1.57	1.28	1.56	7.72
Pipeline	f_{16}	50	1.80	1.57	1.28	1.56	5.40
Pipeline	f_{16}	70	1.86	1.67	1.30	1.62	4.01
Pipeline	f_{16}	100	1.86	1.67	1.30	1.62	2.82
Pipeline	f_{16}	130	1.86	1.67	1.30	1.62	2.17
Pipeline	f_{16}	180	1.86	1.67	1.30	1.62	1.57
Pipeline	f_{16}	250	2.37	1.92	1.36	1.90	1.31
Pipeline	f_{ITRS}	35	0.71	0.65	0.69	0.65	8.83
Pipeline	f_{ITRS}	50	0.71	0.65	0.69	0.65	6.54
Pipeline	f_{ITRS}	70	0.88	0.80	0.78	0.80	4.81
Pipeline	f_{ITRS}	100	1.05	0.99	0.93	0.98	3.43
Pipeline	f_{ITRS}	130	1.36	1.16	0.99	1.17	2.45
Pipeline	f_{ITRS}	180	1.62	1.27	1.02	1.30	1.62
Pipeline	f_{ITRS}	250	1.86	1.67	1.30	1.62	1.21

Table C.1: IPC for integer benchmarks for Pipeline scaling.

Scaling	Clock	Tech	164.gzip	175.vpr	176.gcc	181.mcf	197.parser
Capacity	f_8	35	1.09	1.27	0.99	1.30	1.00
Capacity	f_8	50	1.09	1.27	0.99	1.30	1.00
Capacity	f_8	70	1.09	1.27	0.99	1.33	0.99
Capacity	f_8	100	1.07	1.19	1.03	1.31	0.96
Capacity	f_8	130	1.11	1.30	1.03	1.35	1.02
Capacity	f_8	180	1.11	1.30	1.03	1.35	1.02
Capacity	f_8	250	1.11	1.30	1.04	1.35	1.02
Capacity	f_{16}	35	1.78	2.10	1.48	2.40	1.57
Capacity	f_{16}	50	1.78	2.10	1.48	2.40	1.57
Capacity	f_{16}	70	1.76	2.14	1.50	2.44	1.56
Capacity	f_{16}	100	1.76	2.13	1.51	2.44	1.56
Capacity	f_{16}	130	1.76	2.14	1.50	2.44	1.56
Capacity	f_{16}	180	1.79	2.18	1.57	2.48	1.63
Capacity	f_{16}	250	1.78	2.18	1.58	2.48	1.62
Capacity	f_{ITRS}	35	0.73	0.79	0.64	0.80	0.63
Capacity	f_{ITRS}	50	0.69	0.71	0.59	0.73	0.58
Capacity	f_{ITRS}	70	0.88	1.03	0.82	1.11	0.83
Capacity	f_{ITRS}	100	1.09	1.27	0.99	1.33	1.00
Capacity	f_{ITRS}	130	1.07	1.01	0.85	1.25	0.86
Capacity	f_{ITRS}	180	1.17	1.39	0.99	1.66	1.04
Capacity	f_{ITRS}	250	1.81	2.21	1.58	2.51	1.65

Scaling	Clock	Tech	253.perlbmk	256.bzip2	300.twolf	Mean IPC	BIPS
Capacity	f_8	35	1.29	1.14	1.01	1.13	11.20
Capacity	f_8	50	1.29	1.14	1.01	1.13	7.83
Capacity	f_8	70	1.33	1.14	1.01	1.14	5.63
Capacity	f_8	100	1.33	1.21	1.08	1.14	3.96
Capacity	f_8	130	1.35	1.15	0.99	1.16	3.08
Capacity	f_8	180	1.35	1.15	0.99	1.16	2.23
Capacity	f_8	250	1.36	1.15	0.99	1.16	1.56
Capacity	f_{16}	35	2.20	1.89	1.39	1.82	9.03
Capacity	f_{16}	50	2.20	1.89	1.39	1.82	6.31
Capacity	f_{16}	70	2.26	1.87	1.38	1.83	4.54
Capacity	f_{16}	100	2.29	1.87	1.38	1.83	3.19
Capacity	f_{16}	130	2.27	1.87	1.38	1.83	2.45
Capacity	f_{16}	180	2.29	1.89	1.39	1.87	1.81
Capacity	f_{16}	250	2.31	1.89	1.39	1.87	1.29
Capacity	f_{ITRS}	35	0.84	0.81	0.80	0.75	10.14
Capacity	f_{ITRS}	50	0.76	0.75	0.79	0.70	6.97
Capacity	f_{ITRS}	70	1.10	0.97	0.92	0.95	5.71
Capacity	f_{ITRS}	100	1.33	1.14	1.01	1.14	3.98
Capacity	f_{ITRS}	130	1.25	1.26	1.03	1.06	2.23
Capacity	f_{ITRS}	180	1.57	1.31	1.07	1.26	1.57
Capacity	f_{ITRS}	250	2.30	1.92	1.39	1.89	1.41

Table C.2: IPC for integer benchmarks for Capacity scaling.

Scaling	Clock	Tech	168.wupwise	177.mesa	179.art	183.equake
Pipeline	f_8	35	1.42	1.20	1.07	0.94
Pipeline	f_8	50	1.42	1.20	1.07	0.94
Pipeline	f_8	70	1.52	1.36	1.26	1.08
Pipeline	f_8	100	1.52	1.36	1.26	1.08
Pipeline	f_8	130	1.52	1.36	1.26	1.08
Pipeline	f_8	180	1.52	1.36	1.26	1.08
Pipeline	f_8	250	1.48	1.62	1.53	1.38
Pipeline	f_{16}	35	2.37	2.11	2.02	1.84
Pipeline	f_{16}	50	2.37	2.11	2.02	1.84
Pipeline	f_{16}	70	2.56	2.16	2.05	1.89
Pipeline	f_{16}	100	2.56	2.16	2.05	1.89
Pipeline	f_{16}	130	2.56	2.16	2.05	1.89
Pipeline	f_{16}	180	2.56	2.16	2.05	1.89
Pipeline	f_{16}	250	2.56	2.58	2.61	2.56
Pipeline	f_{ITRS}	35	1.01	0.90	0.81	0.70
Pipeline	f_{ITRS}	50	1.01	0.90	0.81	0.70
Pipeline	f_{ITRS}	70	1.10	1.12	1.06	0.90
Pipeline	f_{ITRS}	100	1.52	1.36	1.26	1.08
Pipeline	f_{ITRS}	130	1.59	1.63	1.55	1.42
Pipeline	f_{ITRS}	180	1.60	1.89	1.93	1.72
Pipeline	f_{ITRS}	250	2.56	2.16	2.05	1.89

Scaling	Clock	Tech	188.ammp	Mean IPC	BIPS
Pipeline	f_8	35	0.83	1.07	10.66
Pipeline	f_8	50	0.83	1.07	7.45
Pipeline	f_8	70	0.97	1.22	6.07
Pipeline	f_8	100	0.97	1.22	4.24
Pipeline	f_8	130	0.97	1.22	3.27
Pipeline	f_8	180	0.97	1.22	2.36
Pipeline	f_8	250	1.27	1.45	1.96
Pipeline	f_{16}	35	1.77	2.01	9.98
Pipeline	f_{16}	50	1.77	2.01	6.98
Pipeline	f_{16}	70	1.79	2.07	5.14
Pipeline	f_{16}	100	1.79	2.07	3.61
Pipeline	f_{16}	130	1.79	2.07	2.78
Pipeline	f_{16}	180	1.79	2.07	2.01
Pipeline	f_{16}	250	2.45	2.55	1.76
Pipeline	f_{ITRS}	35	0.64	0.80	10.78
Pipeline	f_{ITRS}	50	0.64	0.80	7.98
Pipeline	f_{ITRS}	70	0.80	0.99	5.94
Pipeline	f_{ITRS}	100	0.97	1.22	4.28
Pipeline	f_{ITRS}	130	1.30	1.50	3.14
Pipeline	f_{ITRS}	180	1.53	1.73	2.16
Pipeline	f_{ITRS}	250	1.79	2.07	1.55

Table C.3: IPC for floating point benchmarks for Pipeline scaling.

Scaling	Clock	Tech	168.wupwise	177.mesa	179.art	183.equake
Capacity	f_8	35	1.70	1.60	1.53	1.35
Capacity	f_8	50	1.70	1.60	1.53	1.35
Capacity	f_8	70	2.05	1.68	1.53	1.42
Capacity	f_8	100	2.05	1.61	1.45	1.35
Capacity	f_8	130	2.05	1.68	1.54	1.43
Capacity	f_8	180	2.06	1.68	1.54	1.43
Capacity	f_8	250	2.47	1.68	1.54	1.45
Capacity	f_{16}	35	2.61	2.46	2.53	2.38
Capacity	f_{16}	50	2.61	2.46	2.53	2.38
Capacity	f_{16}	70	2.81	2.52	2.54	2.39
Capacity	f_{16}	100	2.81	2.52	2.54	2.38
Capacity	f_{16}	130	2.81	2.52	2.54	2.39
Capacity	f_{16}	180	2.90	2.56	2.55	2.45
Capacity	f_{16}	250	2.94	2.56	2.54	2.43
Capacity	f_{ITRS}	35	1.46	1.08	1.05	0.84
Capacity	f_{ITRS}	50	1.46	0.98	1.03	0.75
Capacity	f_{ITRS}	70	0.99	1.31	1.45	1.14
Capacity	f_{ITRS}	100	2.05	1.68	1.53	1.42
Capacity	f_{ITRS}	130	2.58	1.47	1.80	1.14
Capacity	f_{ITRS}	180	2.59	1.90	1.87	1.70
Capacity	f_{ITRS}	250	2.81	2.57	2.53	2.49

Scaling	Clock	Tech	188.ammp	Mean IPC	BIPS
Capacity	f_8	35	1.26	1.48	14.67
Capacity	f_8	50	1.26	1.48	10.26
Capacity	f_8	70	1.30	1.57	7.81
Capacity	f_8	100	1.29	1.53	5.30
Capacity	f_8	130	1.31	1.58	4.22
Capacity	f_8	180	1.31	1.58	3.06
Capacity	f_8	250	1.31	1.65	2.22
Capacity	f_{16}	35	2.36	2.47	12.23
Capacity	f_{16}	50	2.36	2.47	8.56
Capacity	f_{16}	70	2.40	2.53	6.27
Capacity	f_{16}	100	2.40	2.52	4.39
Capacity	f_{16}	130	2.40	2.53	3.39
Capacity	f_{16}	180	2.43	2.57	2.49
Capacity	f_{16}	250	2.42	2.57	1.78
Capacity	f_{ITRS}	35	0.79	1.02	13.79
Capacity	f_{ITRS}	50	0.76	0.97	9.66
Capacity	f_{ITRS}	70	1.12	1.19	7.14
Capacity	f_{ITRS}	100	1.30	1.57	5.51
Capacity	f_{ITRS}	130	1.29	1.59	3.33
Capacity	f_{ITRS}	180	1.58	1.90	2.37
Capacity	f_{ITRS}	250	2.45	2.57	1.93

Table C.4: IPC for floating point benchmarks for Capacity scaling.

Appendix D

Clustered Cache Performance

D.1 Baseline Choices

Width	gzip	vpr	gcc	mcf	crafty	197.parser
4	1.01	1.08	1.19	0.10	1.19	0.84
8	1.28	1.18	1.83	0.10	2.19	1.00
16	1.42	1.22	2.19	0.10	3.04	1.04
32	1.44	1.22	2.29	0.10	3.19	1.06

Width	eon	perlbmk	gap	bzip2	Mean
4	1.68	0.96	0.24	1.70	1.00
8	2.72	1.53	0.24	2.30	1.44
16	3.11	1.92	0.24	2.42	1.67
32	3.19	1.87	0.24	2.44	1.70

Table D.1: IPC for integer benchmarks for varying issue widths.

Width	wupwise	swim	mgrid	applu	mesa	galgel
4	1.69	0.52	1.18	0.52	1.46	1.67
8	2.19	0.52	1.50	0.53	2.47	2.09
16	2.40	0.52	1.49	0.53	2.96	2.24
32	2.42	0.52	1.48	0.53	3.06	2.26

Width	art	equake	ammp	lucas	fma3d	Mean
4	0.75	0.59	1.06	0.44	1.02	0.99
8	0.79	0.59	1.39	0.44	1.22	1.25
16	0.80	0.60	1.51	0.44	1.27	1.34
32	0.80	0.58	1.51	0.44	1.28	1.35

Table D.2: IPC for floating point benchmarks for varying issue widths.

Capacity	gzip	vpr	gcc	mcf	crafty	parser
32K	0.98	1.22	1.96	0.10	2.53	0.96
64K	1.29	1.24	2.01	0.10	2.89	1.01
128K	1.52	1.23	1.98	0.10	2.83	1.05
256K	1.27	1.16	2.05	0.10	2.50	1.04
512K	1.13	1.10	1.89	0.10	2.18	0.99

Capacity	eon	perlbmk	gap	bzip2	Mean
32K	2.54	2.10	0.25	2.65	1.53
64K	2.67	2.07	0.25	2.64	1.61
128K	2.63	1.94	0.25	2.54	1.61
256K	2.42	1.67	0.24	2.26	1.47
512K	2.21	1.44	0.24	2.04	1.33

Table D.3: IPC for integer benchmarks as a function of unified DL1 capacity.

Capacity	wupwise	swim	mgrid	applu	mesa	galgel
32K	2.19	0.51	1.50	0.52	2.81	1.89
64K	2.19	0.51	1.49	0.53	2.78	1.90
128K	2.18	0.51	1.49	0.52	2.69	1.89
256K	2.14	0.51	1.48	0.52	2.51	1.86
512K	2.11	0.52	1.33	0.52	2.33	1.83

Capacity	art	equake	ammp	lucas	fma3d	Mean
32K	0.64	0.60	1.35	0.44	1.21	1.03
64K	0.64	0.59	1.31	0.44	1.22	1.02
128K	0.63	0.59	1.36	0.44	1.21	1.02
256K	0.61	0.60	1.38	0.44	1.19	1.00
512K	0.64	0.60	1.35	0.41	1.15	0.98

Table D.4: IPC for floating point benchmarks as a function of unified DL1 capacity.

Capacity	Assoc.	gzip	vpr	gcc	mcf	crafty	parser
256K	1	1.45	0.62	1.23	0.09	3.10	0.84
512K	1	1.35	0.76	1.37	0.09	3.17	1.02
1024K	1	1.16	0.92	1.53	0.09	2.99	1.04
2048K	1	0.95	1.05	2.06	0.11	2.65	0.94
4096K	1	0.64	0.92	1.61	0.12	1.94	0.66
256K	2	1.44	0.68	1.27	0.09	3.25	0.92
512K	2	1.37	0.85	1.43	0.09	3.26	1.04
1024K	2	1.16	1.04	1.61	0.09	3.03	1.06
2048K	2	0.95	1.16	2.10	0.11	2.66	0.96
4096K	2	0.64	0.98	1.63	0.13	1.94	0.66
256K	4	1.44	0.71	1.25	0.09	3.28	0.94
512K	4	1.37	0.90	1.41	0.09	3.28	1.08
1024K	4	1.16	1.11	2.23	0.09	3.04	1.07
2048K	4	0.95	1.22	2.13	0.10	2.66	0.96
4096K	4	0.64	1.00	1.63	0.12	1.94	0.66

Capacity	Assoc.	eon	perlbmk	gap	bzip2	Mean
256K	1	3.16	1.97	0.29	1.51	1.43
512K	1	3.17	2.16	0.28	1.81	1.52
1024K	1	3.06	2.12	0.27	2.14	1.53
2048K	1	2.79	2.03	0.25	2.52	1.54
4096K	1	2.13	1.74	0.20	2.24	1.22
256K	2	3.21	2.00	0.29	1.59	1.47
512K	2	3.18	2.18	0.28	1.91	1.56
1024K	2	3.06	2.14	0.27	2.23	1.57
2048K	2	2.79	2.04	0.25	2.60	1.56
4096K	2	2.13	1.75	0.20	2.28	1.24
256K	4	3.22	2.19	0.29	1.63	1.50
512K	4	3.18	2.21	0.28	1.95	1.57
1024K	4	3.06	2.14	0.27	2.25	1.64
2048K	4	2.79	2.04	0.25	2.62	1.57
4096K	4	2.13	1.75	0.20	2.29	1.24

Table D.5: IPC for integer benchmarks as a function of L2 organization.

Capacity	Assoc.	wupwise	swim	mgrid	applu	mesa	galgel
256K	1	2.26	0.49	0.96	0.45	2.89	0.85
512K	1	2.40	0.49	1.28	0.46	2.92	0.96
1024K	1	2.42	0.51	1.29	0.46	2.94	1.00
2048K	1	2.39	0.52	1.24	0.53	2.96	1.63
4096K	1	2.26	0.52	1.10	0.53	2.94	2.46
256K	2	2.30	0.44	1.02	0.47	2.70	0.88
512K	2	2.45	0.51	1.20	0.52	2.94	0.96
1024K	2	2.47	0.51	1.50	0.52	2.95	1.26
2048K	2	2.41	0.52	1.41	0.52	2.96	1.38
4096K	2	2.27	0.52	1.20	0.53	2.94	2.47
256K	4	2.31	0.51	1.05	0.46	2.70	0.82
512K	4	2.48	0.51	1.11	0.54	2.94	0.97
1024K	4	2.48	0.52	1.50	0.53	2.95	1.40
2048K	4	2.40	0.52	1.49	0.53	2.96	2.24
4096K	4	2.29	0.52	1.27	0.53	2.94	2.48

Capacity	Assoc.	art	equake	ammp	lucas	fma3d	Mean
256K	1	0.10	0.56	0.75	0.34	1.10	0.98
512K	1	0.13	0.57	0.89	0.35	1.13	1.05
1024K	1	0.24	0.58	1.09	0.35	1.16	1.09
2048K	1	0.43	0.59	1.21	0.35	1.15	1.18
4096K	1	1.24	0.60	1.14	0.38	1.14	1.30
256K	2	0.10	0.57	0.75	0.35	1.09	0.97
512K	2	0.13	0.59	0.92	0.36	1.22	1.07
1024K	2	0.23	0.59	1.20	0.37	1.24	1.17
2048K	2	0.82	0.60	1.41	0.37	1.25	1.24
4096K	2	1.24	0.60	1.19	0.40	1.23	1.33
256K	4	0.10	0.58	0.75	0.42	1.15	0.99
512K	4	0.11	0.59	0.93	0.43	1.26	1.08
1024K	4	0.19	0.59	1.27	0.43	1.28	1.19
2048K	4	0.80	0.60	1.51	0.44	1.27	1.34
4096K	4	1.24	0.60	1.23	0.46	1.25	1.35

Table D.6: IPC for floating point benchmarks as a function of L2 organization.

D.2 Round Robin Steering

Topology	Mapping	Capacity	gzip	vpr	gcc	mcf	crafty	parser
Cluster Centric	Static	32K	0.77	0.95	2.12	0.10	2.85	0.89
		64K	0.92	0.96	2.16	0.10	3.09	0.92
		128K	1.05	0.94	2.15	0.10	3.02	0.95
		256K	0.94	0.88	2.12	0.10	2.72	0.94
Cluster Centric	Dynamic without replication	32K	0.79	0.93	1.57	0.10	3.20	0.91
		64K	0.99	0.96	1.62	0.10	3.51	0.97
		128K	1.22	0.97	1.70	0.10	3.49	1.01
		256K	1.11	0.92	2.05	0.10	3.18	1.00
Cluster Centric	Dynamic with replication	32K	0.69	0.69	1.25	0.10	2.51	0.73
		64K	0.84	0.69	1.28	0.10	2.65	0.77
		128K	1.00	0.67	1.30	0.10	2.58	0.79
		256K	0.91	0.61	1.35	0.10	2.32	0.76
Cache Centric	Static	32K	0.72	0.91	2.00	0.10	2.61	0.85
		64K	0.85	0.92	2.04	0.10	2.79	0.88
		128K	0.95	0.90	2.02	0.10	2.73	0.91
		256K	0.86	0.85	1.96	0.10	2.49	0.90
Cache Centric	Dynamic without replication	32K	0.70	0.86	1.47	0.10	2.75	0.85
		64K	0.85	0.89	1.52	0.10	2.95	0.90
		128K	1.02	0.89	1.58	0.10	2.93	0.94
		256K	0.94	0.85	1.84	0.10	2.70	0.93
Cache Centric	Dynamic with replication	32K	0.62	0.67	1.21	0.10	2.30	0.70
		64K	0.74	0.67	1.24	0.10	2.41	0.73
		128K	0.87	0.66	1.26	0.10	2.36	0.76
		256K	0.80	0.61	1.30	0.10	2.15	0.73

Topology	Mapping	Capacity	eon	perlbmk	gap	bzip2	Mean
Cluster Centric	Static	32K	2.27	1.91	0.24	2.22	1.43
		64K	2.42	1.87	0.24	2.17	1.49
		128K	2.35	1.74	0.24	2.05	1.46
		256K	2.13	1.50	0.23	1.83	1.34
Cluster Centric	Dynamic without replication	32K	2.67	2.43	0.22	2.75	1.56
		64K	2.94	2.39	0.22	2.71	1.64
		128K	2.87	2.20	0.22	2.57	1.64
		256K	2.59	1.84	0.21	2.26	1.53
Cluster Centric	Dynamic with replication	32K	1.54	1.43	0.22	1.35	1.05
		64K	1.62	1.39	0.21	1.32	1.09
		128K	1.58	1.28	0.21	1.24	1.07
		256K	1.41	1.07	0.21	1.09	0.98
Cache Centric	Static	32K	2.14	1.83	0.24	2.19	1.36
		64K	2.28	1.80	0.24	2.14	1.40
		128K	2.22	1.68	0.23	2.03	1.38
		256K	2.03	1.45	0.23	1.82	1.27
Cache Centric	Dynamic without replication	32K	2.36	2.10	0.22	2.44	1.39
		64K	2.54	2.07	0.22	2.40	1.44
		128K	2.49	1.92	0.21	2.28	1.44
		256K	2.26	1.63	0.21	2.04	1.35
Cache Centric	Dynamic with replication	32K	1.51	1.39	0.21	1.37	1.01
		64K	1.60	1.36	0.21	1.34	1.04
		128K	1.57	1.24	0.21	1.26	1.03
		256K	1.40	1.04	0.20	1.11	0.95

Table D.7: IPC for integer benchmarks as a function of DL1 bank size for round robin steering.

Toplogy	Mapping	Capacity	wupwise	swim	mgrid	applu	mesa	galgel
Cluster Centric	Static	32K	2.06	0.52	1.63	0.54	2.72	1.78
		64K	2.05	0.52	1.63	0.54	2.68	1.82
		128K	2.03	0.52	1.63	0.54	2.58	1.81
		256K	1.98	0.52	1.63	0.53	2.38	1.79
Cluster Centric	Dynamic without replication	32K	2.00	0.52	1.62	0.53	3.03	1.70
		64K	2.00	0.52	1.62	0.53	3.02	1.74
		128K	1.97	0.52	1.62	0.53	2.96	1.74
		256K	1.94	0.52	1.60	0.53	2.72	1.74
Cluster Centric	Dynamic with replication	32K	1.91	0.52	1.58	0.53	2.06	1.65
		64K	1.90	0.52	1.57	0.53	2.05	1.69
		128K	1.88	0.52	1.57	0.53	1.98	1.69
		256K	1.84	0.52	1.55	0.53	1.80	1.68
Cache Centric	Static	32K	1.95	0.52	1.62	0.53	2.51	1.61
		64K	1.95	0.52	1.62	0.53	2.47	1.63
		128K	1.92	0.52	1.61	0.53	2.39	1.63
		256K	1.88	0.52	1.61	0.53	2.23	1.61
Cache Centric	Dynamic without replication	32K	1.87	0.52	1.58	0.53	2.65	1.55
		64K	1.87	0.52	1.58	0.53	2.64	1.58
		128K	1.85	0.52	1.57	0.53	2.58	1.58
		256K	1.81	0.52	1.54	0.53	2.40	1.58
Cache Centric	Dynamic with replication	32K	1.80	0.52	1.54	0.53	2.04	1.52
		64K	1.80	0.52	1.54	0.53	2.02	1.55
		128K	1.78	0.52	1.53	0.53	1.95	1.55
		256K	1.74	0.52	1.50	0.53	1.78	1.54

Topology	Mapping	Capacity	art	equake	ammp	lucas	fma3d	Mean
Cluster Centric	Static	32K	0.76	0.56	1.36	0.44	1.19	1.23
		64K	0.76	0.57	1.39	0.44	1.19	1.23
		128K	0.76	0.56	1.45	0.44	1.19	1.23
		256K	0.76	0.56	1.44	0.44	1.16	1.20
Cluster Centric	Dynamic without replication	32K	0.68	0.57	1.29	0.44	1.21	1.24
		64K	0.68	0.57	1.35	0.42	1.22	1.24
		128K	0.69	0.57	1.44	0.40	1.22	1.24
		256K	0.72	0.56	1.45	0.38	1.19	1.21
Cluster Centric	Dynamic with replication	32K	0.65	0.56	1.18	0.44	1.02	1.10
		64K	0.65	0.55	1.23	0.42	1.03	1.10
		128K	0.66	0.55	1.30	0.40	1.02	1.10
		256K	0.69	0.54	1.32	0.37	0.97	1.07
Cache Centric	Static	32K	0.74	0.56	1.28	0.44	1.15	1.17
		64K	0.74	0.56	1.30	0.44	1.16	1.18
		128K	0.74	0.56	1.36	0.44	1.15	1.17
		256K	0.74	0.56	1.35	0.44	1.12	1.14
Cache Centric	Dynamic without replication	32K	0.67	0.56	1.20	0.44	1.16	1.16
		64K	0.68	0.56	1.25	0.42	1.17	1.16
		128K	0.69	0.56	1.32	0.40	1.16	1.16
		256K	0.72	0.56	1.34	0.38	1.13	1.14
Cache Centric	Dynamic with replication	32K	0.65	0.55	1.12	0.44	0.98	1.06
		64K	0.65	0.55	1.16	0.42	0.99	1.07
		128K	0.66	0.54	1.23	0.40	0.98	1.06
		256K	0.69	0.54	1.24	0.37	0.95	1.03

Table D.8: IPC for floating point benchmarks as a function of DL1 bank size for round robin steering.

D.3 Dependence Steering

Topology	Mapping	Capacity	gzip	vpr	gcc	mcf	crafty	parser
Cluster Centric	Static	32K	0.95	1.22	2.13	0.10	2.66	0.96
		64K	1.20	1.23	2.17	0.10	3.05	1.00
		128K	1.42	1.22	2.17	0.10	2.99	1.04
		256K	1.22	1.15	2.14	0.10	2.61	1.02
Cluster Centric	Dynamic without replication	32K	1.06	1.21	2.10	0.10	3.00	1.02
		64K	1.42	1.25	2.15	0.10	3.44	1.07
		128K	1.85	1.26	2.15	0.10	3.37	1.13
		256K	1.54	1.21	2.25	0.10	3.00	1.12
Cluster Centric	Dynamic with replication	32K	0.93	0.76	1.80	0.10	2.74	0.86
		64K	1.21	0.75	1.83	0.10	3.05	0.89
		128K	1.51	0.72	1.78	0.10	2.91	0.91
		256K	1.26	0.65	1.75	0.10	2.51	0.89
Cache Centric	Static	32K	0.99	1.27	2.20	0.10	2.73	0.99
		64K	1.27	1.28	2.27	0.10	3.18	1.03
		128K	1.53	1.27	2.27	0.10	3.15	1.07
		256K	1.30	1.20	2.26	0.10	2.77	1.06
Cache Centric	Dynamic without replication	32K	1.04	1.21	2.09	0.10	2.94	1.02
		64K	1.39	1.24	2.15	0.10	3.37	1.06
		128K	1.81	1.25	2.14	0.10	3.30	1.12
		256K	1.51	1.21	2.24	0.10	2.94	1.11
Cache Centric	Dynamic with replication	32K	0.94	0.79	1.88	0.10	2.78	0.87
		64K	1.21	0.79	1.91	0.10	3.14	0.90
		128K	1.53	0.76	1.86	0.10	3.03	0.94
		256K	1.28	0.68	1.83	0.10	2.62	0.91

Topology	Mapping	Capacity	eon	perlbmk	gap	bzip2	Mean
Cluster Centric	Static	32K	2.79	2.04	0.25	2.62	1.57
		64K	2.98	2.02	0.24	2.57	1.66
		128K	2.91	1.89	0.24	2.42	1.64
		256K	2.65	1.63	0.24	2.13	1.49
Cluster Centric	Dynamic without replication	32K	2.89	2.31	0.24	3.31	1.72
		64K	3.03	2.31	0.24	3.29	1.83
		128K	2.99	2.21	0.24	3.09	1.84
		256K	2.78	1.85	0.23	2.69	1.68
Cluster Centric	Dynamic with replication	32K	1.89	1.84	0.24	1.74	1.29
		64K	1.94	1.79	0.24	1.68	1.35
		128K	1.85	1.60	0.23	1.55	1.32
		256K	1.63	1.29	0.22	1.31	1.16
Cache Centric	Static	32K	2.95	2.20	0.25	2.98	1.67
		64K	3.14	2.18	0.25	2.92	1.76
		128K	3.10	2.05	0.24	2.75	1.75
		256K	2.83	1.76	0.24	2.38	1.59
Cache Centric	Dynamic without replication	32K	2.88	2.30	0.24	3.26	1.71
		64K	3.02	2.30	0.24	3.25	1.81
		128K	2.97	2.20	0.24	3.08	1.82
		256K	2.77	1.84	0.23	2.67	1.66
Cache Centric	Dynamic with replication	32K	2.11	1.92	0.24	1.89	1.35
		64K	2.18	1.93	0.24	1.83	1.42
		128K	2.08	1.72	0.23	1.68	1.39
		256K	1.79	1.38	0.22	1.43	1.22

Table D.9: IPC for integer benchmarks as a function of DL1 bank size for dependence steering.

Topology	Mapping	Capacity	wupwise	swim	mgrid	applu	mesa	galgel
Cluster Centric	Static	32K	2.44	0.52	1.50	0.54	3.14	2.19
		64K	2.44	0.52	1.50	0.53	3.10	2.24
		128K	2.40	0.52	1.49	0.53	2.96	2.24
		256K	2.32	0.52	1.50	0.53	2.70	2.21
Cluster Centric	Dynamic without replication	32K	2.42	0.51	1.42	0.53	3.33	2.01
		64K	2.38	0.51	1.41	0.53	3.30	2.02
		128K	2.34	0.51	1.39	0.52	3.18	2.02
		256K	2.29	0.49	1.35	0.50	2.87	1.97
Cluster Centric	Dynamic with replication	32K	2.29	0.51	1.45	0.53	2.39	1.98
		64K	2.26	0.51	1.42	0.53	2.32	2.00
		128K	2.23	0.51	1.38	0.52	2.19	1.98
		256K	2.14	0.50	1.32	0.49	1.97	1.93
Cache Centric	Static	32K	2.47	0.52	1.51	0.53	3.31	2.17
		64K	2.46	0.52	1.50	0.53	3.27	2.20
		128K	2.43	0.52	1.50	0.53	3.14	2.22
		256K	2.35	0.52	1.51	0.53	2.85	2.20
Cache Centric	Dynamic without replication	32K	2.39	0.51	1.41	0.53	3.27	1.97
		64K	2.37	0.51	1.40	0.52	3.25	1.99
		128K	2.32	0.51	1.39	0.52	3.14	1.99
		256K	2.27	0.50	1.35	0.50	2.85	1.94
Cache Centric	Dynamic with replication	32K	2.28	0.51	1.42	0.53	2.59	1.95
		64K	2.25	0.51	1.41	0.53	2.53	1.97
		128K	2.23	0.51	1.39	0.52	2.39	1.96
		256K	2.15	0.50	1.34	0.49	2.13	1.91

Topology	Mapping	Capacity	art	equake	ammp	lucas	fma3d	Mean
Cluster Centric	Static	32K	0.80	0.60	1.40	0.45	1.28	1.35
		64K	0.80	0.60	1.44	0.44	1.28	1.35
		128K	0.80	0.60	1.51	0.44	1.27	1.34
		256K	0.81	0.59	1.48	0.44	1.24	1.30
Cluster Centric	Dynamic without replication	32K	0.71	0.60	1.43	0.43	1.30	1.33
		64K	0.71	0.60	1.47	0.36	1.30	1.33
		128K	0.72	0.59	1.54	0.34	1.29	1.31
		256K	0.70	0.57	1.52	0.32	1.25	1.26
Cluster Centric	Dynamic with replication	32K	0.71	0.58	1.33	0.43	1.15	1.21
		64K	0.71	0.58	1.36	0.36	1.13	1.20
		128K	0.72	0.57	1.40	0.34	1.12	1.18
		256K	0.71	0.55	1.37	0.32	1.07	1.12
Cache Centric	Static	32K	0.80	0.60	1.43	0.45	1.29	1.37
		64K	0.80	0.60	1.47	0.44	1.30	1.37
		128K	0.80	0.60	1.55	0.44	1.29	1.36
		256K	0.81	0.60	1.53	0.44	1.25	1.32
Cache Centric	Dynamic without replication	32K	0.71	0.60	1.44	0.43	1.29	1.32
		64K	0.72	0.60	1.48	0.36	1.29	1.32
		128K	0.72	0.59	1.53	0.34	1.28	1.30
		256K	0.71	0.57	1.51	0.32	1.25	1.25
Cache Centric	Dynamic with replication	32K	0.71	0.59	1.37	0.43	1.17	1.23
		64K	0.71	0.59	1.40	0.36	1.17	1.22
		128K	0.72	0.57	1.44	0.34	1.14	1.20
		256K	0.70	0.55	1.40	0.32	1.08	1.14

Table D.10: IPC for floating point benchmarks as a function of DL1 bank size for dependence steering.

D.4 Predictive Steering

Topology	Mapping	Steering	gzip	vpr	gcc	mcf	crafty	parser
Cluster Centric	Static	Predictive	1.34	1.11	2.59	0.10	4.47	1.13
		Hybrid	1.34	1.11	2.58	0.10	4.47	1.13
		Oracle	1.64	1.22	2.76	0.10	4.86	1.17
Cluster Centric	Dynamic without replication	Predictive	1.77	1.30	2.79	0.10	5.35	1.21
		Hybrid	1.75	1.30	2.83	0.10	5.31	1.21
		Oracle	1.77	1.40	3.08	0.11	5.23	1.36
Cluster Centric	Dynamic with replication	Predictive	1.31	0.84	1.95	0.10	3.48	0.92
		Hybrid	1.29	0.90	2.07	0.10	3.84	0.96
		Oracle	1.45	1.02	2.36	0.11	4.12	1.12
Cache Centric	Static	Predictive	1.26	1.10	2.57	0.10	4.16	1.12
		Hybrid	1.27	1.10	2.56	0.10	4.16	1.12
		Oracle	1.44	1.18	2.69	0.10	4.40	1.15
Cache Centric	Dynamic without replication	Predictive	1.53	1.27	2.75	0.10	4.82	1.19
		Hybrid	1.53	1.27	2.75	0.10	4.82	1.19
		Oracle	1.53	1.33	3.01	0.11	4.79	1.32
Cache Centric	Dynamic with replication	Predictive	1.24	0.86	1.97	0.10	3.52	0.93
		Hybrid	1.24	0.87	1.96	0.10	3.53	0.93
		Oracle	1.32	0.97	2.33	0.11	3.86	1.08

Topology	Mapping	Steering	eon	perlbmk	gap	bzip2	Mean
Cluster Centric	Static	Predictive	3.31	2.35	0.24	2.93	1.96
		Hybrid	3.32	2.33	0.24	2.93	1.95
		Oracle	3.65	2.67	0.24	3.19	2.15
Cluster Centric	Dynamic without replication	Predictive	3.78	2.48	0.24	3.03	2.20
		Hybrid	3.79	2.47	0.24	3.01	2.20
		Oracle	3.19	2.66	0.25	3.26	2.23
Cluster Centric	Dynamic with replication	Predictive	1.89	1.50	0.23	1.44	1.37
		Hybrid	2.16	1.64	0.23	1.67	1.49
		Oracle	2.06	1.75	0.24	1.93	1.62
Cache Centric	Static	Predictive	3.21	2.36	0.24	2.92	1.90
		Hybrid	3.22	2.34	0.24	2.92	1.90
		Oracle	3.44	2.49	0.25	3.09	2.02
Cache Centric	Dynamic without replication	Predictive	3.60	2.41	0.24	2.90	2.08
		Hybrid	3.60	2.41	0.24	2.90	2.08
		Oracle	3.15	2.59	0.25	3.16	2.12
Cache Centric	Dynamic with replication	Predictive	2.06	1.53	0.23	1.59	1.40
		Hybrid	2.04	1.52	0.23	1.57	1.40
		Oracle	2.13	1.72	0.24	1.88	1.56

Table D.11: IPC for integer benchmarks for various predictive steering policies.

Topology	Mapping	Steering	wupwise	swim	mgrid	applu	mesa	galgel
Cluster	Static	Predictive	2.40	0.52	1.67	0.54	3.67	2.23
Centric		Hybrid	2.40	0.52	1.67	0.54	3.67	2.23
		Oracle	2.48	0.52	1.68	0.55	3.81	2.25
Cluster	Dynamic	Predictive	2.52	0.51	1.31	0.53	3.87	2.05
Centric	without	Hybrid	2.52	0.51	1.31	0.53	3.87	2.05
	replication	Oracle	2.41	0.51	1.48	0.52	3.54	2.20
Cluster	Dynamic	Predictive	2.35	0.51	1.32	0.52	2.34	1.98
Centric	with	Hybrid	2.36	0.51	1.29	0.52	2.63	2.02
	replication	Oracle	2.28	0.51	1.48	0.52	2.62	2.18
Cache	Static	Predictive	2.38	0.52	1.67	0.54	3.47	2.15
Centric		Hybrid	2.38	0.52	1.67	0.54	3.47	2.15
		Oracle	2.44	0.52	1.67	0.54	3.58	2.15
Cache	Dynamic	Predictive	2.48	0.51	1.34	0.53	3.75	2.00
Centric	without	Hybrid	2.48	0.51	1.34	0.53	3.75	2.00
	replication	Oracle	2.40	0.51	1.51	0.52	3.38	2.14
Cache	Dynamic	Predictive	2.33	0.51	1.32	0.52	2.52	1.96
Centric	with	Hybrid	2.33	0.51	1.32	0.52	2.52	1.96
	replication	Oracle	2.26	0.51	1.50	0.52	2.68	2.12

Topology	Mapping	Steering	art	equake	ammp	lucas	fma3d	Mean
Cluster Centric	Static	Predictive	0.81	0.61	1.77	0.44	1.33	1.45
		Hybrid	0.81	0.61	1.77	0.44	1.33	1.45
		Oracle	0.81	0.61	1.82	0.44	1.33	1.48
Cluster Centric	Dynamic without replication	Predictive	0.69	0.57	1.88	0.36	1.34	1.42
		Hybrid	0.69	0.57	1.88	0.36	1.34	1.42
		Oracle	0.78	0.59	1.94	0.33	1.34	1.42
Cluster Centric	Dynamic with replication	Predictive	0.70	0.55	1.70	0.37	1.16	1.23
		Hybrid	0.71	0.55	1.74	0.37	1.19	1.26
		Oracle	0.77	0.58	1.78	0.33	1.23	1.30
Cache Centric	Static	Predictive	0.81	0.61	1.74	0.44	1.31	1.42
		Hybrid	0.81	0.61	1.74	0.44	1.31	1.42
		Oracle	0.81	0.61	1.79	0.44	1.32	1.44
Cache Centric	Dynamic without replication	Predictive	0.70	0.57	1.83	0.36	1.32	1.40
		Hybrid	0.70	0.57	1.83	0.36	1.32	1.40
		Oracle	0.80	0.59	1.90	0.33	1.33	1.40
Cache Centric	Dynamic with replication	Predictive	0.69	0.55	1.69	0.37	1.17	1.24
		Hybrid	0.69	0.55	1.69	0.37	1.17	1.24
		Oracle	0.77	0.57	1.77	0.33	1.22	1.29

Table D.12: IPC for floating point benchmarks for various predictive steering policies.

Topology	Mapping	Steering	gzip	vpr	gcc	mcf	crafty	parser
Cluster Centric	Static	Dependence	15.6	9.5	6.3	3.1	7.1	10.0
		Predictive	41.3	54.2	33.0	20.2	44.0	36.3
		Hybrid	41.2	54.2	33.0	20.2	44.0	36.4
		Oracle	39.2	52.4	31.7	12.6	43.8	34.7
Cluster Centric	Dynamic without replication	Dependence	15.5	9.5	6.4	3.0	7.1	10.0
		Predictive	36.1	33.9	28.5	8.8	40.0	23.1
		Hybrid	36.0	33.6	28.4	8.7	40.0	24.0
		Oracle	37.5	39.9	27.5	9.4	33.4	21.3
Cluster Centric	Dynamic with replication	Dependence	15.6	9.7	6.4	3.0	7.1	10.0
		Predictive	37.4	37.4	28.7	8.3	39.9	25.3
		Hybrid	37.4	37.3	28.8	8.8	39.8	24.7
		Oracle	36.3	41.6	26.8	9.4	31.8	25.2
Cache Centric	Static	Dependence	15.6	9.5	6.4	3.0	7.1	10.0
		Predictive	41.1	54.2	33.0	20.2	44.0	36.4
		Hybrid	41.1	54.2	33.0	20.2	44.0	36.4
		Oracle	38.7	52.4	31.7	12.6	43.8	34.7
Cache Centric	Dynamic without replication	Dependence	15.5	9.5	6.4	3.0	7.1	10.1
		Predictive	36.1	33.0	28.7	9.5	39.6	23.2
		Hybrid	36.1	33.0	28.7	9.5	39.6	23.2
		Oracle	36.4	40.0	27.6	9.5	33.3	21.3
Cache Centric	Dynamic with replication	Dependence	15.7	9.7	6.4	3.0	7.1	10.0
		Predictive	36.9	37.9	28.3	8.5	40.2	25.4
		Hybrid	36.7	37.5	29.1	8.5	40.3	25.3
		Oracle	35.0	40.7	26.8	9.4	32.1	24.8

Topology	Mapping	Steering	eon	perlbmk	gap	bzip2	Mean
Cluster Centric	Static	Dependence	4.8	2.7	12.5	10.4	8.2
		Predictive	54.6	48.3	25.1	36.1	39.3
		Hybrid	54.6	48.4	29.5	36.1	39.8
		Oracle	55.4	47.0	20.1	33.1	37.0
Cluster Centric	Dynamic without replication	Dependence	4.7	2.7	12.5	10.3	8.2
		Predictive	39.6	28.0	15.3	31.0	28.4
		Hybrid	39.7	27.3	17.4	30.9	28.6
		Oracle	15.0	29.1	15.0	25.4	25.3
Cluster Centric	Dynamic with replication	Dependence	4.9	2.7	12.5	10.2	8.2
		Predictive	41.7	32.2	15.3	31.8	29.8
		Hybrid	41.1	31.9	15.2	31.9	29.7
		Oracle	15.4	25.8	14.2	26.2	25.3
Cache Centric	Static	Dependence	4.8	2.7	12.5	10.4	8.2
		Predictive	54.6	48.2	27.4	36.3	39.5
		Hybrid	54.6	48.3	25.2	36.1	39.3
		Oracle	55.4	47.0	20.1	32.9	36.9
Cache Centric	Dynamic without replication	Dependence	4.7	2.7	12.5	10.5	8.2
		Predictive	39.9	29.5	15.4	31.2	28.6
		Hybrid	39.9	29.5	15.4	31.2	28.6
		Oracle	15.0	28.6	14.9	24.9	25.2
Cache Centric	Dynamic with replication	Dependence	4.9	2.7	12.5	10.4	8.2
		Predictive	41.7	30.0	15.3	30.7	29.5
		Hybrid	41.1	31.0	18.1	32.2	30.0
		Oracle	15.4	26.1	14.4	26.9	25.2

Table D.13: Transfer instructions (millions) for integer benchmarks for various predictive steering policies.

Topology	Mapping	Steering	wupwise	swim	mgrid	applu	mesa	galgel
Cluster Centric	Static	Dependence	17.5	14.7	21.6	28.5	12.9	19.2
		Predictive	46.3	46.8	64.8	59.6	49.4	53.0
		Hybrid	46.3	46.8	64.8	59.6	49.4	53.0
		Oracle	43.9	44.3	63.2	55.9	49.0	48.8
Cluster Centric	Dynamic without replication	Dependence	17.4	14.6	21.9	30.2	12.6	19.0
		Predictive	39.6	25.2	33.9	45.7	37.5	34.3
		Hybrid	39.6	25.2	33.9	45.7	37.5	34.3
		Oracle	25.2	30.3	35.0	38.8	26.9	38.8
Cluster Centric	Dynamic with replication	Dependence	17.6	13.6	21.9	29.6	13.0	19.6
		Predictive	39.6	24.9	31.8	46.0	38.0	33.2
		Hybrid	39.5	24.8	32.5	46.1	38.1	34.0
		Oracle	25.3	28.7	33.9	39.0	26.3	38.8
Cache Centric	Static	Dependence	17.9	15.1	21.5	26.9	12.8	19.1
		Predictive	46.3	46.7	64.8	60.0	49.4	53.3
		Hybrid	46.3	46.7	64.8	60.0	49.4	53.3
		Oracle	43.9	44.2	63.0	56.1	49.0	49.0
Cache Centric	Dynamic without replication	Dependence	17.4	14.3	21.3	28.9	12.5	19.0
		Predictive	39.5	25.2	36.6	45.6	37.8	33.7
		Hybrid	39.5	25.2	36.6	45.6	37.8	33.7
		Oracle	25.4	30.5	35.1	36.7	25.7	38.9
Cache Centric	Dynamic with replication	Dependence	17.7	13.5	21.5	28.1	12.9	19.5
		Predictive	40.1	24.9	33.1	46.5	38.0	33.5
		Hybrid	40.1	24.9	33.1	46.5	38.0	33.5
		Oracle	25.6	28.5	34.3	38.1	26.1	38.8

Topology	Mapping	Steering	art	equake	ammp	lucas	fma3d	Mean
Cluster Centric	Static	Dependence	24.0	22.6	7.7	30.4	11.5	19.1
		Predictive	44.0	51.2	57.4	48.4	57.4	52.6
		Hybrid	44.0	51.2	57.4	48.4	57.4	52.6
		Oracle	34.0	40.3	54.1	32.9	55.0	47.4
Cluster Centric	Dynamic without replication	Dependence	24.1	24.8	7.6	28.2	12.1	19.3
		Predictive	29.0	23.8	43.3	29.1	50.4	35.6
		Hybrid	29.0	23.8	43.3	29.1	50.4	35.6
		Oracle	28.9	25.8	35.2	31.2	41.3	32.5
Cluster Centric	Dynamic with replication	Dependence	24.0	22.8	7.6	29.5	11.8	19.2
		Predictive	29.4	21.7	44.7	28.8	49.6	35.2
		Hybrid	28.2	22.0	44.9	28.8	49.8	35.3
		Oracle	28.7	25.6	35.4	30.6	36.6	31.7
Cache Centric	Static	Dependence	23.9	22.7	7.7	29.3	11.5	18.9
		Predictive	43.3	52.1	57.1	48.6	57.5	52.6
		Hybrid	43.3	52.1	57.1	48.6	57.5	52.6
		Oracle	34.1	40.8	54.0	33.6	55.0	47.5
Cache Centric	Dynamic without replication	Dependence	23.9	24.1	7.6	27.7	11.8	19.0
		Predictive	28.1	25.0	44.6	30.0	50.6	36.1
		Hybrid	28.1	25.0	44.6	30.0	50.6	36.1
		Oracle	30.7	24.5	35.6	30.9	41.5	32.3
Cache Centric	Dynamic with replication	Dependence	24.0	22.7	7.5	28.2	11.4	18.8
		Predictive	27.8	24.2	44.7	28.7	50.9	35.7
		Hybrid	27.8	24.2	44.7	28.7	50.9	35.7
		Oracle	27.9	23.5	35.2	31.0	37.0	31.4

Table D.14: Transfer instructions (millions) for floating point benchmarks for various predictive steering policies.

Bibliography

[1] Vikas Agarwal, M. S. Hrishikesh, Stephen W. Keckler, and Doug Burger. Clock rate versus IPC: The end of the road for conventional microarchitectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June 2000.

[2] Vikas Agarwal, Stephen W. Keckler, and Doug Burger. Scaling of microarchitectural structures in future process technologies. Technical Report TR2000-02, Department of Computer Sciences, The University of Texas at Austin, April 2000.

[3] David H. Albonesi. Dynamic ipc/clock rate optimization. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 282–292, June 1998.

[4] B.S. Amrutur and M.A. Horowitz. Speed and power scaling of SRAMs. *IEEE Journal of Solid State Circuits*, 35(2):175–185, February 2000.

[5] C. J. Anderson, J. Petrovick, J. M. Keaty, J. Warnock, G. Nussbaum, J. M. Tendier, C. Carter, S. Chu, J. Clabes, J. DiLullo, P. Dudley, P. Harvey, B. Krauter, J. LeBlanc, Lu Pong-Fei, B. McCredie ang G. Plum, P. J. Restle, S. Runyon, M. Scheuermann, S. Schmidt, J. Wagoner, R. Weiss, S. Weitzel, and B. Zoric. Physical design of a fourth-generation POWER GHz microprocessor. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 232–233, February 2001.

[6] Amirali Baniasadi and Andreas Moshovos. Instruction distribution heuristics for quad-cluster dynamically-scheduled, superscalar processors. In *Proceedings of the 33rd International Symposium on Microarchitecture*, pages 337–347, December 2000.

[7] Rajeev Barua, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Maps: A compiler-managed memory system for raw machines. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pages 4–15, May 1999.

[8] Mark T. Bohr. Interconnect scaling – the real limiter to high performance ULSI. *Solid State Technology*, 39(9):105–111, September 1996.

[9] Geordie Braceras, Alan Roberts, John Connor, Reid Wistort, Terry Frederick, Marcel Robillard, Stu Hall, Steve Burns, and Matt Graf. A 940MHz data rate 8Mb CMOS SRAM. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 198–199, February 1999.

[10] Doug Burger and Todd M. Austin. The simplescalar tool set version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin, June 1997.

[11] Doug Burger, Alain Kägi, and M.S. Hrishikesh. Memory hierarchy extensions to simplescalar 3.0. Technical Report TR99-25, Department of Computer Sciences, The University of Texas at Austin, April 2000.

[12] Ramon Canal, Joan Manuel Parcerisa, and Antonio Gonzalez. A cost-effective clustered architecture. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 160–168, October 1999.

[13] Ramon Canal, Joan Manuel Parcerisa, and Antonio Gonzalez. Dynamic cluster assignment mechanisms. In *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture*, pages 133–142, Toulouse, France, January 2000. IEEE Computer Society TCCA.

[14] Sangyeun Cho, Pen-Chung Yew, and Gyungho Lee. Decoupling local variable accesses in a wide-issue superscalar processor. In *proceedings of the 26th annual international symposium on computer architecture*, pages 100–110, May 1999.

[15] Rajagopalan Desikan, Doug Burger, and Stephen W. Keckler. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 266–277, July 2001.

[16] Rajagopalan Desikan, Lakshminarasimhan Sethumadhavan, Ramadass Nagarajan, Doug Burger, and Stephen W. Keckler. Lightweight distributed selective re-execution and its implications for value speculation. In *1st Value Prediction Workshop*, June 2003.

[17] Keith Diefendorff. Power4 focuses on memory bandwidth. *Microprocessor Report*, 13(13), October 1999.

[18] Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, and Dean M. Tullsen. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro*, 17(5):12–19, 1997.

[19] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic. The multicluster architecture: Reducing cycle time through partitioning. In *Proceedings of the 30th International Symposium on Microarchitecture*, December 1997.

[20] M. M. Fernandes, J. Llosa, and N. Topham. Distributed modulo scheduling. In *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture*, January 1999.

[21] E. Gibert, J. Sanchez, and A. Gonzalez. Effective instruction scheduling techniques for an interleaved cache clustered VLIW processor. In *Proceedings of the 35th Annual*

International Symposium on Microarchitecture, pages 123– 133, 2002.

[22] Shashank Gupta, Stephen W. Keckler, and Doug Burger. Technology independent area and delay estimates for microprocessors building blocks. Technical Report TR2000-01, Department of Computer Sciences, The University of Texas at Austin, Austin, TX, February 2000.

[23] Lance Hammond, Basem Nayfeh, and Kunle Olukotun. A single-chip multiprocessor. *IEEE Computer*, 30(9):79–85, September 1997.

[24] A. Hartstein and T.R. Puzak. The optimum pipeline depth for a microprocessor. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 7–13, June 2002.

[25] John L. Henning. Spec cpu2000: Measuring cpu performance in the new millennium. *Computer*, 33(7):28–35, 2000.

[26] Dana S. Henry, Bradley C. Kuszmaul, and Vinod Viswanath. The ultrascalar processor - an asymptotically scalable superscalar microarchitecture. In *The 20th Anniversary Conference on Advanced Research in VLSI*, May 1999.

[27] Mark Horowitz, Ron Ho, and Ken Mai. The future of wires. In *Seminconductor Research Corporation Workshop on Interconnects for Systems on a Chip*, May 1999.

[28] M. S. Hrishikesh, N. P. Jouppi, K. I. Farkas, D. Burger, S. W. Keckler, and P. Shivakumar. The optimal logic depth per pipeline stage is 6 to 8 fo4 inverter delays. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 14–24, June 2002.

[29] M. S. Hrishikesh, Stephen W. Keckler, and Doug Burger. Impact of technology scaling on instruction execution throughput. Technical Report TR2000-06, Department of Computer Sciences, The University of Texas at Austin, November 2000.

[30] Ron Kalla, Balaram Sinharoy, and Joel Tendler. POWER5: IBM's next generation POWER microprocessor. In *Proceedings of Hot Chips 15*, August 2003.

[31] R.E. Kessler. The alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, March/April 1999.

[32] Richard E. Kessler. The alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.

[33] Changkyu Kim, Doug Burger, and Stephen W. Keckler. An adaptive, non-uniform cache structure for wire-dominated on-chip caches. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[34] David Kroft. Lockup-free instruction fetch/prefetch cache organization. In *Proceedings of the Eighth International Symposium on Computer Architecture*, pages 81–87, May 1981.

[35] S. R. Kunkel and J. E. Smith. Optimal pipelining in supercomputers. In *Proceedings of the 13th Annual International Symposium on Computer Architecture*, pages 404–411, June 1986.

[36] Bradley C. Kuszmaul, Dana S. Henry, and Gebriel H. Loh. A comparison of scalable superscalar processors. In *Proceedings of the Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 126–137, June 1999.

[37] Walter Lee, Rajeev Barua, Devabhaktuni Srikrishna, Jonathan Babb, Vivek Sarkar, , and Saman Amarasinghe. Space-time scheduling of instruction-level parallelism on a Raw machine. In *Eighth International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS)*, pages 46–57, October 1998.

[38] D. Limaye, R. Rakvic, and J. P. Shen. Parallel cachelets. In *Proceedings. 2001 International Conference on Computer Design*, September 2001.

[39] Doug Matzke. Will physical scalability sabotage performance gains? *IEEE Computer*, 30(9):37–39, September 1997.

[40] S. Naffziger. A subnanosecond $0.5\mu m$ 64b adder design. In *Digest of Technical Papers, International Solid-State Circuits Conference*, pages 362–363, February 1996.

[41] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang. The case for a single-chip multiprocessor. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 2–11, October 1996.

[42] Subbarao Palacharla, Norman P. Jouppi, and J.E. Smith. Complexity-effective superscalar processors. In *Proceedings of the 24th Annual International Symposium on Computer Archtecture*, pages 206–218, June 1997.

[43] Il Park, Chong Liang Ooi, and T.N. Vijaykumar. Reducing design complexity of the load/store queue. In *Proceedings of the 36th Annual International Symposium on Microarchitecture*, December 2003.

[44] Paul Racunas and Yale N. Patt. Partitioned first-level cache design for clustered microarchitectures. In *Proceedings of the 17th annual international conference on*

Supercomputing, pages 22 – 31, June 2003.

[45] Glenn Reinman, Todd Austin, and Brad Calder. A scalable front-end architecture for fast instruction delivery. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pages 234–245, May 1999.

[46] Glenn Reinman and Norm Jouppi. Extensions to cacti, 1999. Unpublished document.

[47] R. Riedlinger and T. Grutkowski. The high-bandwidth 256kb 2nd-level cache on an itanium microprocessor. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 340–537, February 2002.

[48] Jude A. Rivers, Gary S. Tyson, Edward S. Davidson, and Todd M. Austin. On high-bandwidth data cache design for multi-issue processors. In *International Symposium on Microarchitecture*, pages 46–56, 1997.

[49] Scott Rixner, William J. Dally, Brucek Khailany, Peter Mattson, Ujval J. Kapasi, and John D. Owens. Register organization for media processing. In *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture*, January 2000.

[50] Eric Rotenberg, Quinn Jacobson, Yiannakis Sazeides, and Jim Smith. Trace processors. In *Proceedings of 30th Annual International Symposium on Microarchitecture*, pages 138–148, December 1997.

[51] The international technology roadmap for semiconductors. Semiconductor Industry Association, 2001.

[52] Lakshminarasimhan Sethumadhavan, Rajagopalan Desikan, Doug Burger, Charles R. Moore, and Stephen W. Keckler. Scalable hardware memory disambiguation for high

ILP processors. In *Proceedings of the 36th Annual International Symposium on Microarchitecture*, December 2003.

[53] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[54] Hiroshi Shimizu, Kenji Ijitsu, Hideo Akiyoshi, Keizo Aoyama, Hirotaka Takatsuka, Kou Watanabe, Ryota Nanjo, and Yoshihiro Takao. A 1.4ns access 700MHz 288Kb SRAM macro with expandable architecture. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 190–191, 459, February 1999.

[55] Premkishore Shivakumar and Norman P. Jouppi. Cacti 3.0: An integrated cache timing, power and area model. Technical Report 2001/2, Compaq Computer Corporation, August 2001.

[56] Gurindar S. Sohi. Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers. *IEEE Transactions on Computers*, 39(3):349–359, March 1990.

[57] E. Sprangle and D. Carmean. Increasing processor performance by implementing deeper pipelines. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 25–34, June 2002.

[58] Dennis Sylvester and Kurt Keutzer. Rethinking deep-submicron circuit design. *IEEE Computer*, 32(11):25–33, November 1999.

[59] B. S. Thakar and Gyungho Lee. Access region cache: a multi-porting solution for future wide-issue processors. In *Proceedings. 2001 International Conference on Computer*

Design, September 2001.

[60] A. J. van Genderen and N. P. van der Meijs. Xspace user's manual. Technical Report ET-CAS 96-02, Delft University of Technology, Department of Electrical Engineering, August 1996.

[61] Elliot Waingold, Michael Taylor, Devabhaktuni Srikrishna, Vivek Sarkar, Walter Lee, Victor Lee, Jang Kim, Matthew Frank, Peter Finch, Rajeev Barua, Jonathan Babb, Saman Amarasinghe, and Anant Agarwal. Baring it all to software: Raw machines. *IEEE Computer*, 30(9):86–93, September 1997.

[62] Steven J.E. Wilton and Norman P. Jouppi. An enhanced access and cycle time model for on-chip caches. Technical Report 95/3, Digital Equipment Corporation, Western Research Laboratory, 1995.

[63] A. Wolfe and R. Boleyn. Two-ported cache alternatives for superscalar processors. In *Proceedings of the 26th Annual International Symposium on Microarchitecture*, pages 41–48, December 1993.

[64] A. Yoaz, M. Erez, R. Ronen, and S. Jourdan. Speculation techniques for improving load related instruction scheduling. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pages 42–53, May 1999.

[65] Cangsang Zhao, Uddalak Bhattacharya, Martin Denham, Jim Kolousek, Yi Lu, Yong-Gee Ng, Novat Nintunze, Kamal Sarkez, and Hemmige Varadarajan. An 18Mb, 12.3GB/s cmos pipeline-burst cache SRAM with 1.54Gb/s/pin. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 200–201,461, February 1999.

Index

Vita

Vikas Agarwal was born in New Delhi, India on 27 December 1974, the son of Dr. Yogeshwar K. Agarwal and Usha Agarwal. He received the Bachelor of Technology degree in Electrical Engineering from the Indian Institute of Technology, Bombay in 1996. He entered the graduate program in Solid State Electronics at the University of Texas at Austin in September 1996 and graduated with a Master of Science degree in 1998. Subsequently he joined the Ph.D. program in Computer Engineering at the University of Texas at Austin.

Permanent address: 11900 Hobby Horse Court Apt. 212
Austin, Texas 78758

This dissertation was typeset with LaTeX† by the author.

†LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.