1-1-2002

Dynamic data memory partitioning for access region caches

Sun Kyu Park
Iowa State University

Dynamic data memory partitioning for access region caches

by

Sun Kyu Park

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Gyungho Lee, Major Professor
Akhilesh Tyagi
Gurpur Prabhu

Iowa State University

Ames, Iowa

2002

Graduate College
Iowa State University

This is to certify that the Master's thesis of

Sun Kyu Park

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

For wide-issue processors, data cache needs to be heavily multi-ported with extremely wide data-paths. A recent proposal of multi-porting cache design divides memory streams into multiple independent sub-streams with the help of prediction mechanism before they enter the reservation stations. Partitioned memory-reference instructions are then fed into separate memory pipelines, each of which is connected to a small data-cache, called access region cache (ARC). A selection function for mapping memory references to each ARC can affect the data memory bandwidth as conflicts and load balance at each ARC may differ. In this thesis, we study various static and dynamic memory partitioning methods to see the effects of distributing memory references among the ARCs through exposing memory traffic of those designs. Six different approaches of distributing memory references, including two randomization methods and two dynamic methods, are considered. The potential effects on the memory performance with ARC are measured and compared with existing multi-porting solution as well as an ideal multi-ported data cache. This study concludes that scattering access conflicts dynamically, redirecting conflicting references dynamically to different ARCs at each cycle, can increase the memory bandwidth. However, increasing data bandwidth alone does not always results in performance improvement. Keeping the cache miss rate low is as important as sufficient memory bandwidth to achieve higher performance in wide-issue processors.

1. INTRODUCTION

Modern superscalar processors improve performance by increasing the number of instructions executed at very high clock rates and, assisted by hardware mechanism for control speculation, register renaming, and data-flow execution [23, 8]. Researchers are actively proposing very aggressive microarchitectures that can issue up to 16 or more instructions in a single cycle with ample on-chip hardware resources [3]. Efficient handling of memory references is a more critical factor to achieve high performance since exploiting the instruction level parallelism (ILP) places a greater demand on the memory system to service multiple requests per cycle. Cache memories have been used in almost all recent microprocessors to shorten the average memory access latency. Previously, memory latency problem caused by the processor-memory speed gap was a key factor, so the main concerns of memory designers were cache size, associativity and line size. In addition to memory latency problem caused by the processor-memory speed gap, memory bandwidth (or cache bandwidth) to processor core is extremely important for a wide-issue processor to achieve its full performance potential [6, 9, 18, 20]. Due to such high memory bandwidth requirements, single ported cache has become performance bottleneck. Though current microprocessors implement dual ported cache to reduce this bandwidth bottleneck, dual ported cache will be eventually inadequate for more aggressive superscalar microprocessor. There is a need to explore scalable techniques for increasing the effective number of cache ports to provide a sustainable memory bandwidth in wide-issue processors at a very high clock rate. A straightforward approach

for increasing the bandwidth is to implement a multi-ported data cache [20]. Multiple ported data cache can be implemented in following ways [9]: ideal multi-porting, multiple cache copying, virtual multi-porting and multi-banking. These techniques generally have a limited scalability and are not able to support beyond dual-port, except multi-banking. Multi-banking is claimed to be a scalable multi-porting [18], but the performance can be hampered by bank conflicts (access conflicts) and additional interconnection delays. The crossbar area increases with number of banks. However, the main factor is bank conflict as shown in multi-banked memories of vector processors[15]. While increasing the number of banks can partially alleviate bank conflicts in vector processor, this may not be practical for caches. The more number of banks also implies deeper crossbar and thus longer interconnection latency.

Bank prediction has been proposed as speculation technique for multi-banking cache to reduce the extra latency caused by the crossbar interconnection [13, 16]. Another recently proposed approach based on data-decoupled architecture divides memory streams into multiple independent sub-streams with the help of prediction mechanism before they enter the reservation stations [4, 5, 6]. Partitioned memory-reference instructions are then fed into separate memory pipelines, each of which is connected to a small data-cache, called, access region cache (ARC) [14]. To avoid an interconnection network in the critical path, memory references are partitioned early in the pipeline before their actual addresses are known. The separation of independent memory references, in an ideal situation, facilitates the use of multiple caches with smaller number of ports and thus increases the data bandwidth. This multi-banking

technique holds the key to a low cost cache memory design that can cope with increasing degrees of ILP. Once we divide program's memory space into multiple independent areas, then memory reference instructions can be partitioned based on the access regions they access. Most memory reference instructions access either a fixed location or a set of locations that belong to a single data structure, such as an array or C structure. Therefore, even when it is difficult to predict the exact address of a memory reference, it is still feasible to predict its access region. Providing separate memory pipeline and separate data cache for each access region, multiple memory reference instructions can be served in parallel. Since access region cache is an exclusive partition of the traditional data cache, it does not create a new level of coherency problem.

Future high-performance microprocessors also need design reconsideration due to technology scaling and increasing clock-rates. Recent studies [1] show that as feature sizes shrink and wires become slower relative to the logic, the amount of state that can be accessed in a single clock-cycle will cease to grow. In this regard, small structures may become viable solutions in high clock-rate microprocessors implemented in shrinking technologies. This is the main motivation behind partitioning the data memory stream into multiple access regions and serving them with smaller ARCs. This approach relieves the hardware complexity involved in a large instruction window by splitting memory instructions early in the pipeline and using smaller structures. It allows use of single ported data caches attached to separate instruction windows, putting lesser burden on the bandwidth requirement of cache itself. The network and the control logic for

orchestrating memory access between a large number of reservation stations and cache ports become simpler.

A selection function for mapping memory references to each ARC can affect the data memory bandwidth as conflicts and load balance at each ARC may differ. Previous studies of Access Region are restricted to static partition of data, which show same partition of data regardless of programs' memory access pattern. This static partition may suffer unbalanced load and access conflicts. In this thesis, we consider speculative dynamic schemes that define access regions dynamically to achieve a balanced partition with scattered ARC conflicts. These may improve the performance if they could reduce access conflicts in ARCs and distribute memory references evenly to all ARCs. Thus, one has to address these memory traffic issues such as load balance of memory references and access conflict in designing a data cache using multi-banking scheme. This thesis studies the memory traffic with various methods of mapping memory reference to each ARC, either statically or dynamically, and the corresponding configurations of multiple memory pipelines, each of which may differ in performance and hardware complexity. We first consider the cache bank interleaved scheme that defines ARC by simple bit selection of memory address [15, 18]. We also evaluate address randomization methods [7, 17] to reduce access conflicts. These schemes are incorporated with bank prediction mechanism to reduce the extra latency caused by the crossbar interconnection. Then we consider the interleaved static ARC scheme based on data-decoupled architecture [14]. We further propose and evaluate dynamic approaches for defining Access Region based on program's memory access pattern to achieve a balanced partition with less access

conflicts. The main purpose of this thesis is to provide the base idea in exploring prospects of multi-porting solution via dynamic data partition for ARC by exposing memory traffic/performance tradeoffs.

The rest of this thesis is organized as follows: The next chapter summarizes background and previous related multi-porting solutions. Chapter 3 discusses the proposed designs for reducing memory access conflicts. Chapter 4 describes our experimental approach and the benchmarks used. Chapter 5 presents simulation results and studies memory traffic to show the potential of the proposed design. Conclusions are summarized in Chapter 6.

2. RELATED WORKS

As more aggressive wide-issue processors demand higher memory bandwidth, designing an effective multi-ported data cache has been a topic of active research. Sohi and Franklin [20] predicted that the L1 cache bandwidth will eventually become a performance bottleneck for a multiple-issue processor and proposed a non-blocking, multi-ported data cache design with interleaved banks as a solution. Though Wilson *et al.* [22] argued that adding more ports to the L1 cache could become costly and/or inefficient in terms of space and time, multi-banking still seems to be a practical solution for achieving multi-porting beyond two. One such multi-banking solution is a cache-bank interleaved scheme [17, 18], in which a number of independent single-ported cache banks are employed to get multiple ported data-cache memory. This cache-line addresses are interleaved through the banks and the banks are accessed in parallel. The cache-line address interleaved scheme is a better option in a multi-bank cache system design because it ensures only one bank getting affected by a cache miss. However this interleaving technique suffers from the bank conflicts [9, 18]. Neefs *et al.* [13] reported potential benefit of bank prediction to reduce the bank conflicts. Yoaz *et al.* [24] proposed bank prediction that increases the cache port utilization through balanced scheduling of load instructions toward multiple cache banks. Another method called access-combine technique [18] uses small multi-ported line buffers for each bank. It combines simultaneous accesses going to the same cache-line and avoids the bank conflicts to a certain extent.

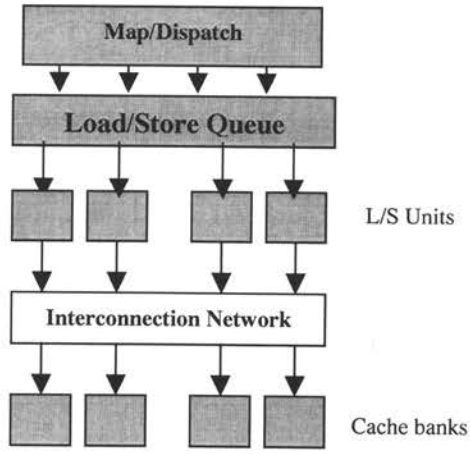


Figure 2.1: A Practical Multi-porting Design - An interleaved bank cache memory pipeline with 4 cache-banks and an interconnection network.

A typical cache-bank interleaved design is shown in Figure 2.1. This system employs an interconnection network (crossbar) to distribute memory references among multiple cache banks from the load/store (L/S) units. This network connects L/S units of the processor to multiple cache banks. If there are n L/S units capable of issuing one memory access per cycle, then the network must have a peak bandwidth of n data per cycle. The area of a crossbar increases super-linearly as the banks and ports increase. This in turn increases delay in the signal path between L/S units and cache banks. In other words, it increases the cache access time. So the advantage of using smaller caches to reduce the access time may be nullified by the extra latency added by an interconnection network. As semiconductor devices shrink and clock-rate becomes faster [1, 16], the latency of an interconnection network will increase to a great extent. As seen already in the design of Alpha 21264 [10], it may become necessary to introduce one or two extra pipeline stages for "in-transit" through the interconnection network. This is due

to the fact that wires become slower relative to the logic when feature sizes shrink. Hence, in future microprocessor designs, the interconnection network in a multi-banking solution poses severe threat as it adds significant cost in terms of latency to the memory-access path. In this context, some recent studies reported potential benefit of bank prediction [13, 24] to reduce the extra latency caused by the interconnection network.

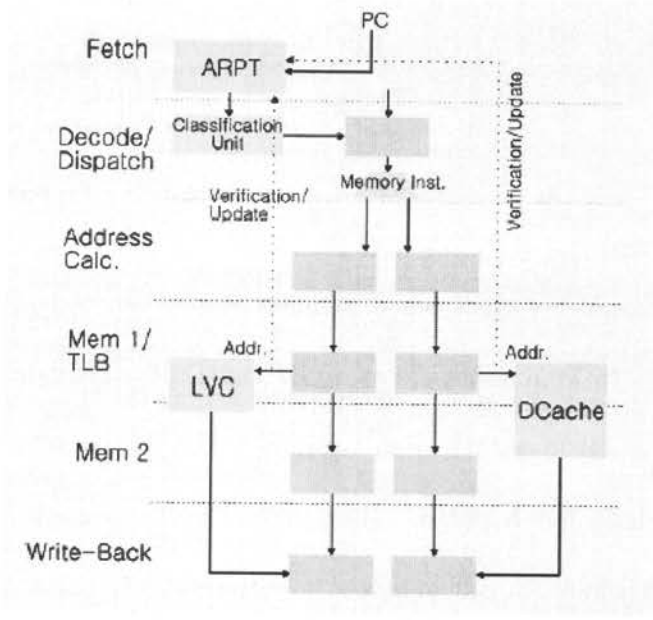


Figure 2.2: The Data-decoupled Architecture - Its memory pipeline with dual memory access queues and two single-ported data caches (stack and non-stack).

The data-decoupled architecture (DDA) [4, 5, 6] is shown in Figure 2.2. It is an alternative solution to achieve higher memory bandwidth without having an interconnection network in the critical path. It divides the data memory stream into two independent streams (stack and non-stack). Each stream is fed to an independent cache.

Based on this partitioning, memory reference instructions are steered to proper memory pipelines before their actual addresses are known [5]. This is done at run-time by using prediction. The stack region contains run-time stack references for local variables while the non-stack region contains references accessing the data segment and the heap area. A small local variable cache (LVC) serves stack references while non-stack references are served by another data cache. The advantage of data decoupling is in decreased hardware complexity - less cache ports requirements, simpler instruction issue logic and no need of an interconnection network in the critical path. But, it does not provide a scalable multi-porting solution beyond two. Lee *et al.* [12] recently proposed an alternative approach based on the concept of this data decoupling. This approach partitions memory references into stack and non-stack streams and serves the stack references by a register file like structure called *stack value file*. Here, the focus is mainly on reducing the demands on first-level cache and data bus traffic.

Figure 2.3 shows a memory pipeline design of the recently proposed interleaved static access region cache [14], which is complementary approach as scalable multi-porting based on the concept of data decoupling. The memory pipeline divides data memory stream into two independent streams (stack and non-stack) first. Then, each of these streams is further partitioned into multiple sub-pipelines (one for each access region), each of which is connected to a small ARC. To avoid the interconnection network in the *critical* path, for each memory reference its ARC is predicted early in the pipeline *before* actual addresses of the data they access are known [5, 6, 14]. As a static way of partitioning memory references, cache-line interleaving is employed. A portion of

the effective address (bits next to *byte offset*) is utilized as an ARC number. This cache-line address interleaving scheme is a better option in a multi-bank cache design because it ensures that only one bank gets affected by a cache miss. Also, this address interleaving provides a simple way of validating ARC prediction. However, this cache-line address interleaving is prone to conflicts. Also, allocating the same number of ARCs to stack and non-stack region may lead unbalanced distribution and underutilize the high locality of stack region. Limaye *et al.* [11] recently proposed Parallel Cachelet, which is a kind of dynamic multi-porting cache. This approach partitions memory reference based on the dynamic pattern of programs memory accesses. However, this scheme suffers from broadcasting store instructions to all partitioned caches and replicating data in multiple caches.

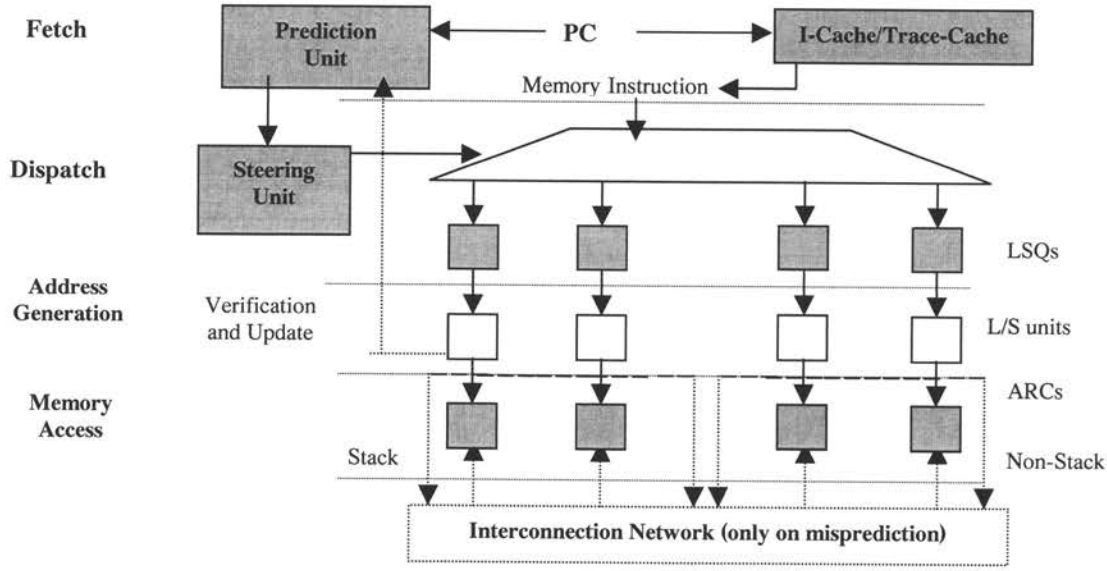


Figure 2.3: A Recently Proposed Interleaved Static ARC Microarchitecture with Multiple Memory Pipelines & Access-regions.

3. SCATTERING ACCESS CONFLICTS

Partitioning of data memory stream into multiple access regions and serving them in parallel with smaller cache (ARCs) requires a selection function for mapping the memory references to each ARC. This function can affect the bandwidth delivered by this multi-cache implementation since it influences the distribution of accesses to each cache. While access regions can be dynamically defined by partitioning either data or memory instructions, in any case, the access-region definition should support two fundamental issues: First, decoded memory access instructions should be partitioned into independent streams before they enter the instruction window. Second, each partitioned stream should contain an adequate amount of workload to justify the scheduling and possible communication overhead. The distribution should attempt to reduce access conflicts in each ARC because they can seriously degrade the delivered performance. To reduce conflict rate of static ARC design, we consider three conflict scattering schemes. The first one we consider is an address randomization method using exclusive-OR based placement function to generate corresponding an ARC number. Others are dynamic schemes that define access regions dynamically depending on the memory access pattern of program. One is a register based dynamic scheme that defines access regions depending on the base register number. The other is rotational scheme that allocate data to ARC by round robin based on the dynamic patterns of concurrent accesses. The first

defines access region by partitioning both data and memory instructions while the later allocates ARC by partitioning memory instructions basically. These schemes may dynamically reallocate the conflicting data from one ARC to another ARC based on program's memory access pattern in order to achieve a good dynamic load distribution and to reduce access conflicts.

3.1. Randomized Access Region Cache

Randomized ARC design uses exclusive-OR function on a portion of the effective address to allocate ARC number to each memory reference. As the effective address of a ready memory instruction is calculated, ARC number is defined through a randomization function. This portion does not change during the address translation at the Translation Look aside Buffer (TLB). Hence, we can verify this prediction (ARCP) in parallel with TLB access. This prediction validation can be done in parallel with the ARC access. Final cache hit or miss signal can be combined with correct prediction or misprediction signal. We consider two different types of randomized mapping functions. First, we use a simple bit wise exclusive-OR of the lowest two tag bits and two ARC bits next to byte offset in the interleaved static ARC design. We also use the irreducible polynomial 5 mapping to generate skewed ARC number [7, 17]. Rest of the design is same as the interleaved static ARC design.

3.2. Dynamic Access Region Cache

3.2.1. Register based dynamic ARC

The access region can be defined based on base register number of a memory instruction. We use profiled data from eight different SPEC CPU2000 benchmark programs [21] and formed different groups of base registers in memory reference instructions. The access regions are initially allocated based on those groups so as to achieve adequate amount of workload in each ARC (in terms of references). Figure 3.1 shows a memory pipeline design of this register based dynamic access region cache.

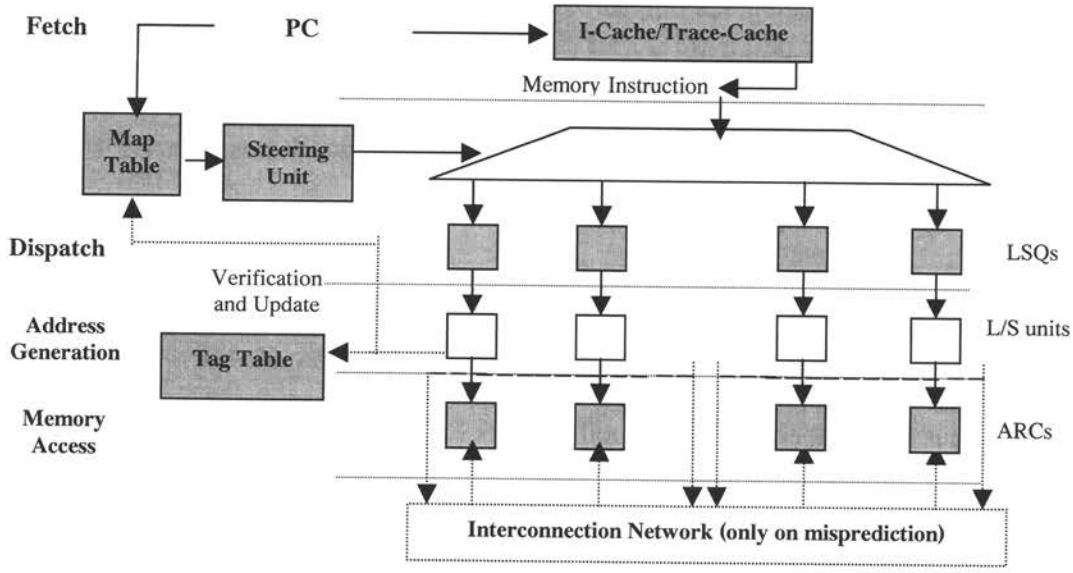


Figure 3.1: Proposed Dynamic Microarchitecture with Multiple Memory Pipelines & Access Region Caches.

We use a base register number in a memory reference instruction as initial prediction information once the instruction is decoded in the decode stage. The base register number is translated into an access region number by using a small (32 entry) table indexed by the base register. Because it is possible to have the same effective address generated by different memory reference instructions using different base registers and to handle redirection of conflicting data we devise a dynamic map table, which stores the mapping between two different base registers generating same effective address. This table is indexed by a portion of instruction's PC in the decode stage as shown in Figure 3.2. If the appropriate entry is valid, then the target ARC number from that entry is considered as prediction information. Otherwise, the translated ARC number of the base register number is considered as prediction information. If an entry in this table is flagged as a conflicting entry, a random value by a global rotating counter is taken as a predicted ARC number and then the conflict bit is reset. This table is updated during the prediction verification process if there is a misprediction or a redirection. Each entry has two bit target ARC, one valid bit and one conflict bit. Note that the valid bit is required to validate an entry. All entries are made invalid during initialization.

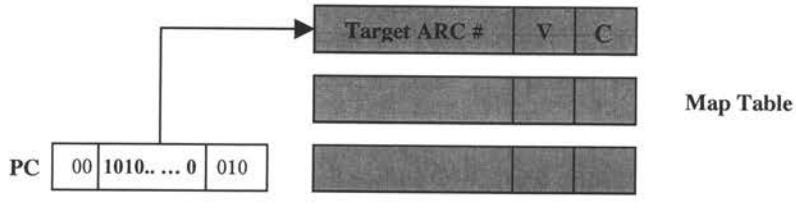


Figure 3.2: Dynamic Map Table - Each entry has 3 fields. It is indexed by a portion of instruction's PC.

3.2.2. Instruction based rotational ARC

In rotational scheme, the allocating of memory accesses to ARC is not based on the data address but on the dynamic pattern of programs memory accesses. When a load/store instruction is executed the first time it is assigned to an ARC by round robin. Initially, concurrently executed memory instructions are assigned to different ARC to avoid access conflicts. The ARC number can be stored in a separate map table as in register based ARC or possibly placed in the instruction cache along with the memory instruction. When accessing the data cache, the attached ARC number of a memory instruction selects the ARC. However, because of the dynamic behavior of program, the group of loads that may execute concurrently may vary during program execution. Consequently subsequent execution of the same memory instruction may result another ARC conflict later even if all the memory instructions have been previously assigned to ARCs. Hence we consider that possible reallocation of such memory instructions that result future conflict. To able to this, subsequent execution of the same load can result in it being dynamically redirected to another ARC by round robin. In this case, the same old data in the previous ARC should be invalidated to prevent any data being in multiple ARCs as in the register based dynamic ARC design. The main purpose is to reduce access conflicts by distributing concurrent accesses to different ARCs by partitioning of memory access instructions rather than data.

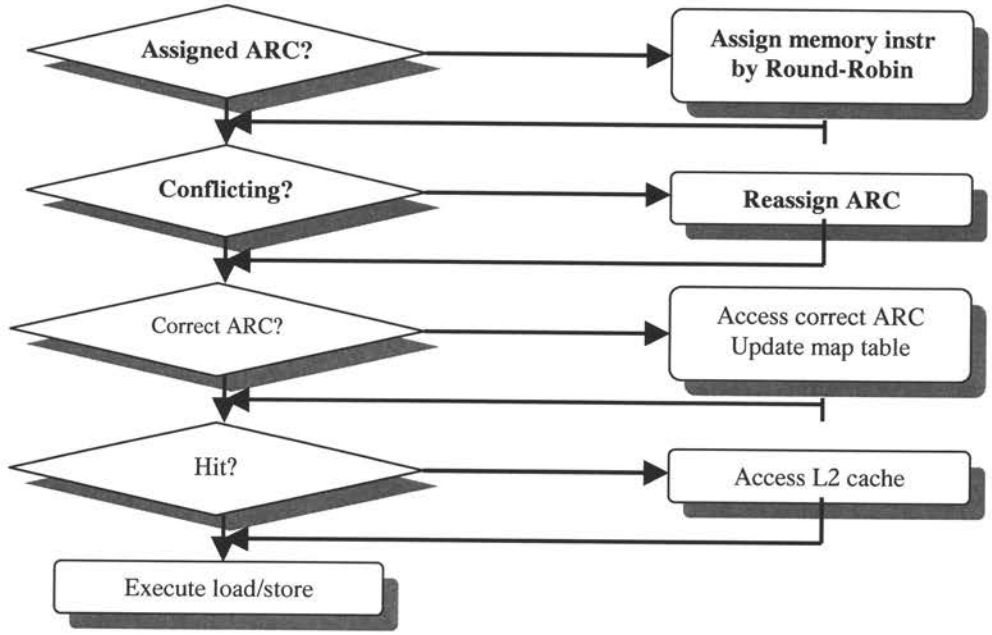


Figure 3.3: Assignment and Access Process of Proposed Rotational Scheme.

3.2.3. ARC reallocation and prediction verification in dynamic schemes

In order to reduce access conflicts, the conflicting references are dynamically reallocated to different ARCs. During the memory access stage if a reference is found to create a conflict, its entry in the map table is flagged. Next time, this flagged reference is redirected to an ARC indicated by a global rotating counter. This method in an ideal case ensures an access distribution with minimal conflicts.

It is necessary to verify the prediction information before an instruction initiates a memory access or alters contents in the memory. It ensures a single updated copy of the data in any ARC and prevents data inconsistencies among them. We devise a verification method for dynamic ARC schemes since we do not assign data into ARC based on the

effective address. We use a central table called, tag table that stores tag bits of an effective address generated by a memory reference instruction. The table keeps tags of all ARCs and is divided into 4 groups (the number of ARCs) such that each group belongs to an ARC. Whenever there is an ARC-miss, a new tag is allocated into the tag table. To store tag bits of four ARCs of size 16KB each, (an equivalent of 64KB data cache) with 32 bytes line size, the tag table has 2K entries. The tag table is accessed to verify the correct ARC of each memory reference instruction. Once the effective address of a memory reference instruction is calculated, tag bits of the address generated are compared with tag bits from a particular entry in the tag table. Tag table is indexed by index bits (bits next to byte offset field) of the effective address appended by the predicted ARC number. The tag table can be accessed in parallel with the TLB access. This prediction can be verified after the TLB is accessed for address translation and before actual data cache access takes place.

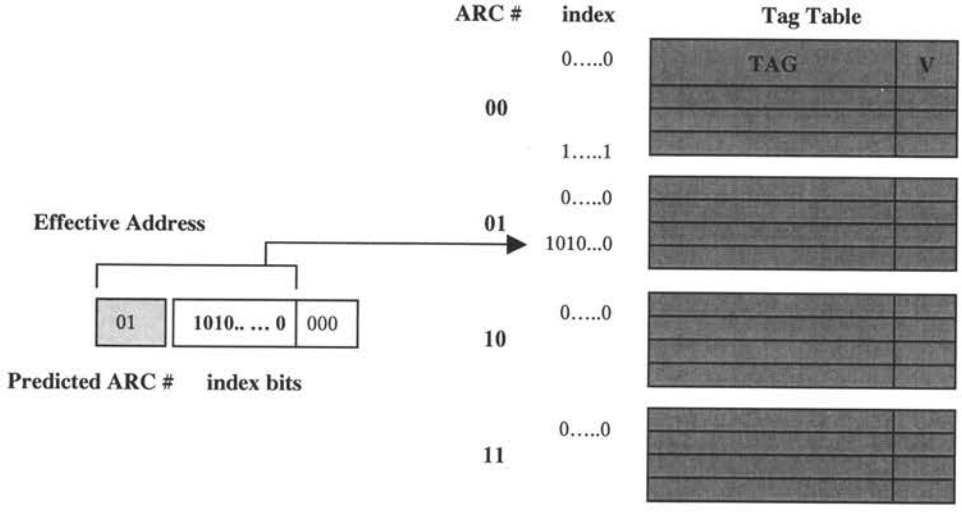


Figure 3.4: Tag Table - Each entry has 2 fields, which is indexed by index bits from the effective address concatenated by predicted ARC number.

As shown in Figure 3.4, each entry in the tag table has tag bits and a valid bit. All entries are invalidated during initialization. When the tag bit of the indexed entry does not match with tag bits of the address generated by memory reference instruction, it is assumed to be a misprediction. In that case with 4 ARCs, remaining 3 entries are searched for a tag match by changing the predicted ARC number portion of the tag table index. If any match is found, then that particular ARC is the correct ARC. This is a true misprediction. Accordingly, the dynamic map table is updated. The correct ARC number is stored as the target ARC, and the valid bit sets flag. Mispredicted memory instructions are re-routed to correct ARCs through an interconnect network. Hence, additional latency due to re-routing is imposed to the wrongly predicted memory instructions as a misprediction penalty and thereby stalling next depending instructions. This in a sense gives higher priority to the mispredicted memory instructions to access the memory ports

of correct ARCs after re-routing. Note that this interconnection network is not in the critical path of memory instructions all the time. If the ARCP were fairly accurate, only a small number of incorrectly predicted memory instructions would have extra routing latency. If match is not found, then new tag is allocated in an entry corresponding to the predicted ARC number as a true cache miss not a misprediction. When a conflicting reference is redirected, its old entry in the tag table is invalidated. A new entry is allocated for this reference in a location that corresponds to its new ARC. The target ARC stores this new ARC number and the conflict flag is set in the map table. Note that, new entries are allocated only in case of a cache miss. Since, the data cache miss rate is very low, the number of write-accesses to this tag table are very small. Note that, access serialization is required in case more than one references generating same physical address are to be allocated simultaneously.

4. EXPERIMENTAL APPROACH

4.1. Simulator Specifications

We have used a cycle-accurate execution driven simulator derived from the *Sim-Outorder* simulator in the Simplescalar toolset [2]. The machine model used is a superscalar processor that supports an out-of-order issue and execution based on the register update unit (RUU) [19]. The processor pipeline consists of the following stages: fetch, dispatch (decode and register renaming), issue, execution, write back and commit. The processor's memory system employs LSQ. Store value is placed in the queue if the store is speculative and is actually written to the cache at the commit stage. For stores, address computation is decoupled from the cache access. Load is dispatched to the memory system when its effective address is computed and addresses of all previous stores are known. The load may be satisfied either by the memory system or by earlier store value residing in the LSQ. In latter case, store-to-load forwarding is done in one cycle, which is an optimal scenario. We modified the basic processor pipeline to incorporate our designs of multiple memory pipelines and ARCs.

To evaluate our proposed approach as emerging trend towards aggressive ILP exploitation through multiple instruction issue, a superscalar processor model that can issue up to 16 instructions per cycle is used. We have employed an ideal front-end for the processor model in our study such as a perfect instruction cache with a perfect branch predictor, in order to assert a maximum data-bandwidth demand on the memory system.

In addition, adequate resources are provided to the instruction-processing phase so as to highlight the effect of bandwidth in the data supply. The parameters we assumed for the base model are summarized in Table 4.1.

Table 4.1: Base Machine Model Configuration.

Issue/decode/ commit width	16
ROB/LSQ size	256/128
Registers	32 GPRs/32 FPRs
Functional Units	16 Int. + 16 FP ALUs 4 Int. + 4 FP Mult./Div. Units
L1 D-Cache/ Hit time	Direct-mapped 64 KB/ 3 cycles. Direct-mapped 4 – 16KB/ 2 cycles.
L2 D-Cache / Hit time	4 way set assoc. 512 KB/ 12 cycles
Memory latency	50 cycles
I Cache/ Hit time	Perfect cache/ 1 cycle
Branch prediction	Perfect

We compare the proposed dynamic designs having 4 ARCs with the ideal base machine model and previously proposed schemes including the cache bank interleaved scheme with and without bank prediction, and the interleaved static ARC scheme. Each bank/ARC in all design schemes is single ported and of size 16KB. The cache-line (block) size is varied between 8, 16, and 32 bytes for all L1 data caches. All caches are assumed to be non-blocking. In our proposed designs as well as all other multi-bank schemes, we assume L1 data cache access latency to be 2 cycles, while 3-cycle latency is assumed for ideal schemes. However, L2 cache access latency and the main memory access latency are same as that in an ideal base machine model. We assumed higher

access latency for ideal multi-ported L1 data cache than that of an ARC or a cache-bank because of its larger size as well as large number of ports. All multi-porting schemes are simulated with interconnection network latency of 1 cycle except ideal schemes. The misprediction penalty in our proposed design is assumed to be the latency for going through the crossbar network to re-route wrongly predicted memory references. As a result, the dependent instructions are delayed and reside in their respective reservation stations.

The number of the tag table entries in the proposed dynamic ARC designs is 2K for 32B, 4K for 16B, and 8K for 8B line size. It is indexed by 9, 10, and 11 for 32B, 16B, and 8B line sizes respectively, bits next to byte offset from the effective address concatenated by 2 bit predicted ARC number. Each entry consists of tag bits and a valid bit. The dynamic map table is a 1K-entry table and each entry has a 2-bit source & target ARC number, a valid and a conflict bit. It is indexed by 10 bits of instruction's PC (above least-significant bits).

4.2. Benchmark Programs

The simulation workloads are eight integer benchmark programs from the SPEC CPU2000 benchmark suite [21] to assess performance of the proposed solution. All of the programs are pre-compiled little-endian PISA SPEC CPU2000 binaries. Each benchmark is simulated to its one billion instruction with input files from train input set. Table 4.2

lists these benchmark programs along with description and percentage of memory reference instructions in each of them.

Table 4.2: SPEC CPU2000 Integer Benchmarks.

Benchmark	Description	Loads (%)	Stores (%)
256.bzip	Compression	38.7	13.3
176.gcc	C Compiler	28.8	13.0
164.gzip	Compression	23.6	10.6
181.mcf	Combinatorial Optimization	34.8	13.2
197.parser	Word Processing	38.6	14.1
253.perl	PERL Language	33.3	15.6
255.vortex	Object-oriented Database	33.6	21.4
175.vpr	FPGA Placement and Routing	35.0	10.9

5. PERFORMANCE EVALUATION

This evaluation section provides the simulation results and analyzes them in concern of memory traffic for the effectiveness and feasibility of reducing conflicts and fair load balancing through address randomization and dynamic partitioning methods. We compare the proposed schemes with other multi-porting schemes introduced in the related works section. For the performance of designs, in addition to IPC, instruction per cycle, we also consider the data bandwidth in terms of committed data bandwidth (CDB). The CDB is defined as the number of committed data references supported by ARCs per cycle. In our simulations, we measured the CDB by dividing total number of loads and stores committed by total number of memory-access cycles. Higher data bandwidth may suggest that data references are more evenly distributed among all ARCs, achieving a fair load balancing in terms of the number of data references going to each ARC. If the one particular cache is getting most of the references, it may become a bottleneck and hamper the performance. Intuitively, it could achieve the effect of multi-ported cache if the data references are evenly distributed among all ARCs. Hence, we measure the number of references going to each independent ARC and then analyze the load balancing. However, average number of references directed to each ARC alone does not accurately reflect the bandwidth supplied by each ARC. For example, consider a case of four bursts, which have 4 simultaneous references to the same ARC, directed to four different ARCs in turn. Load balancing is perfectly even among all ARCs, but real data memory bandwidth remains same as the bandwidth of 1-port cache. To alleviate this problem, we

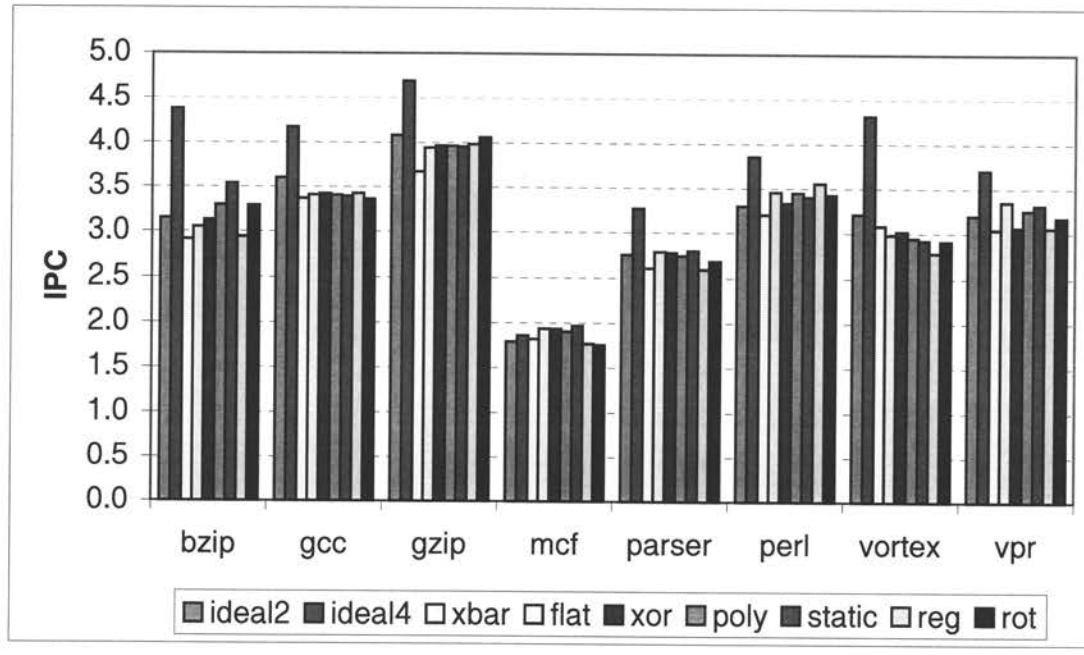
measure and analyze access conflicts. Finally, we consider miss rate and the ARC prediction accuracy because it limits the performance of multi-porting cache designs.

5.1. Simulation Results

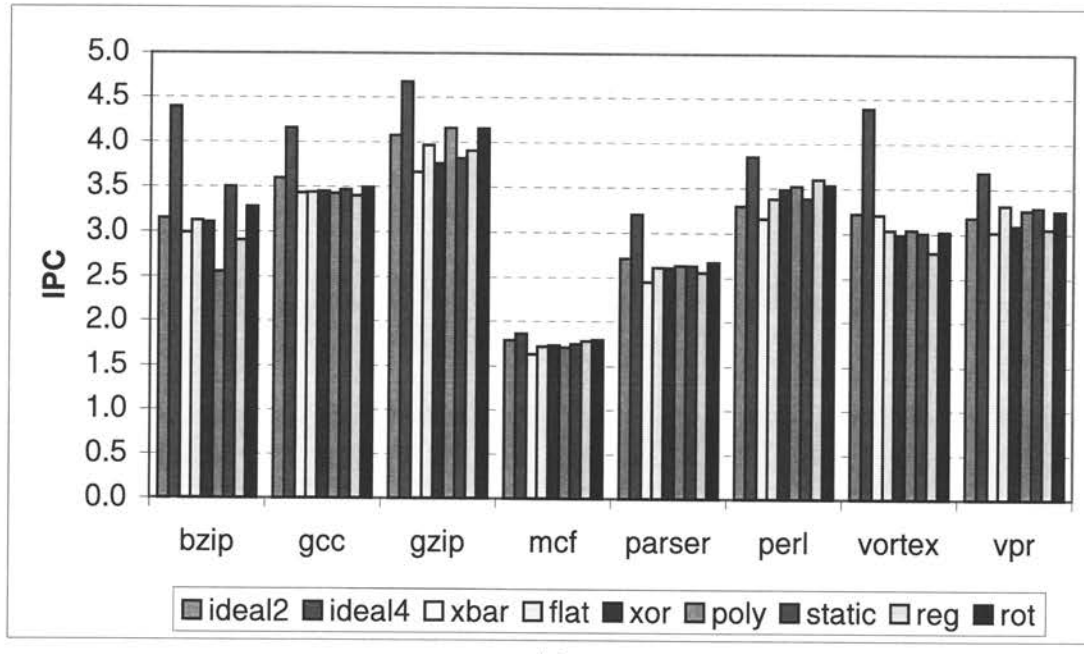
To compare the effectiveness of our designs, we have simulated the ideal base machine model with 2-ported, 4-ported L1 data cache and the cache bank interleaved scheme with a crossbar network. We have also simulated the flat scheme that is the cache-bank interleaved scheme incorporating bank prediction. In this section, we present IPC results of these designs. The misprediction penalty in our proposed designs is assumed to be 1 cycle for going through the crossbar network to re-route wrongly predicted memory references.

Table 5.1: Simulated Designs.

Design	Description
xbar	Typical cache-bank interleaved multi-banking scheme
flat	xbar scheme with incorporate bank prediction
xor	Randomized flat scheme by xoring two lowest tag bits
poly	Randomized flat scheme by irreducible polynomial 5 mapping
static	Cache-line interleaved static ARC based on the DDA
reg	Register based dynamic ARC
rot	Instruction based rotational ARC by round robin

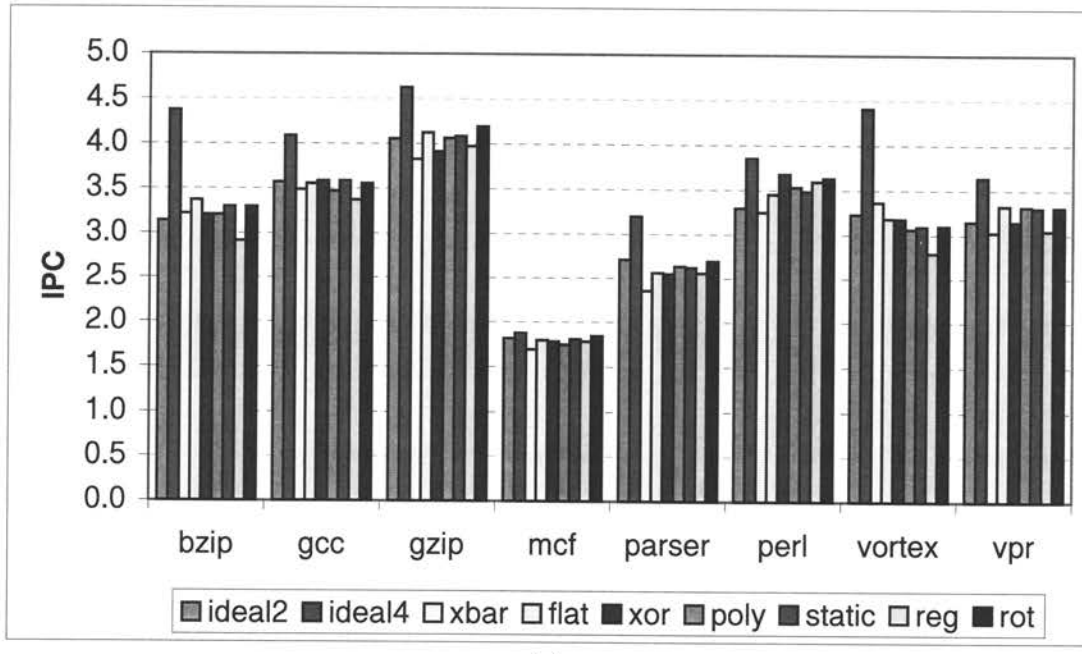


(a)

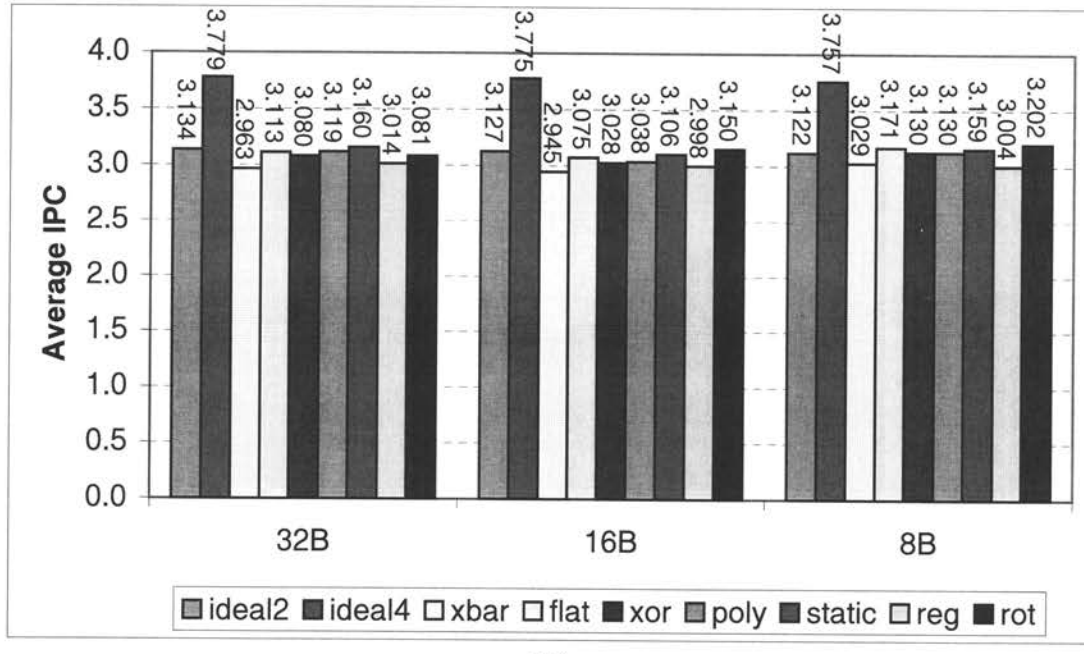


(b)

Figure 5.1: IPC Results - The Performance of various multi-porting schemes with 8 SPEC CPU2000 Integer benchmarks. (a): 32B, (b):16B, (c):8B line size, (d): Average IPC.



(c)



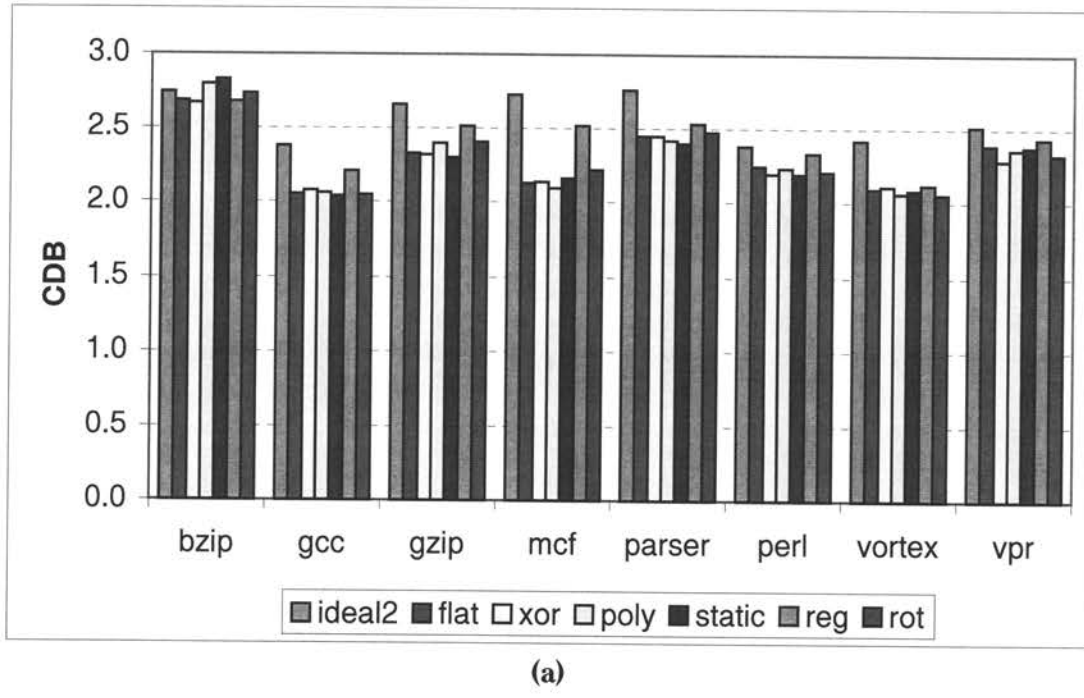
(d)

Figure 5.1: (continued)

Figure 5.1 shows IPC results of various multi-porting designs including proposed schemes with 32 byte, 16 byte, and 8 byte line size. As shown in Figure 5.1 (d), all speculative approaches outperform the interleaved scheme without incorporating bank prediction (xbar). Our proposed rotational scheme achieves the best performance among practical solutions when cache line is 16 bytes or 8 bytes. The interleaved static ARC scheme achieves the highest IPC in 32 byte line size. Even though both xor and polynomial randomized schemes show a little bit of improvement over the flat scheme for some benchmarks, these randomized ARC schemes outperform neither the static ARC nor the rotational dynamic ARC scheme. The static interleaved scheme still achieves good overall performance compared other schemes. Though the register based dynamic ARC solution achieves best performance for some benchmarks, the average IPC is the lowest among speculative designs. For some benchmarks, it even shows lower IPC than the xbar scheme. The rotational ARC scheme performs well for small line size caches.

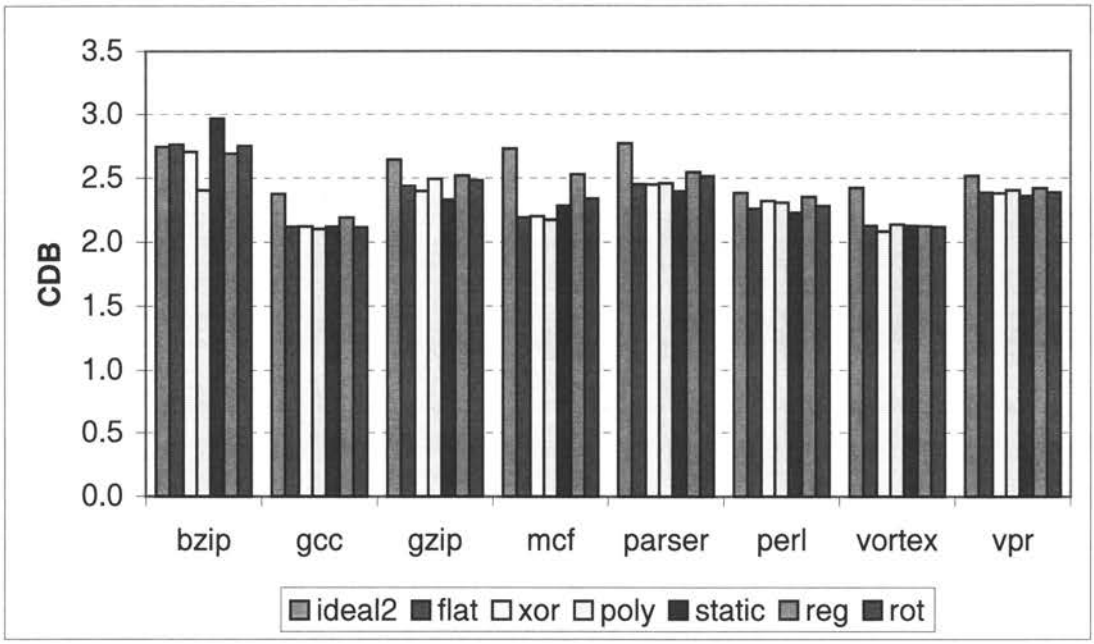
Figure 5.2 shows the committed data bandwidth (CDB) of all speculative schemes including ideal multi-porting designs. As shown in Figure 5.2 (d), on average, the dynamic ARC schemes outperform other practical schemes in terms of CDB metric. In case of the register based dynamic scheme, it does not achieve high IPC despite it achieves the highest performance in terms of CDB. This is because the register based dynamic scheme may remove more access conflicts or may distribute memory reference more evenly to all ARC while other factors, such as miss rate and prediction accuracy, may degraded the overall performance. However, the higher CDB a data cache design

achieves, the better performance it gets as long as cache miss rate and ARC prediction accuracy can keep up with other designs.

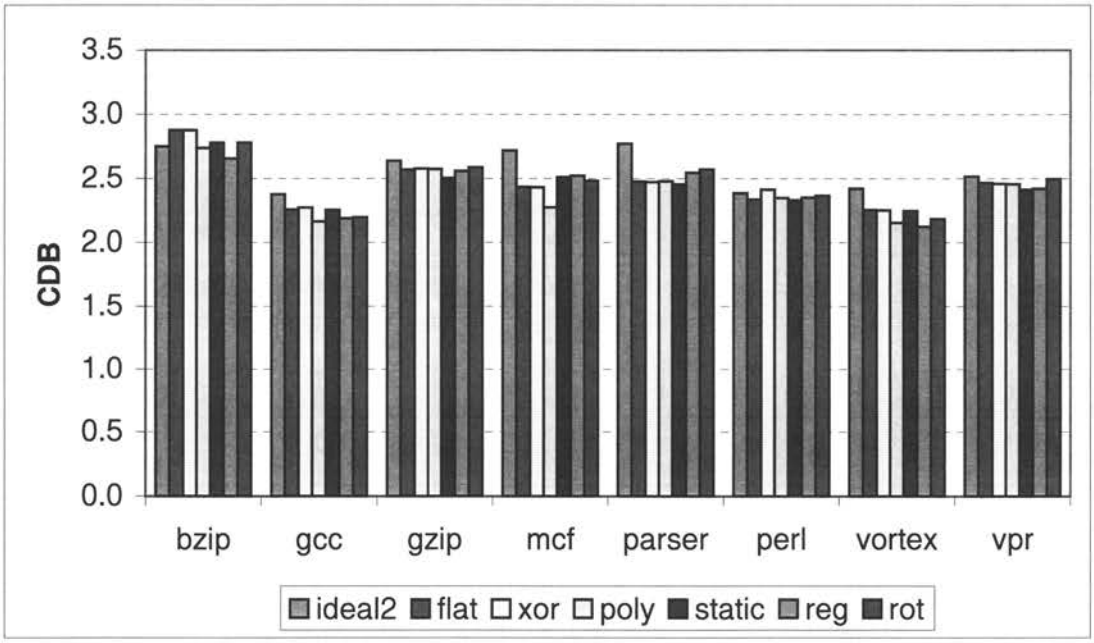


(a)

Figure 5.2: CDB Results - The committed data bandwidth of various multi-porting schemes with 8 SPEC CPU2000 Integer benchmarks. (a): 32B, (b):16B, (c):8B line size, (d): Average CDB.

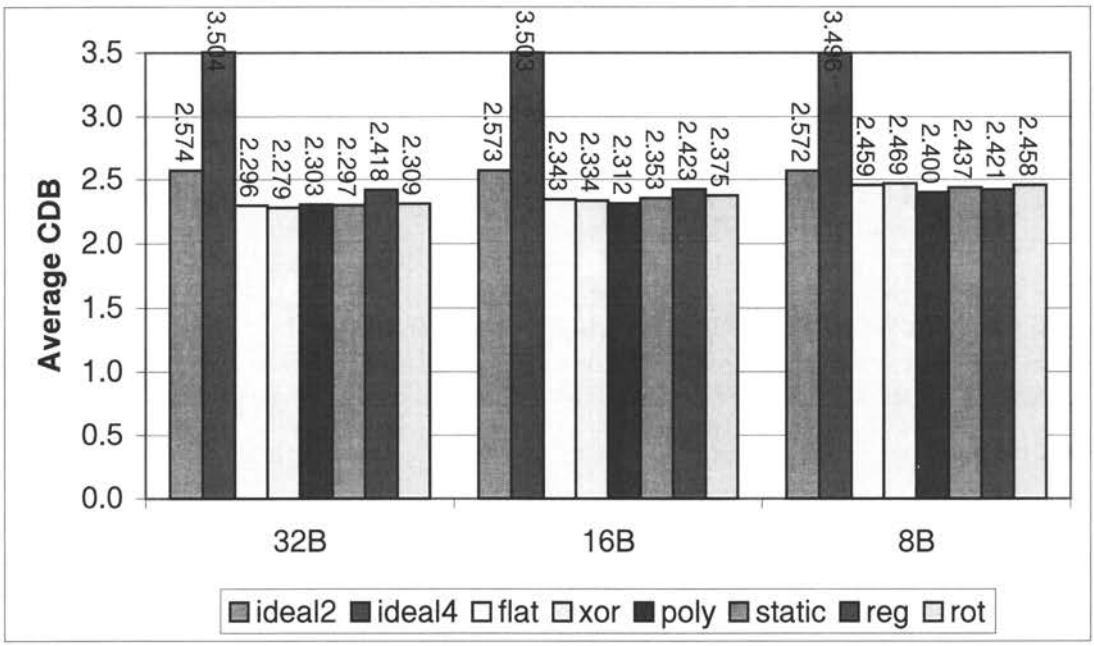


(b)



(c)

Figure 5.2: (continued)

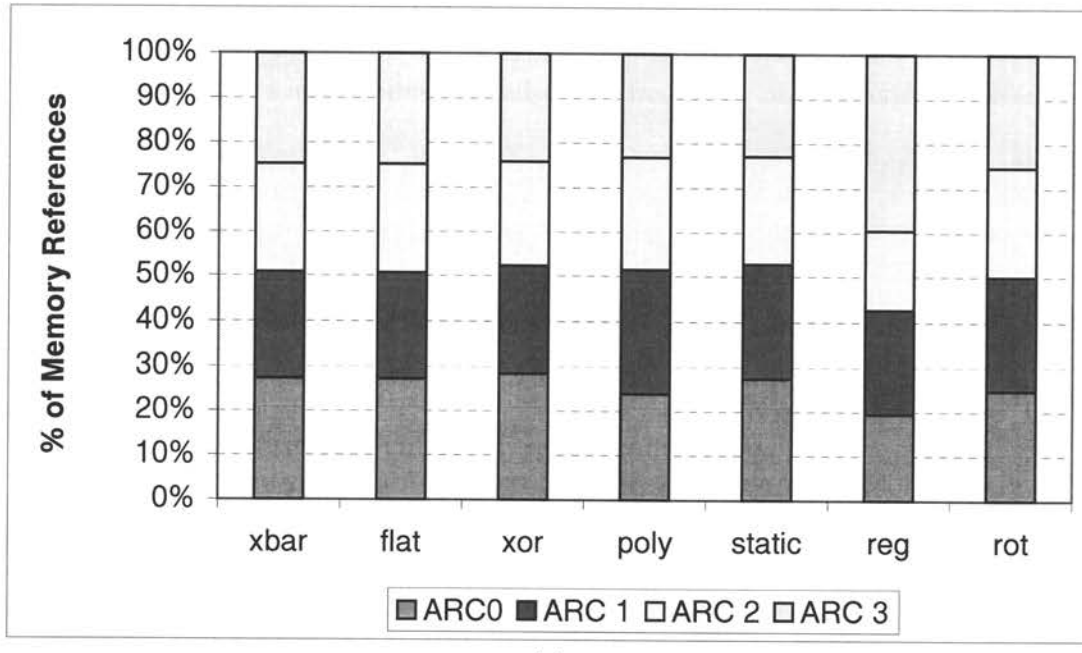


(d)

Figure 5.2: (continued)

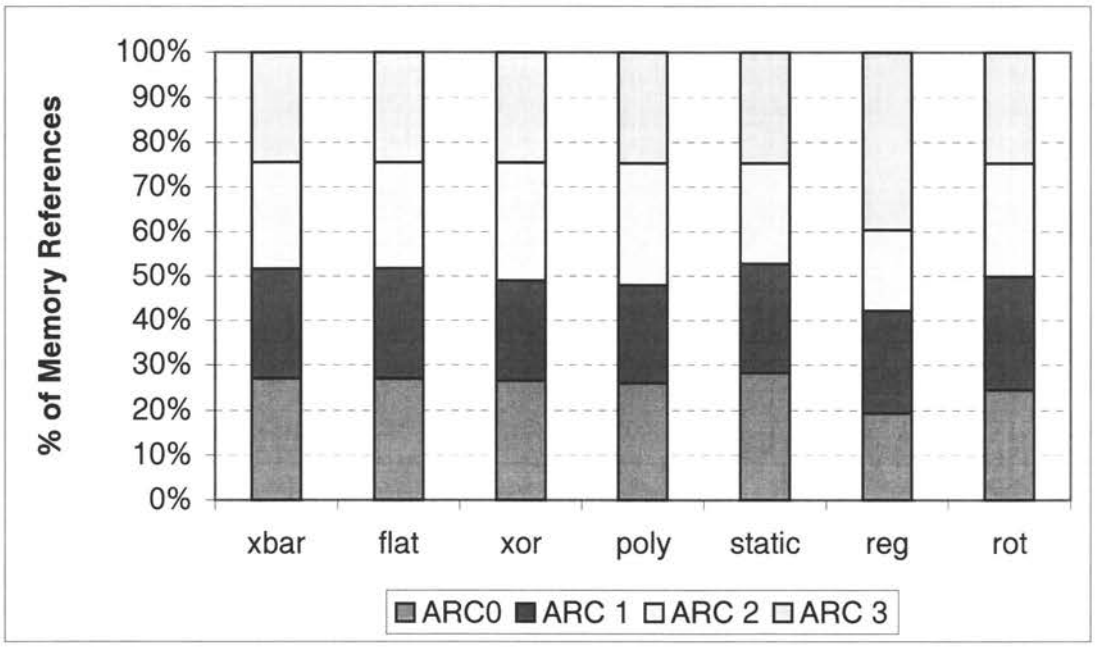
Load balancing of different schemes with various cache line sizes are shown in Figure 5.3. Load balancing of the interleaved multi-bank cache design (xbar) has the same load balancing as the flat scheme because it uses the same portion of the effective address in access allocation. Proper balanced distribution of the memory references among all ARCs is important to achieve higher data-bandwidth in our designs. The load balancing is measured by calculating total number of references going to each ARC. Results shows all schemes achieve compatible load balancing. While different size of cache line affects little on load balancing, different design show different distribution among ARCs. In the register based dynamic ARC scheme, the number of references going to each ARC depends upon how frequently corresponding base register is used in

the memory reference instructions as well as the frequency of redirection of references due to conflicts. In addition to this, it also depends upon the total number of memory instructions that are generating same effective address using different base registers. As shown in Figure 5.3, a certain ARC show high percentage of accesses in the register based dynamic scheme. This is because great portion of memory instructions use base register 30, which is for frame pointer. In contrast to this, the dynamic rotational scheme achieves almost perfect load balancing because it allocate memory reference base on memory instructions.

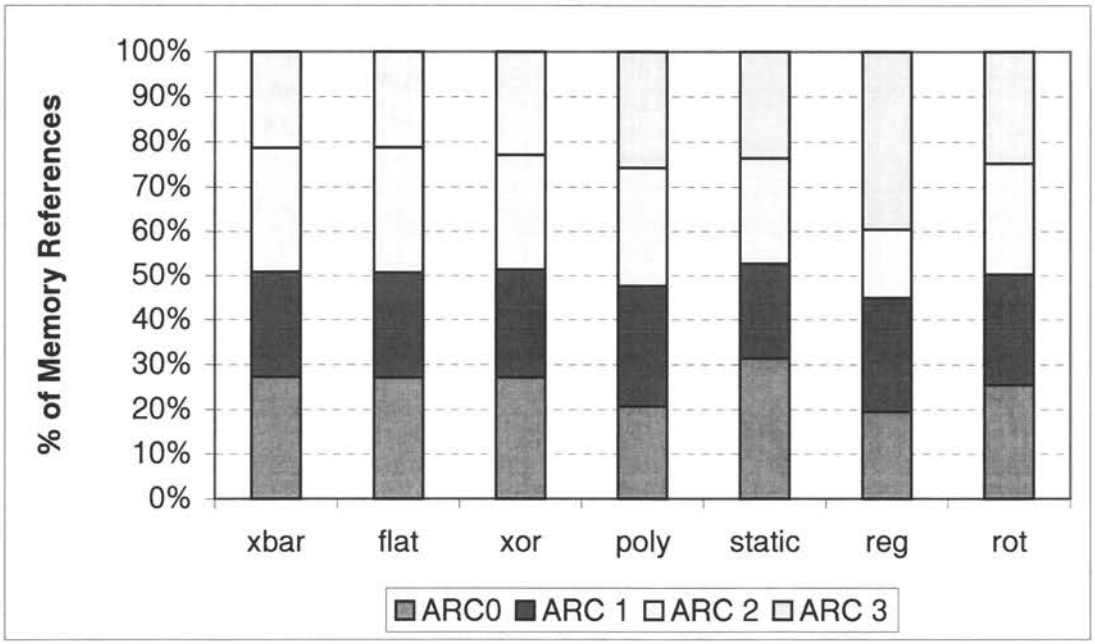


(a)

Figure 5.3: Load Balancing – The distribution of memory references to each ARC in various schemes. (a): 32B, (b):16B, (c):8B cache line size.



(b)



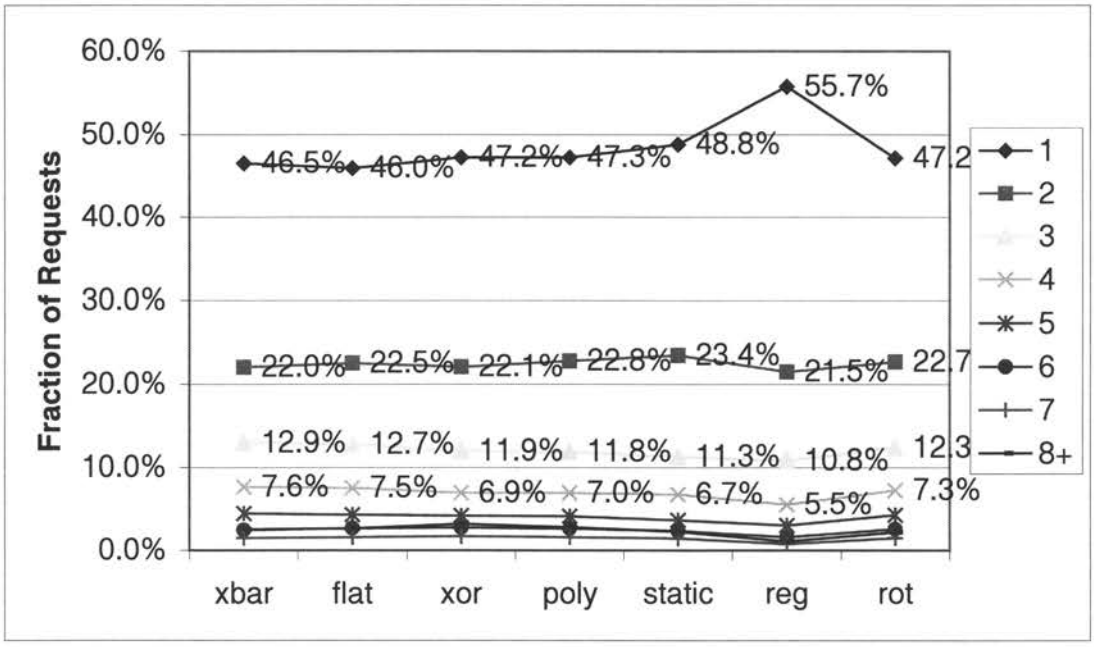
(c)

Figure 5.3: (continued)

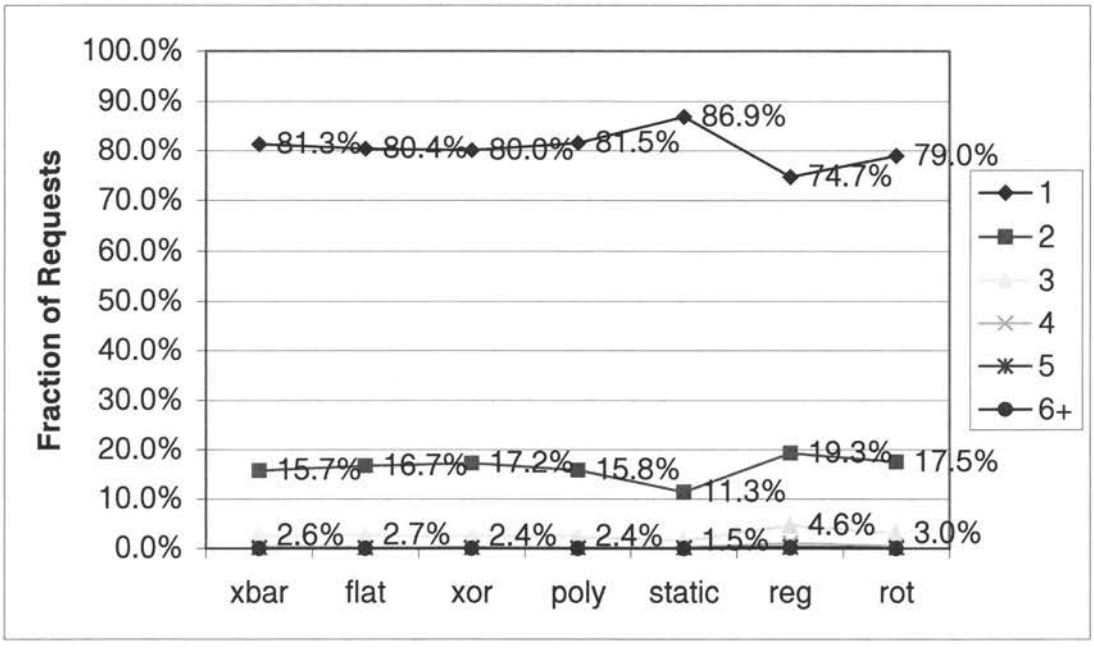
The access conflict of the memory references among ARCs is also as important as load balancing to achieve higher data-bandwidth in our designs. Since load balancing alone does not explain difference between IPC and CBD among various schemes, we also evaluate access conflicts of various schemes in Figure 5.4, 5.5, and 5.6 for 32B, 16B, and 8B line size respectively. Access conflicts occur when two or more memory requests need to access the same ARC simultaneously. ARC conflicts are shown in Figure 5.4 (a), 5.5 (a) and 5.6 (a), which indicate the distribution of the number of simultaneous requests to a bank. These numbers are measured by counting the number of cache requests in the load store queue which are both allocated to the same ARC and ready to execute in each cycle. These statistics of conflicts have been averaged over all ARCs. For instance of the flat scheme with 32B cache line size, 46 % of memory request never experience ARC conflict, 22.5 % conflict with another request, 12.7 % conflict with two other requests and the remaining 18.8 % conflict with three or more other requests. 54 % of memory references experience conflict. As predicted from CDB results, the register based dynamic scheme reduces about 10 % of access conflicts from the flat scheme while other schemes reduce just little bit. As cache line size decreases, access conflicts reduce accordingly in all schemes.

As spatial locality suggests that many consecutive requests can access the same line and thus same ARC, we further analyze the nature of access conflicts. The distribution of the number of distinct lines among conflicting requests is shown in Figure 5.4 (b), 5.5 (b) and 5.6 (b) in order to determine what fraction of conflicts access the same line. For instance of the flat scheme with 32B cache line size, 81 % of pending

memory requests target the same cache line. Confronting this to the previous result that 46 % of requests do not conflict in the flat scheme, 35 % of requests experience conflict with one or more request that access the same line. Only 19 % of requests have access conflicts to the same line in the register based dynamic scheme while 32 % have conflicts to the same line in the dynamic rotational scheme for 32B line size. These conflicts to the same line decrease as the cache line size decreases. 31 %, 16 %, and 29 % for 16B line size and 27 %, 14%, and 26 % for 8B line size experience the same line conflicts in the flat, the register based and the rotational scheme, respectively. As results show, dynamic schemes, especially the register based, reduce many conflicts while other schemes show little reduction over the flat scheme. In addition, different randomizing functions work better for different applications depend on reference pattern. Also, results indicate there are only few cases where four of more distinct lines are involved in an ARC conflict in all schemes.

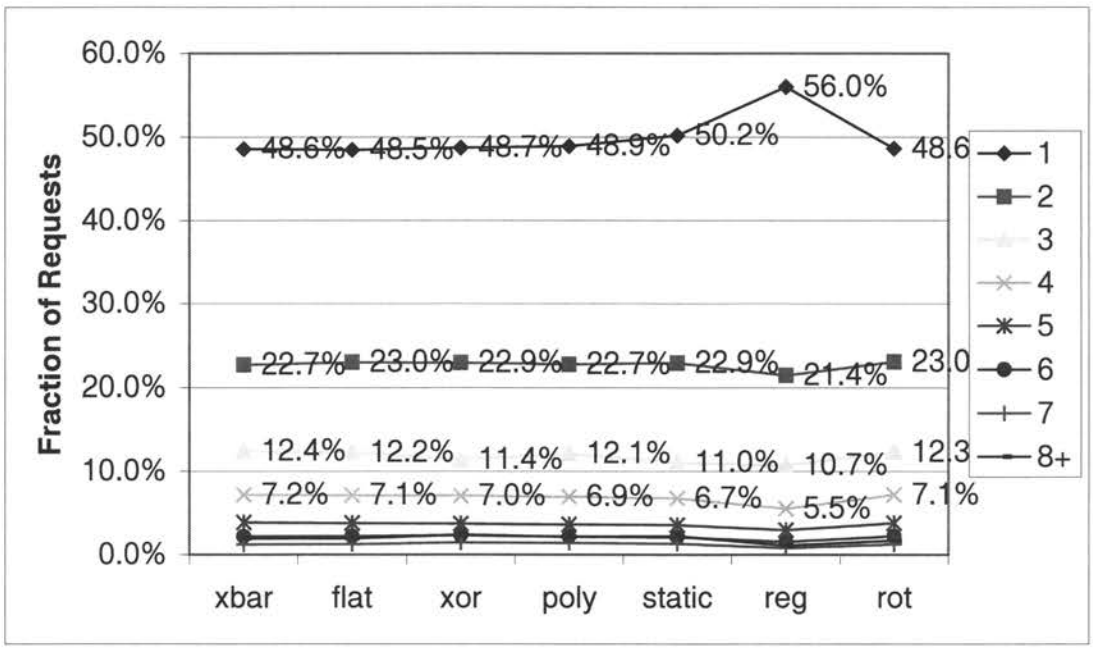


(a)

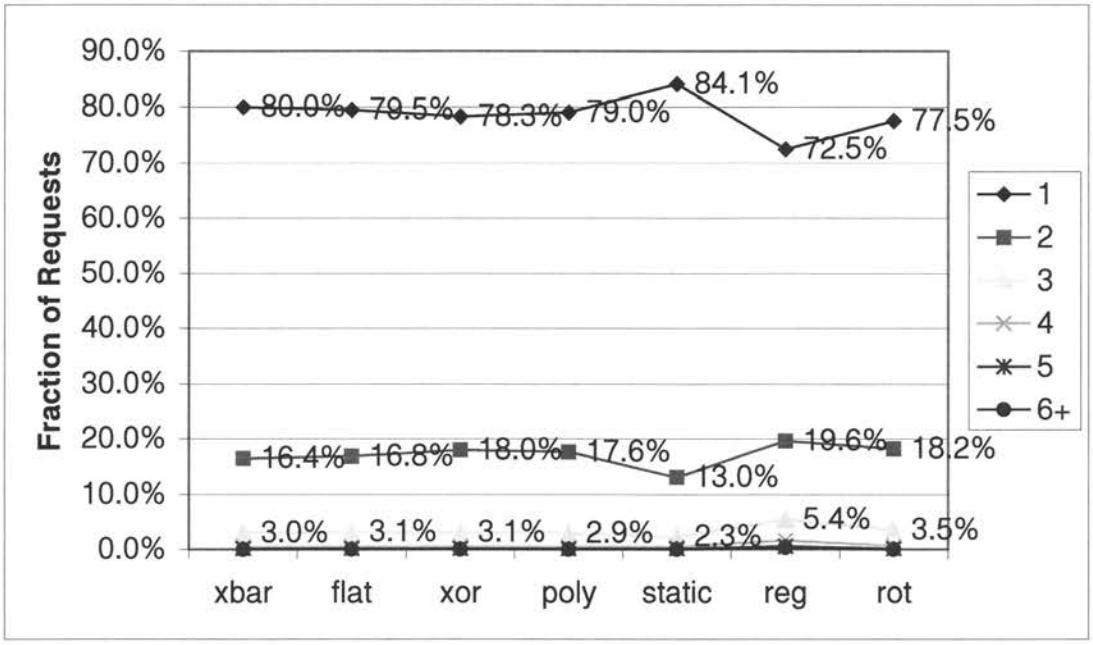


(b)

Figure 5.4: Access Conflict – 32B cache line size. (a): Distribution of number of simultaneous memory requests to a bank. (b): Distribution of number of distinct lines among conflicting memory requests.

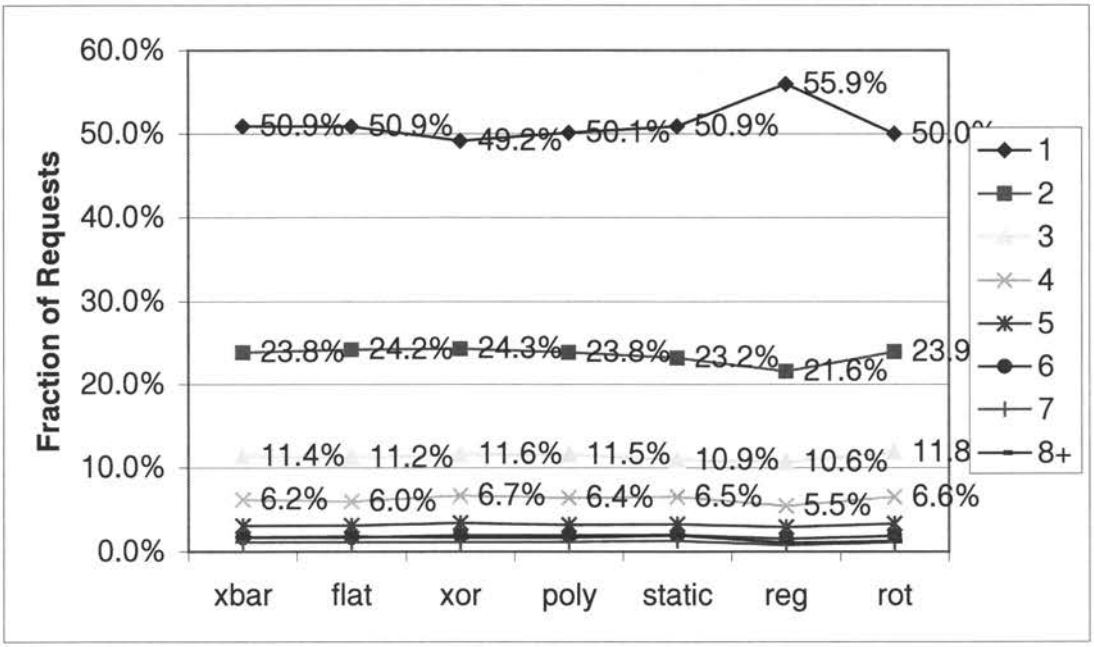


(a)

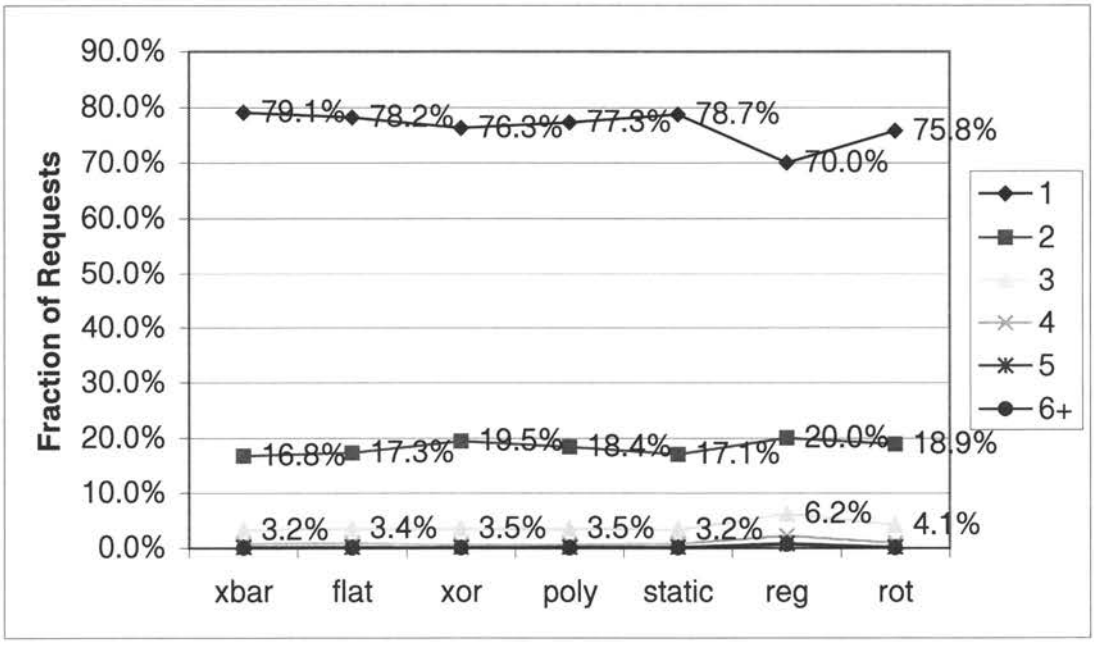


(b)

Figure 5.5: Access Conflict – 16B cache line size. (a): Distribution of number of simultaneous memory requests to a bank. (b): Distribution of number of distinct lines among conflicting memory requests.



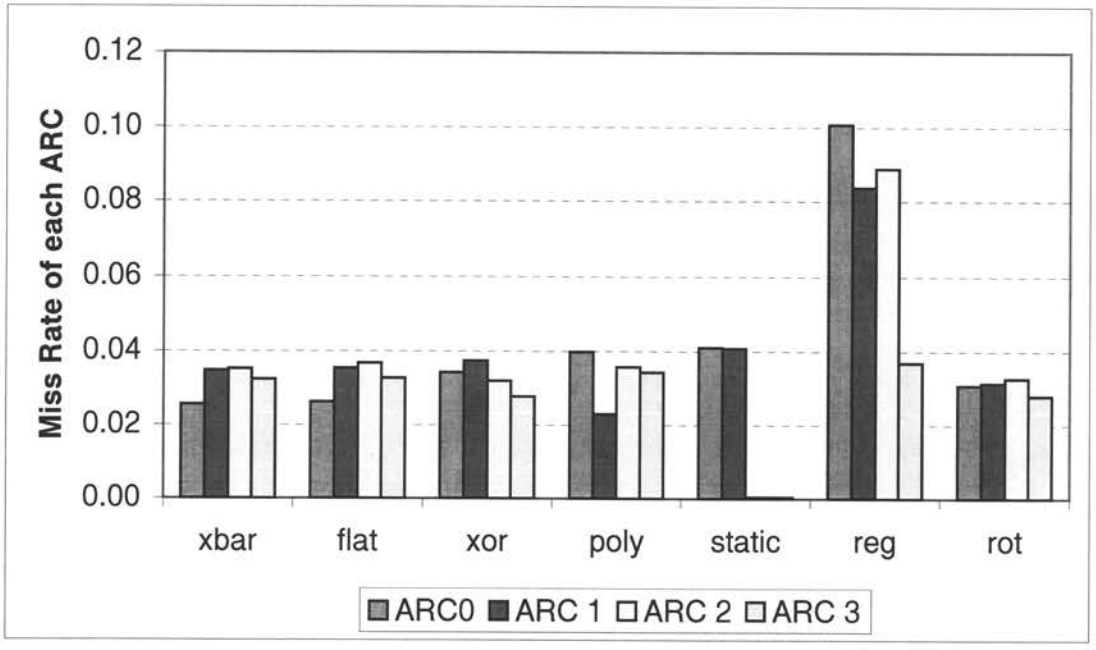
(a)



(b)

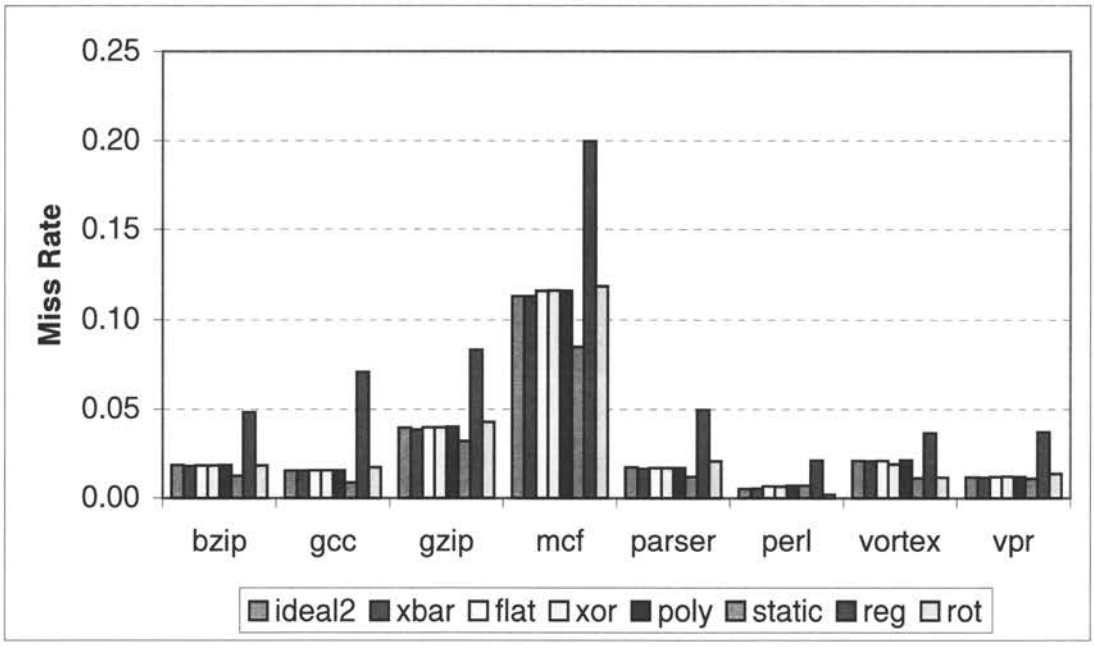
Figure 5.6: Access Conflict – 8B cache line size. (a): Distribution of number of simultaneous memory requests to a bank. (b): Distribution of number of distinct lines among conflicting memory requests.

Figure 5.7 shows the cache miss rate of various multi-porting designs. The miss rate increases as the cache line size decrease in all schemes as shown in (c). The interleaved static ARC scheme has very low miss rate due to low miss rate of ARC for references to the stack region. ARC 2 and ARC 3, which are for the stack region show miss rate of less than 0.0001 in Figure 5.7 (a). This low miss rate of stack region ARC is because stack accesses have a very high degree of data locality, and space required by local variables is very small [4, 5]. In contrast, the register based dynamic ARC scheme has higher miss rate compared to other schemes.

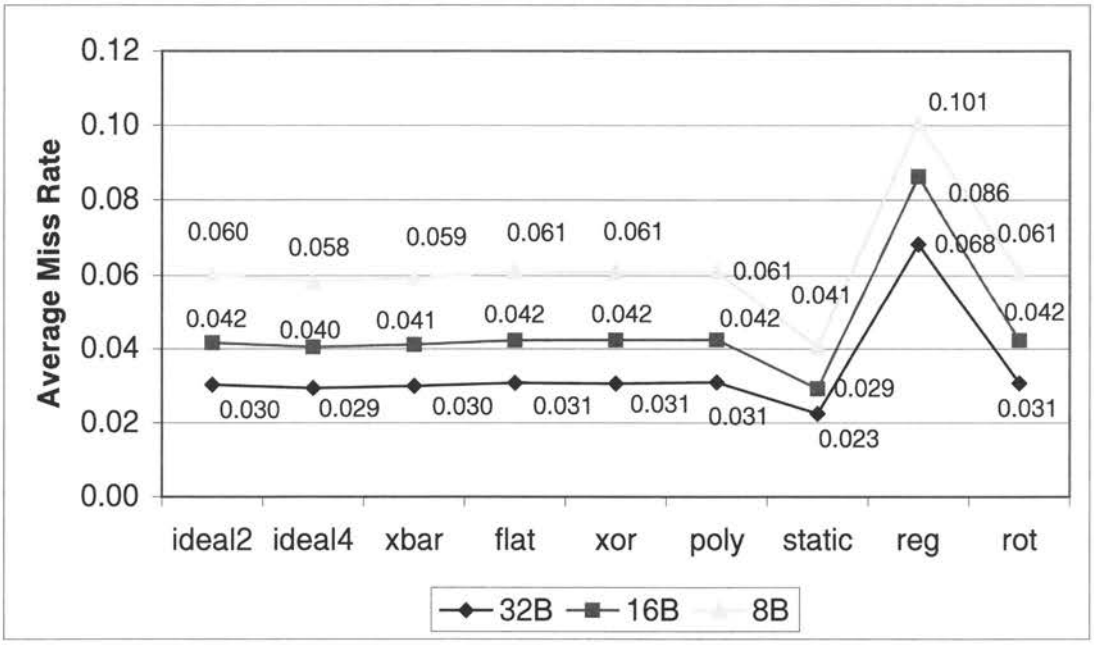


(a)

Figure 5.7: Miss Rate - (a): Average miss rate of each ARC with 32B line size, (b): Average miss rate of ARC in various scheme with 32B line size, (c): Average miss rate of ARC with 32B, 16B and 8B line size.



(b)



(c)

Figure 5.7: (continued)

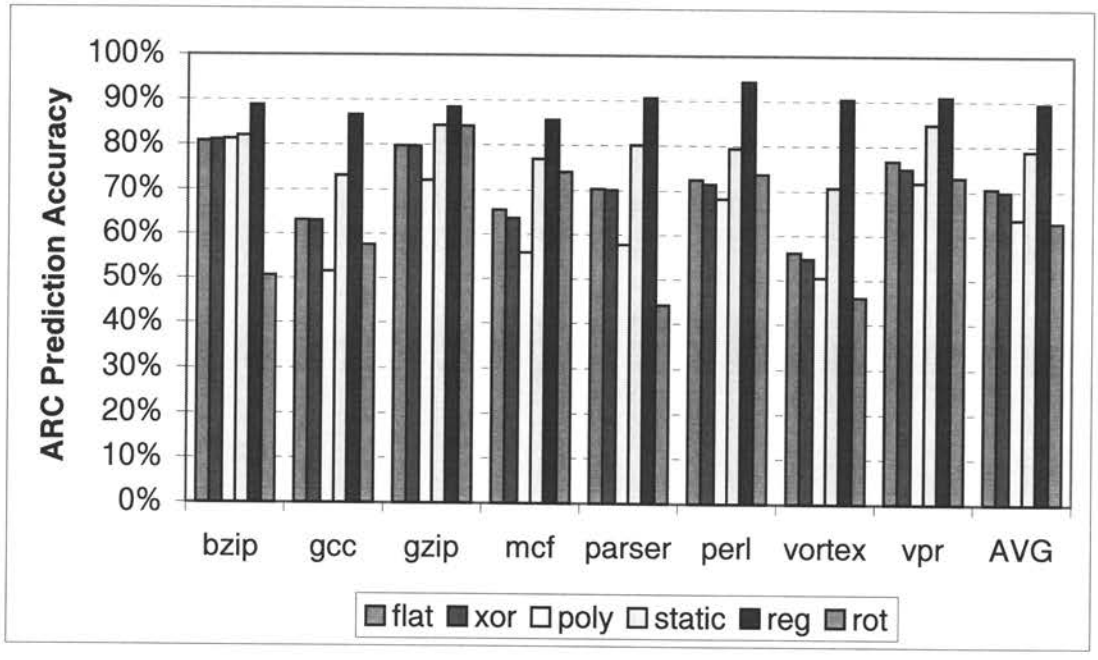


Figure 5.8: Prediction Accuracy- Accuracy of bank prediction and ARC prediction.

Figure 5.8 shows the prediction accuracy of ARC predictor in various designs. Since it's a last value predictor, its prediction accuracy largely depends on the memory reference pattern in benchmark programs. Some benchmarks like vortex have very low prediction accuracy. This is due to a typical reference pattern in which the programs access different ARCs in a loop in steps. The register based dynamic scheme shows very high prediction accuracy. This high accuracy is because we completely know about a register number (hence the ARC number) in the decode stage and the map table takes care of references with the same effective address generated by different memory reference instructions using different base registers or dynamically redirected references. The average prediction accuracy in this design is about 90 %. The dynamic scheme with register based prediction using a map table attempts to exploit a typical reference pattern

seen in most of the benchmarks. These ways of accessing the ARPT and predicting ARCs are baseline approaches. On the other hand, the rotational scheme shows very low prediction accuracy because many memory reference instructions may be reallocated to different ARC according to the phase of benchmark programs.

5.2. Discussions

As compared the results shown in Figure 5.1 (d) and Figure 5.2 (d), the higher data memory bandwidth does not always guarantee the higher performance in terms of IPC for wide issue processors. No certain design consistently outperforms other scheme overall because any scheme can not always achieve the best performance in terms of all of performance factors such as access conflict, load balancing, miss rate and prediction accuracy. In terms of load balancing, the rotational scheme achieves almost perfect distribution to each ARC as shown in Figure 5.3. The register based dynamic scheme has the least conflict in each ARC. It also has the least conflicts to access the same cache line while the static interleaved scheme shows the most. This is due to bursty characteristic of stack region. In regard to the cache miss rate, the interleaved static scheme achieves the lowest because of the high degree of data locality in stack region while the register based dynamic scheme shows the highest miss rate because of same effective address of different memory instructions with different base registers. As the cache line size decreases, conflicts are reduced while the miss rate increases in all schemes. In addition,

the randomization does not show much impact over the flat scheme though different randomizing functions work for different applications.

In conclusion, a selection function that maps memory reference to each ARC can affect the conflict rate in the multi-cache implementation since it influences the distribution of accesses to each cache. Proposed dynamic scheme that attempts to dynamically distribute memory references among different ARCs increases the data bandwidth. The register based dynamic scheme outperforms other schemes in terms of access conflict as well as prediction accuracy while the rotational scheme outperforms others in terms of load balancing. Resolving access conflict along load balancing is important to achieve higher data-bandwidth. Also, higher prediction accuracy can improve performance. Even though only wrongly predicted memory instructions have to pay additional re-routing penalty in all of ARC designs while all memory instructions have additional network latency in the interleaved multi-bank design (xbar), this advantage is diminished in the static designs due to low prediction accuracy. Note that predicting to a single fixed ARC can achieve a high prediction success rate at the cost of negating the very purpose of multi-porting. The main reason for dynamic data partitioning is to provide sufficient data memory bandwidth at fast latency in wide issue processor. However, increasing data bandwidth alone does not always results in performance improvement. Keeping the cache miss rate low is as important as sufficient memory bandwidth to achieve higher performance.

6. CONCLUSIONS

As the trend towards exploiting higher levels of instruction parallelism through multiple instruction issue continues, more and more demand will be placed on the data memory. With increasing number of the data needed from a cache per cycle, current single- and dual-ported cache implementations bounds to become a major bottleneck for future processors. A recently proposed Access Region Cache can become an attractive scalable multi-porting solution. A selection function for mapping memory references to each ARC can affect the data memory bandwidth as conflicts and the number of accesses at each ARC may differ. In this thesis, six different approaches of distributing memory references to ARC are considered to study memory traffic and analyze the effects of distributing memory references to ARC designs. The first two approaches are previously proposed designs, cache-bank interleaved scheme with bank prediction and interleaved static ARC scheme based on the data decoupling architecture. Second, two address randomizing methods are considered to reduce conflicts by using XOR-based placement functions to generate corresponding an ARC number. We further propose two dynamic schemes that define access regions dynamically to achieve a balanced partition with scattered ARC conflicts. The traffic study of various schemes concludes that scattering access conflicts dynamically, redirecting conflicting references dynamically to different ARCs at each cycle rather than balancing overall distribution of memory references, can improve the data memory bandwidth. However, high miss rate and low ARC prediction accuracy may nullify the benefit of higher data bandwidth and can even degrade overall

performance. Hence any allocation scheme that reduces more access conflicts and achieves fair load balancing must keep low cache miss rate and achieve high prediction accuracy in order to achieve high performance gain.

REFERENCES

[1] V.Agrawal, M.S.Hrishikesh, S.Keckler, D.Burger, "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures", *Proceedings of 27th Int'l Symposium on Computer Architecture*, 2000.

[2] T.M.Austin, D.Burger, "The SimpleScalar Tool Set, Version 2.0", *Computer Science Dept. Technical Report, No. 1342*, Univ. of Wisconsin, June 1997.

[3] T.M.Austin, D.Burger, "Billion Transistor Architectures", *IEEE Computer, Vol.30, No 9, pp46-49*, June 1997.

[4] S.Cho, P.C.Yew, G.Lee, "Decoupling Local Variable Accesses in a Wide-issue Superscalar Processor", *Proceedings of 26th Int'l Symposium on Computer Architecture*, May 1999.

[5] S.Cho, P.C.Yew, G.Lee, "Access Region Locality for High-bandwidth Processor memory System Design", *Proceedings of 32nd Annual ACM/IEEE Int'l Symposium on Microarchitecture,* November 1999.

[6] S.Cho, P.C.Yew, G.Lee, "A High-bandwidth Memory Pipeline for Wide Issue Processors", *IEEE Transactions on Computers,* July 2001.

[7] A.Gonzalez, M.Valero, N.Topham, J.M.Parcerisa, "Eliminating Cache Conflict Misses through XOR-Based Placement Functions", *Proceedings of the 1997 Int'l Conference on Supercomputing*, July 1997.

[8] M.Johnson, *Supersclar Microprocessor Design*, Prentice Hall, 1991.

[9] T.Juan, J.J.Navarro, O.Temam, "Data Caches for Superscalar Processors", *Proceedings of Int'l Conference on Supercomputing*, July 1997.

[10] R.E.Kessler, "The Alpha 21264 Microprocessor", *IEEE- MICRO*, 1999.

[11] D.Limaye, R.Rakvic, J.P.Shen, "Parallel Cachelets", *2001 Int'l Conference on Computer Design*, September 2001.

[12] H.S.Lee, M.Smelyanskiy, C.J.Newburn, G.S.Tyson, "Stack Value File: Custom Microarchitecture for the Stack", *Int'l Symposium on High-Performance Computer Architecture,* January 2001.

[13] H.Neefs, H.Vandierendonck, K.DeBosschere, "A Technique for High-bandwidth and Deterministic Low Latency Load/Store Accesses to Multiple Cache Banks", *Int'l Symposium on High-Performance Computer Architecture,* January 2000.

[14] B.S.Thakar, G.Lee, "Access Region Cache: A Multi-porting Solution for Future Wide-Issue Processors", *2001 Int'l Conference on Computer Design*, September 2001.

[15] W.Oed, O.Lange, "On the Effective Bandwidth of Interleaved Memories in Vector Systems", *IEEE Transactions on Computers*, October 1985.

[16] S.Palacharla, N.P.Jouppi, J.E.Smith, "Complexity-Effective Superscalar Processors", *Proceedings of 24th Int'l Symposium on Computer Architecture*, June 1997.

[17] B.R.Rau, "Pseudo-Random Interleaved Memory", *Proceedings of 18th Int'l Symposium on Computer Architecture*, June 1991.

[18] J.A.Rivers, G.S.Tyson, E.S.Davidson, T.M.Austin, "On High-Bandwidth Data Cache Design for Multi-issue Processors", *Proceedings of Micro-30,* December 1997.

[19] G.S.Sohi, "Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers", *IEEE Transactions on Computers*, March 1990.

[20] G.S.Sohi, M.Franklin, "High-Bandwidth Data Memory Systems for Superscalar Processors", *Proceedings of ASPLOS-IV*, April 1991.

[21] The standard Performance Evaluation Corporation, http://www.specbench.org.

[22] K.M.Wilson, K.Olukotun, M.Rosenblum, "Increasing Cache Port Efficiency for Dynamic Superscalar Microprocessors", *Proceedings of 30th Int'l Symposium on Computer Architecture,* May 1996.

[23] K.C.Yeager, "The MIPS R10000 Superscalar Microprocessor", *IEEE Micro,* Volume 16, Number 2, pp. 28-40, April 1996.

[24] A.Yoaz, E.Mattan, R.Ronen, S.Jourden, "Speculation Techniques for improving Load Related Instruction Scheduling", *Proceedings of 26th Int'l Symposium on Computer Architecture*, May 1999.

ACKNOWLEDGMENTS

I would like to thank my major professor Dr. Gyungho Lee for his constant guidance, direction and assistance.

I am grateful to Dr. Akhilesh Tyagi and Dr.Gurpur Prabhu for being on my committee.

I am indebted to Dr. Don Heller for being on my committee.

I would like to thank my parents, my wife and the entire Coover community for helping me directly or indirectly with my graduate program and for all the encouragement.