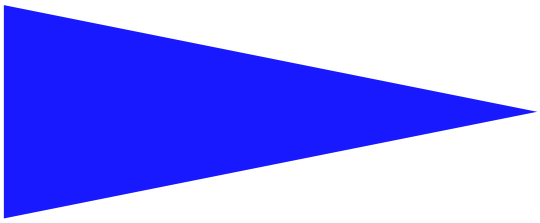
CASH Design Space Exploration

Liqiang He, Romain Dolbeau, André Seznec

I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1815



CASH DESIGN SPACE EXPLORATION

LIQIANG HE, ROMAIN DOLBEAU AND ANDRÉ SEZNEC

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

CASH Design Space Exploration

LIQIANG HE, ROMAIN DOLBEAU and ANDRÉ SEZNEC

Systèmes communicants
Projets CAPS

Publication interne n ° 1815 — September 2006 — 59 pages

Abstract: As the increasing of issue width has diminishing returns with superscalar processor, thread parallelism with a single chip is becoming a reality. In the past few years, both SMT and CMP approaches were first investigated by academics and are now implemented by the industry. In some sense, SMT and CMP represent two extreme design points. CASH parallel processor (for *CMP And SMT Hybird*) is a possible intermediate design points for on-chip thread parallelism in terms of design complexity and hardware sharing. It retains resource sharing as SMT when such a sharing can be made non-critical for implementation, but resource splitting as CMP wherever resource sharing leads to a superlinear increase of the implementation hardware complexity.

This paper explores the multi-dimensional design space for CASH architecture. It compares the performance of single thread running on CASH, SMT and CMP processors. And then the performances of multi-program workloads and parallel workloads are investigated in these processors. At last, It explores the performance varies on CASH with the changing of cache size, and number of associativity of cache. The experiment results show that the CASH processor has a great potential to improve the performances of single thread workload and most of the multi-program workloads, and at the same time maintains a low implementation complexity than the SMT and CMP.

Key-words: CASH, design space exploration, performance

(Résumé : tsvp)

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

CASH Design Space Exploration

Résumé : As the increasing of issue width has diminishing returns with superscalar processor, thread parallelism with a single chip is becoming a reality. In the past few years, both SMT and CMP approaches were first investigated by academics and are now implemented by the industry. In some sense, SMT and CMP represent two extreme design points. CASH parallel processor (for *CMP And SMT Hybird*) is a possible intermediate design points for on-chip thread parallelism in terms of design complexity and hardware sharing. It retains resource sharing as SMT when such a sharing can be made non-critical for implementation, but resource splitting as CMP wherever resource sharing leads to a superlinear increase of the implementation hardware complexity.

This paper explores the multi-dimensional design space for CASH architecture. It compares the performance of single thread running on CASH, SMT and CMP processors. And then the performances of multi-program workloads and parallel workloads are investigated in these processors. At last, It explores the performance varies on CASH with the changing of cache size, and number of associativity of cache. The experiment results show that the CASH processor has a great potential to improve the performances of single thread workload and most of the multi-program workloads, and at the same time maintains a low implementation complexity than the SMT and CMP.

Mots clés : CASH, design space exploration, performance

1 Introduction

As the increasing of issue width has diminishing returns with superscalar processor, thread parallelism with a single chip is becoming a reality. In the past few years, both SMT (Simultaneous MultiThreading, Figure 1.a) and CMP (Chip MultiProcessor, Figure 1.b) approaches were first investigated by academics and are now implemented by the industry. When considering hardware resource sharing among the processes (illustrated in gray on Figures 1), CMP and SMT represent two extreme design points. A possible intermediate design point for on-chip thread parallelism in terms of design complexity and hardware sharing is CASH [1] parallel processor (for *CMP And SMT Hybird*). CASH retains resource sharing as SMT when such a sharing can be made non-critical for implementation, but resource splitting as CMP whenever resource sharing leads to a superlinear increase of the implementation hardware complexity.

On CASH, sparsely or rarely used functional units, branch predictors, instruction caches and data caches can be shared among several processor cores on a single-chip parallel processor. On the other hand, it keeps separated the major parts of the execution cores (rename logic, wake-up and issue logic, bypass network, register files) where wider issue implies higher hardware and design complexity. CASH can not exploit the complete dynamic sharing of resource enabled on SMT (particularly for single process) but retains part of the resource sharing advantage on concurrent workloads and parallel workloads. It is able to use the whole capacity of instruction and data caches and branch predictors with single process workloads. A process migration from processor P_i to processor P_j would not affect the performance of the I-cache and branch prediction structures. With a parallel application a thread prefetches the instructions and warm-up the predictor structures for one another. Sharing the data cache leads to similar prefetching and also avoids coherency traffic induced by the distributed L1 caches in a CMP.

[1] has illustrated the existence and viability of CASH architecture through simulation experiments, but exploring the whole design space of CASH was not done at that time. This paper is the follow-up work of [1] and explores the design space of CASH in details. It compares the performance of single thread running on CASH, SMT and CMP processors. And then the performances of multi-program workloads and parallel workloads are investigated in these processors. At last, it explores the performance variance on CASH with the changing of cache size, number of associativity of cache, and number of cache banks. In every experiment, we compare the performance of the three architectures with a same total issue width.

The experiment results show that:

1. For single thread workload, CASH outperforms CMP due to the larger shared branch predictor and L1 cache in it, but suffers a low performance than SMT because of the fully-shared hardware resources in the latter;

2. In most cases, SMT can benefit from the fully-shared policy at equal total issue width on multi-program and parallel workload than CASH and CMP; but with issue width

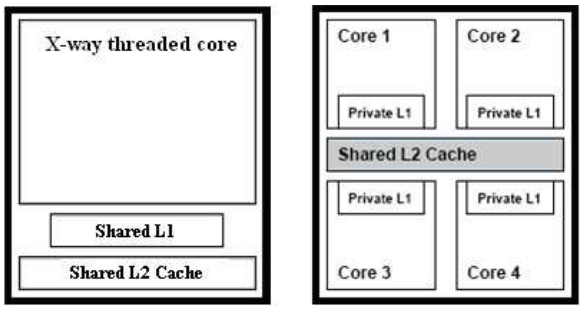


Figure 1: Diagram of the sharing in a SMT processor (a) and a CMP processor (b).

increasing, CASH and CMP improve the performance of programs more higher than SMT, and CASH gets the best average performance, then CMP, and SMT is the last;

3. For multi-program workloads, CASH has four possible cases. Little competition to the shared unit leads to a performance increase due to sharing branch predictor and L1 cache. However, an application sensitive to L1 latency may affect a small performance losing compared with CMP. Some workloads although featuring a large competition for the shared long latency unit or float point unit still have good performance. When the shared long latency unit or float point unit is very heavily used, a performance losing compared with CMP is encountered;

4. When comparing the performance with context switching, CASH performs better than CMP as one can expect because the shared L1 cache structure and branch predictor do not lose information for the threads when they move from one core to another, whereas CMP suffers a cold start effect with the context switching. This could be leveraged to manage temperature hot-spots [2].

5. For parallel workloads, normally the behaviors of the parallel threads are similar. They compete to use the shared unit at the same time and suffer from the longer access time of L1 cache, but benefit from the shared branch predictor and L1 cache. If the number of long latency instructions belonging to a workload is small, it gets a higher relative improvement on CASH than on CMP; Otherwise, when the shared unit is heavily used a lower improvement is obtained since each processor in the CMP has a unshared long latency unit;

6. When investigating the performance varying due to the I-Cache size changing, 128KB I-Cache is shown to be sufficient for most of two-threads and four-threads multi-program workloads;

7. With the increasing of D-Cache size, the performances of workloads continually improve on CASH, but the step of improvement becomes more and more smaller;

8. The difference of performance between two different associativities in I-Cache is small, so a simple two way sets associated mapping is generally suitable for CASH;

9. The difference of performance between two different associativities in D-Cache is very small.

2 Related Work

Prior work has evaluated design space issues for allocating resources to thread execution engine on CMP, SMT and CMPs composed of SMT cores. And some others have investigated both multithreaded and single-threaded clustered architectures that break out portions of a single core and make them more or less accessible to certain instructions or threads within the core.

[3, 4] study the tradeoffs of building multithreaded processors as a group of single-threaded CMP cores, a monolithic SMT core, or a hybrid design of multiple SMT cores and conclude that the hybrid design represents a good performance-complexity design point. However, they do not share resources between cores. Others [5] also explore clustering and hardware partitioning for multithreaded processors and show that the synergistic combination of clustering and SMT minimizes the performance impact of the clustered architecture.

[6] is probably the closet prior work to ours. Different with CASH, it considers sharing entire FPUs and crossbar ports as well as caches, and pays more attention to the latency and area of wiring required by sharing.

3 Complexity of CASH

As explained in [1], the hardware complexity of some of functional stages (register rename, wake-up, issue logic, and operand fetch) increases super linearly and even near quadratically with the issue width in a wide-issue superscalar processor. This is due to the dependencies that may exist among group of instructions that are treated in parallel. All the paths must be dimensioned to be able to treat all the in-flight instructions as if they were all possible dependent. On CASH, a shared instruction cache hardware complexity would be in the same range as the complexity of the several distributed instruction caches on CMP. And for branch predictor, a shared structure on CASH has the same range of complexity on CMP but can obtain a benefit on branch prediction accuracy from the sharing tables. For data cache, a bank-interleaved structure with single-ports banks decrease dramatically the complexity of a multiported structure implementation. Although it lengths the access time by a cycle or more, the single shared L1 data cache is more larger than that in a CMP using the same silicon budget. In addition, the sharing L1 data cache also simplifies the implementation of L2 cache access on CASH while on CMP the coherency between the distributed data caches must be maintained.

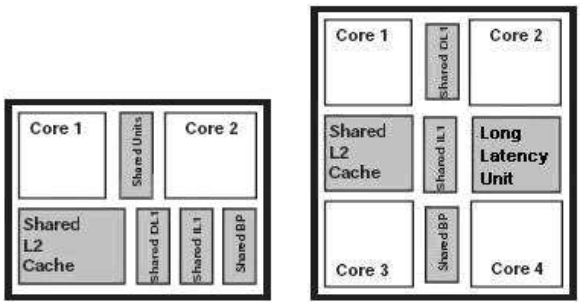


Figure 2: Architecture of simulated CASH processor with two cores (a) and four cores (b).

4 Experimental Methodology

4.1 Simulated "processor" configurations

The simulated CASH processor has two or four cores (Figure 2). All the cores share the L1 cache, L2 cache, branch predictor and a single long latency unit. Every single core has its own functional units. The detailed parameters of configuration are shown in Table 1.

In order to compare the performances of CASH and similar architectures, we also simulate SMT and CMP in our experiments.

CASH and CMP differ by:

1. The first-level instruction and data caches and the buses to the second-level cache;
2. The non-pipelined long latency unit (used for complex integer and floating-point instructions);
3. The branch predictor

And CASH differs with SMT by the non-shared functional units (except the Long Latency Unit), instruction queues, registers and ROBs in every core. Table 1 also lists the simulated SMT and CMP configurations.

In every configuration, the functional units, instruction queues, registers and ROB are shared by all the threads running simultaneously in a SMT processor. On CMP, except for the L2 cache, all the other components are duplicated in every single core.

On CMP, every core has a long latency unit which deals with the complex integer and floating-point instructions. On SMT, one or two long latency units are shared by all the threads. On CASH, all the cores share a single long latency unit and compete to issue one instruction to the queue at every cycle. The other parameters of functional units are common to CASH and CMP.

The branch predictor used in three processors is O-GEHL [7]. It features eight predictor tables indexed through independent functions of the global branch history and branch

Figure 3: The stages of simulated CASH pipeline.

address, and can exploit very long histories through the addition of dynamic history fitting and dynamic threshold fitting. On CASH and SMT, the predictors are shared by all the cores, whereas on CMP it duplicates in every core. Whether shared or not shared, the total hardware budgets of branch predictor in three architectures are same and are 64KB size with base configurations.

On CASH and SMT, the access latency of data cache and L2 cache are different from that on CMP. As explained in [1], due to maintaining coherency between the L1 data cache in different cores, the total access time is longer on CMP then on CASH and SMT. The data caches used on CASH and SMT are bank-interleaved structure and are implemented using single-ported banks. Arbitration is performed on the access to the banks and lengthen the access time by one cycle when comparing with that on CMP.

To compare the performance fairly, we assume the three architectures have same total issue width. We list the correspond configurations to be compared in Table 2.For 2-way, 4-way and 8-way SMT, the processor fetches, decodes, renames and commits 2, 4, or 8 instructions at every cycle for the running threads. For 2-thread workloads, CASH fetches 2 instructions for only one thread at every cycle in a round-robin fashion at 1-way configurations, and 4 instructions at 2-way configuration, and 8 instructions at 4-way configuration. For 4-thread workloads, CASH fetches 4 instructions for only one thread at every cycle in a round-robin fashion at 1-way configuration, and 8 instructions at 2-way configuration. CMP fetches one, two or four instructions for every thread at every cycle at 1-way, 2-way and 4-way configurations.

The pipeline stages of CASH processor is shown in Figure 3. It includes instruction fetch, instruction decode, register renaming, instruction dispatch, execution, memory access, write back, and retire. Some of these functional stages may span over several pipeline cycles. The grey stages (*I.Fetch* and *Mem.Acc*) are shared by all the cores in the processor.

4.2 Simulation environment

We use SMTSIM [8] as the base simulator. It is a cycle accurate execution driven simulator that simulates an out-of-order SMT processor. SMTSIM executes unmodified alpha binaries.

In our simulator, we treat the instructions in Table 3 as long latency instructions in Alpha ISA. Table 2 also lists the latency of them in three architectures.

To simulate the CASH architecture, we modified SMTSIM to act as a CASH processor. This includes the following processes:

1. Duplicate the set of function units (INT, FP, LD/ST) on SMT into several same parts and make them used by different cores on CASH;

Table 1: The base configurations of simulated CASH and SMT, CMP processors.

Parameter	**Value**		
	SMT	CASH	CMP
Functional Unit (INT, FP, LD/ST, LONGLATENCY)	2,1,1,1 (2way) 4,2,2,1 (4way) 8,4,4,2 (8way)	1,1,1,1 (1way) 2,1,1,1 (2way) 4,2,2,1 (4way)	1,1,1,1 (1way) 2,1,1,1 (2way) 4,2,2,1 (4way)
Queue Size (IQ, FQ, LD/ST)	64,64,64 (2way) 128,128,128 (4way, 8way)	32,32,32 per core	32,32,32 per core
Register (IReg, Freg)	128,128 (2way) 256,256 (4way, 8way)	64,64 per core	64,64 per core
ROB	128 (2way) 256 (4way,8way)	64 per core	64 per core
Long Latency Unit Queue	4 entry/unit, issue width=1 inst/queue		
Branch Predictor	O-GEHL, 64KB		16KB/core
BTB (Size, Associativity)	2048, 4		
Icache (Size, Associativity, Bank, Line Length, Latency, Replace)	64KB, 2, 8, 64, 1 cycle, LRU		
Dcache (Size, Associativity, Bank, Line Length, Latency, Replace)	64KB, 2, 8, 64, 2 cycles, LRU		1 cycle
L2 Cache (Size, Associativity, Bank, Line Length, Latency, Rep.)	512KB, 2, 8, 64, 15 cycles, LRU		17 cycles
Pipeline Length	8		
Fetch Width, Fetch Policy	2, 4 or 8 inst/cycle, RR (CASH); 1, 2 or 4 inst/cycle (CMP); ICOUNT.1.X (SMT)		
Decode/Rename/ Commit Width	2, 4, or 8 inst/cycle (CASH, SMT); 1, 2 or 4 inst/cycle (CMP);		

2. Implement a Long Latency Unit on CASH which deals with all the long latency instructions from all the parallel threads;

Prior to execution, caches and predictors were warmed by running one billion instructions of each benchmark after skipping the initialization phase of application. Sets of benchmarks were run together until the completion of 0.8 billion instructions.

In order to compare the performance of CASH and CMP, we also realize a CMP simulator which is also based on SMTSIM. The processes are similar as that for CASH.

Table 2: The base configurations of simulated CASH and SMT, CMP processors.

Workload running environment	**Configuration of architecture**			**Total issue width**
2-thread workload in 2-threaded SMT and 2 cores of CASH and CMP	SMT-2way	CASH-1way	CMP-1way	2
	SMT-4way	CASH-2way	CMP-2way	4
	SMT-8way	CASH-4way	CMP-4way	8
4-thread workload in 4-threaded SMT and 4 cores of CASH and CMP	SMT-4way	CASH-1way	CMP-1way	4
	SMT-8way	CASH-2way	CMP-2way	8

Table 3: Long latency instructions in Alpha ISA and their latencies on SMT, CMP and CASH.

Instruction	**Type**	**Latency**
MULL, MULLV	INT	8
MULQ, MULQV, UMULH		16
DIVS	FP	17
DIVT		30
SQRTT		30

In addition, to be able to compare the performance with context switching, we simulate a thread scheduler in our CMP simulator. We run four-thread workloads in the two-core configuration processors. After a 15 millions cycle interval, the thread scheduler switches two threads out and moves the other two threads in the cores.

4.3 Benchmark set

Our benchmark set comes from the SPEC CPU 2000 [9] benchmark suite. We compiled each program with GCC with the -O3 optimization and produced statically linked executables targeting alpha-ev8 architecture. Each program runs with the reference input set. From these benchmarks, we created twelve two-thread and five four-thread multi-program workloads. They are listed in Table 4. For convenience, we use the serial number of each workload to represent them in the figures of experiment results in following sections.

Table 4 also indicates the type of every combined workload. "F" means floating-point program, and "I" means integer program. In the twelve two-thread workloads, four of them are composed by total floating-point programs, three are composed by total integer programs, and others are mix workloads by floating-point and integer programs. For four-thread workloads, one is combined by all floating-point program, others are mix workloads.

Table 4: Multi-program workloads in our experiment, the type, approximate long latency instructions frequency during simulating, and long latency unit interference between threads.

No.	Workload	Type	Long Latency Inst. Frequency		Interference
1	art-equake	F F	8.19% 0.32%	Mid	Low
2	lucas-mgrid	F F	3.51% 7.77%	Mid	Mid
3	applu-mcf	F I	29.76% 0.22%	High	Low
4	crafty-vpr	I F	0.39% 0.02%	Low	Low
5	twolf-art	I F	0.14% 8.19%	Mid	Low
6	gcc-swim	I F	0.09% 15.88%	Mid	Low
7	gzip-ammp	I F	0% 0.02%	Low	Low
8	bzip2-perlbmk	I I	0% 0.02%	Low	Low
9	gap-eon	I I	6.92% 4.9%	Mid	Mid
10	vortex-parser	I I	0.14% 0.04%	Low	Low
11	apsi-facerec	F F	16.28% 10.58%	High	High
12	fma3d-galgel	F F	0% 12.82%	Mid	Low
13	applu-mcf-crafty-twolf	FIII	29% 0.23% 0.37% 0.13%	Mid	Low
14	lucas-swim-mgrid-applu	FFFF	1.27% 14.95% 7.77% 28.62%	High	High
15	vortex-galgel-bzip2-equake	IFIF	0.14% 13.43% 0% 0.31%	Low	Low
16	apsi-vpr-fma3d-eon	FIFI	16.1% 1.49% 0% 4.75%	Mid	Mid
17	art-ammp-twolf-perlbmk	FFII	8.16% 0.01% 0.14% 0.02%	Mid	Low

To know how many long latency instructions we will meet during simulating, we also list the approximate frequencies of the instructions in Table 4. These frequencies are obtained from the simulating results of CMP architecture with base configuration (64KB Icache, Dcache, 512KL2 cache, etc.). The differences of these frequencies are very small among CASH, SMT and CMP. From Table 4, we know that some programs (*applu*, *swim*, *apsi*, *facerec* and *galgel*) have a very high frequency of long latency instruction, however some others (*equake*, *crafty*, *mcf*, *twolf*, *gcc*, *gzip*, *ammp*, *bzip2*, *perlbmk*, *vortex*, *parser*, *fma3d*, *vpr*), especially the integer programs, have a very low, even near to zero frequency. According to these frequencies, we separate the workloads into three classes, thread with **High**, **Low** and **Mid** long latency instruction frequency. In addition, Table 4 shows the long latency unit interference between threads on CASH processor. Even if some programs have a high long latency instruction frequency during their executions, the competition of using long latency unit between them and the other threads in the same workload is not always very

high. In our workloads, only two of them highly compete using the long latency unit, and the others compete only very rarely on the contrary.

Besides the multi-program workloads, we also chose two parallel workloads (*barnes* and *ocean*) from SPLASH2 [9] benchmark sets. We run them in two-thread and four-thread modes respectively in our CASH simulator and compare the performance with that on SMT and CMP.

4.4 Performance comparison methodology

Fair performance comparison of hardware exploiting thread-level parallelism of different architectures is a difficult issue, using average IPC may lead to biased results towards high IPC applications and cause a unfair conclude for different architectures. So when comparing the performances of CASH, SMT and CMP, we use a relative improvement [11] ***RI*** metric which balance throughput and fairness as shown in Equation 1. Where the IPC_{wld} is the IPC of a thread in a given workload, and the IPC_{alone} is the IPC of a thread when it runs isolated. The fraction IPC_{wld}/IPC_{alone} is the relative improvement of a thread, and the sum of IPC_{wld}/IPC_{alone} of all threads in a workload is the total relative improvement of the workload.

$$RI = \sum_{threads} \frac{IPC_{wld}}{IPC_{alone}} \tag{1}$$

In addition, to compare CASH with CMP, we need a reference and will exploit another metric, reference improvement (***Ref***) which is shown in equation 2. Where IPC_{ref} we used is the IPC of a thread running alone on CMP.

$$Ref = \sum_{threads} \frac{IPC_{wld}}{IPC_{ref}} \tag{2}$$

When comparing the performances of different configurations of CASH architecture, we use the IPC throughput as a simple method. And for the cache hit rate, we show it directly in our experiments.

5 Long Latency Unit Issue Policy

On CASH architecture, the long latency unit is shared by all the cores. Long latency instructions belonging to different threads are issued to a same instruction queue of the unit, such that causing a competence on the entries of the queue. A long latency unit issue policy must be carefully designed and/or selected to decrease the number of conflicts on the queue. In our experiments, we emulate four different issue policies for CASH processor. They are as follows:

1. **Round-Robin (RR)**: long latency instructions belonging to different threads are issued to the queue at a round-robin fashion, and only one thread is permitted to issue long latency instructions to the queue at a single cycle;
2. **Least Serviced Issue First**: the thread which has long latency instruction needing to be issued and has issued the least times of long latency instructions to the queue in the past has the permission to issue at every cycle, and only one thread is permitted to issue at a single cycle;
3. **ICOUNT**: the thread which has long latency instruction needing to be issued and has least instructions in the long latency instruction queue has the permission to issue at every cycle, and only one thread is permitted to issue at a single cycle;
4. **ICOUNT Reverse**: the thread which has long latency instruction needing to be issued and has the most instructions in the long latency instruction queue has the permission to issue at every cycle, and only one thread is permitted to issue at a single cycle.

Our experiment results show that the long latency unit issue policy does not need to prefer one or some threads in a workload due to the low long latency unit interference between the threads at most cases, and RR is the best issue policy when considering the overall performance and fairness. We will use RR as the issue policy in the other experiments.

6 Results

In this section, we present the experiment results. First, we compare the performance of single thread running on CASH, SMT and CMP. Then, the results of multi-program workloads in three architectures are presented and analysised in detail. Third, we investigate the performance varying when changing the cache size, and number of cache associativity in three architectures.

6.1 Single thread performance

We compare the single thread performance on SMT, CMP and CASH with base configuration. The CMP and CASH have two cores in the chip. The difference between SMT and CASH is the unshared separated core in the latter. And the difference between CASH and CMP is the shared long latency unit and memory hierarchy in the former. Here, we do not compare the performance of SMT with CASH and CMP at 1-way configuration.

Figure 4 shows the experiment results. On CASH, the I-Cache, D-Cache, and branch predictor are shared by two cores and so the sizes are double than that on CMP for single thread. From the figure, except for *bzip2* and *gap*, all the benchmarks obtain a higher IPC performance on CASH than on CMP. The average improvement of CASH over CMP is 4.47%, 5.79%, and 4.75% for 1-way, 2-way, and 4-way respectively. At equal total issue width, SMT gets higher IPC than CASH and CMP.

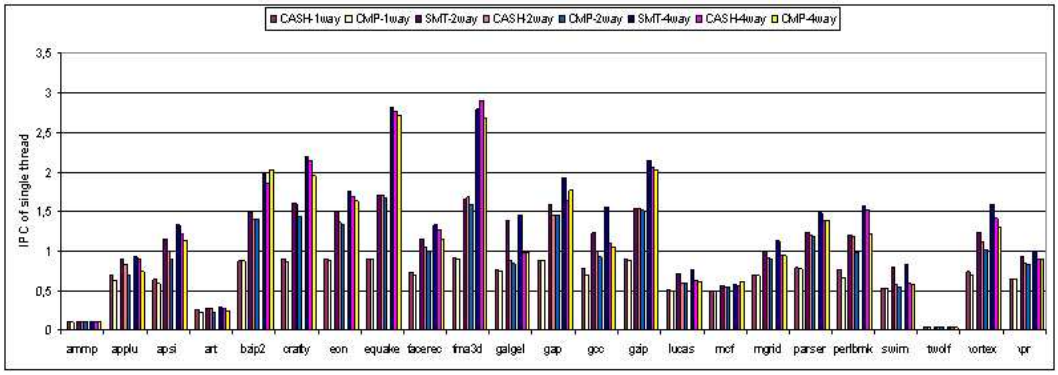


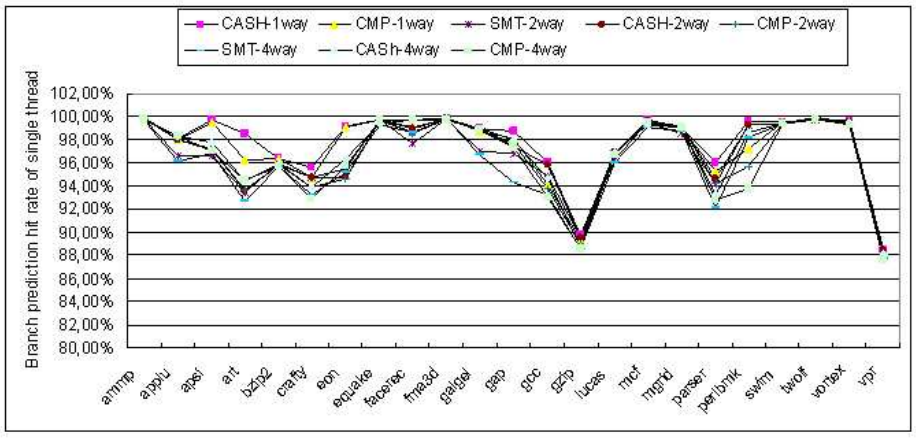
Figure 4: Single thread performance in SMT, CASH and CMP processor.

Table 5: Average hit rates of branch prediction, I-Cache, D-Cache in three architectures.

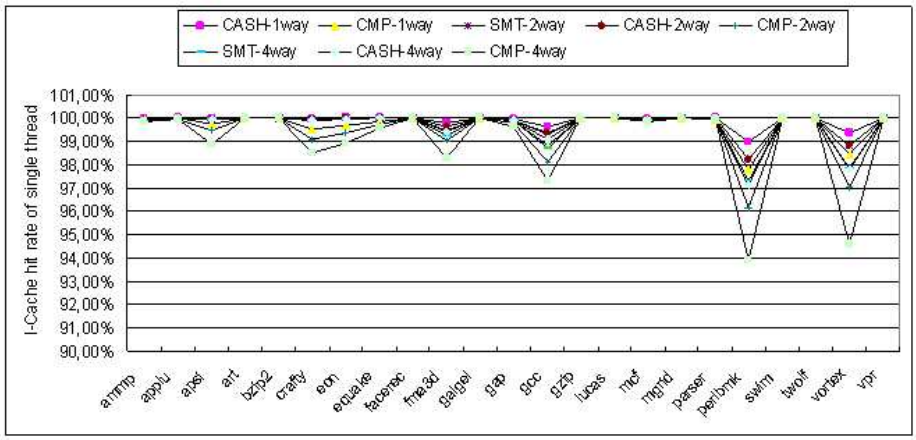
	Bran. pred. hit rate (%)			I-Cache hit rate (%)			D-Cache hit rate (%)		
	1way	2way	4way	1way	2way	4way	1way	2way	4way
SMT	—	96.71	96.23	—	99.82	99.71	—	85.52	84.79
CASH	97.73	97.00	96.94	99.90	99.82	99.69	86.73	86.40	86.10
CMP	97.27	96.61	96.47	99.71	99.47	99.11	86.09	85.91	85.58

Figure 5 shows the branch prediction hit rate and cache hit rates of the threads in three architectures, and the average hit rates are listed in Table 5. For branch prediction (Figure 5.a), near half of the programs obtain a lower hit rate with the issue width increasing, which is clearly due to the more instructions are in the predicting path and cause more mispredictions then that in a lower issue width. For the other programs the hit rates change little when the issue width increases. When comparing the hit rate of single thread, CASH obtains a higher rate at most cases. For *applu*, *art*, *gap* and *lucas*, the hit rates of CASH and CMP are very near and higher than that of SMT, and for *gzip* and *perlbmk*, the hit rates of CASH and SMT are very near and higher than CMP. The average hit rates in three architectures are shown in Table 5.

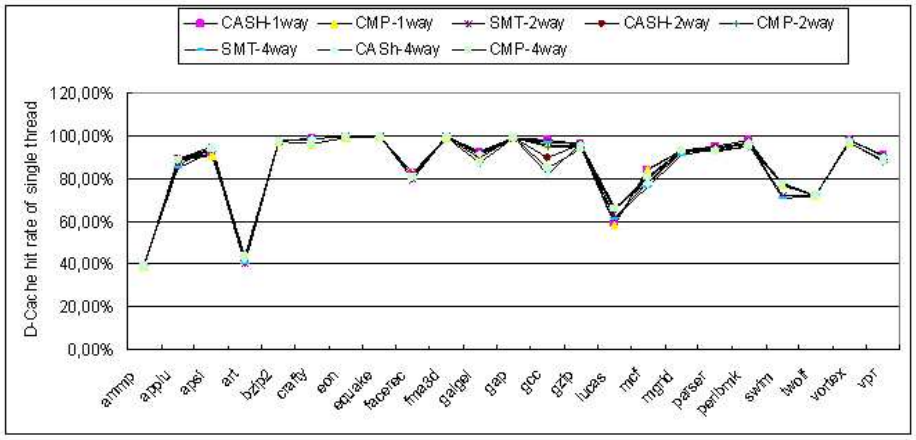
For I-Cache (Figure 5.b), due to the double size for single thread on SMT and CASH, the hit rates are higher than or equal to that on CMP. For most of the threads, the difference of I-Cache hit rate between the three architectures is small. But for memory intensive programs (*apsi*, *crafty*, *perlbmk*, and *vortex*), the larger size of I-Cache improves the hit rate very much. And for *gcc*, *eon* and *fma3d*, the larger size of I-Cache also improves the hit rate very much. At the same time, with the issue width increasing, the hit rates of these programs decrease continuously. For the other programs, the hit rates change little when



(1) Branch prediction hit rate



(2) I-Cache hit rate



(3) D-Cache hit rate

Figure 5: Hit rate of branch prediction and L1 cache of single thread on SMT, CASH and CMP.

the issue width varies. In addition, near half of the programs reach the top point of the hit rate, it shows that an average 32KB I-Cache size for every thread is suitable for these programs in the three architectures. In average, the hit rates on SMT and CMP are near and a little higher than that on CMP, the values are also shown in Table 5.

For D-Cache (Figure 5.c), normally doubling the size improves the hit rate on CASH when comparing with that on CMP. For *gcc* and *mcf*, the hit rates are a little worse on CASH than on CMP. That is partly because the improved I-Cache hit rate causes more instructions being fetched into the queue and needing more data to be fed for these instructions, such that cause a little more cache miss in the D-Cache. With the same reason, the D-Cache hit rate on SMT decreases for some threads when comparing with on CMP and CASH. In average, the CASH obtains the highest hit rate. SMT obtains a higher average hit rate than CMP in 2-way configuration, and a lower one than CMP in 4-way configuration. The differences of the average D-Cache hit rate among them are less than 1.5%, and the values are shown in Table 4.

6.2 Multi-thread workload performance

We run the multi-thread workloads on SMT, CASH and CMP with base configuration. The two-thread workloads run on CASH and CMP with two cores, and four-thread workloads run in them with four cores in a chip. And the SMT processor features twice the core width of the CASH and CMP processors. The total I-Cache size, D-Cache size and branch predictor size are all 64KB in three architectures. And all the three architectures use a shared 512KB L2 cache. The access latency of L1 D-Cache is one cycle longer on CASH and SMT than on CMP, but the latency of L2 cache in the two processors are two cycles less than on CMP due to the unneeded coherency operations between cores.

6.2.1 performance of two-thread workload

The performances measured in IPC of two-thread workloads on SMT, CMP and CASH are shown in Figure 6. We compare the 1-way CASH and CMP with 2-way SMT first. In this case, SMT performs better than CASH and CMP for most of the workloads. For *gzip-ammp*, the performance on SMT is much higher than that on CASH and CMP. This is because the two programs in the workload have very different IPC performances, and SMT can benefit a lot from the fully-shared structure to improve the total performance. And for workloads *lucas-mgrid*, and *applu-mcf*, CASH has near the same performance as CMP. This is because 1) the two programs in workloads have very similar IPC performance, and benefit less from the compensate of using one shared larger L1 cache; 2) the benefit from shared L1 cache and branch predictor diminishes from the longer access latency of the L1 cache. For workloads *twolf-art*, *gcc-swim* and *apsi-facerec*, the performances on CASH are higher than that on CMP. For the other 2-thread workloads, CMP gets better performance than CASH.

When the total issue width increases, all the two-thread workloads obtain a higher performance in three architectures but CASH and CMP benefit more than SMT. When compare 2-way CASH and CMP, half of the workloads get higher performance on CASH than on

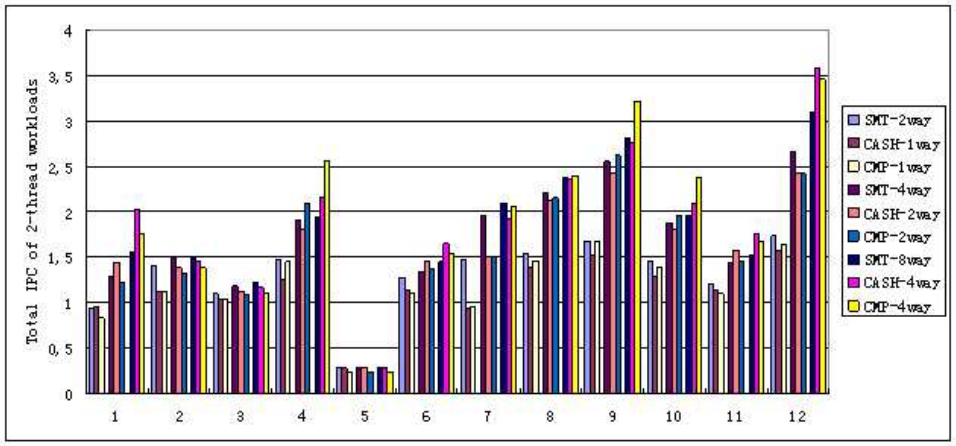


Figure 6: Overall IPC performances of 2-thread workloads on SMT, CMP and CASH.

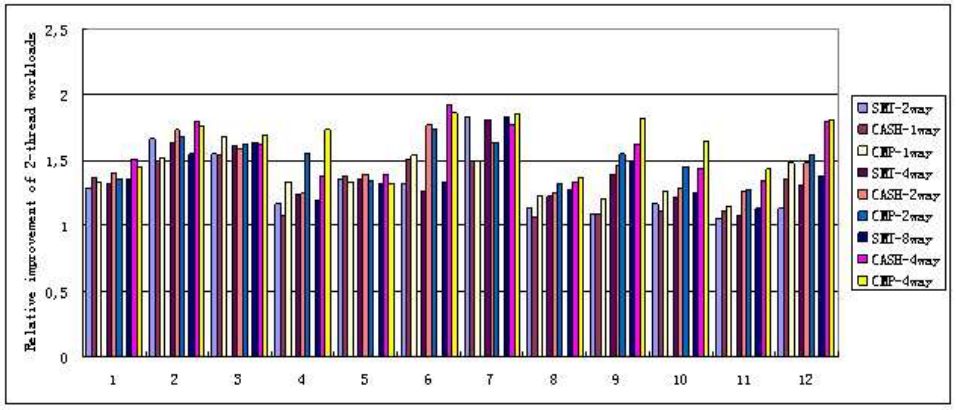


Figure 7: Relative improvement of 2-thread workloads in three architectures.

CMP, and two workloads get near same performance in two processors, and the other four perform worse on CASH than on CMP. When comparing CASH and CMP with 4-way SMT, the latter still gets best performance for six workloads, and the other two all get best performance for three workloads.

When the total issue width is 8-way, CASH gets best performance for four workloads, and both CMP and SMT get best performance for three workloads. For workload *twolf-art* and *bzip2-perlbmk*, the three architectures get very similar total performance.

Using relative improvement ***RI*** as metric to compare the speedup of three architectures, the performance of every workload is shown in Figure 7. In figure 7, all the workloads benefit

Table 6: Average of relative improvements of all the two-thread workloads in three architectures.

Total issue width	**SMT**	**CASH**	**CMP**
2-way	1.309484	1.296165	1.379828
4-way	1.3702	1.4573	1.50348
8-way	1.39486	1.57572	1.644178

Table 7: Average of AI of the two-thread workloads on CASH and CMP.

Total issue width	**CASH**	**CMP**
1-way	1.38267	1.379828
2-way	1.5335	1.50348
4-way	1.65487	1.644178

from multi-thread running in three architectures. The average relative improvements of the two-thread workloads are shown in Table 6. At every case, CMP obtains a highest average relative improvement among the three architectures. CASH gets higher average relative improvement than SMT when the total issue width is 4-way and 8-way. When the total issue width is 2-way, SMT gets a littler higher average performance than CASH.

Using reference improvement ***Ref*** as metric to compare the CASH and CMP, we show the results in Figure 8. Here, we use the performance of single thread running alone on CMP as the reference. From figure 8, CASH obtains higher improvement than CMP at 1-way configuration for five workloads, and CMP obtains the one for six workloads. And for workload *lucas-mgrid*, CASH and CMP get same performance at 1-way configuration. At 2-way configuration, CASH gets higher improvement than CMP for seven workloads, and same improvement as CMP for two workloads, and CMP only gets the higher improvement for three workloads. At 4-way configuration, the same conclusion is obtained. So most cases, CASH can improve the performance higher than CMP with a less hardware budgets for 2-thread workloads. The average Ref is list in Table 7.

To know the performance more clearly on CASH, we compare the IPC of every thread in all the workloads in three architectures with different issue widths in Figure 9. At 2-way of total issue width (Figure9.1), SMT can benefit a lot from the fully-shared structure and obtains high performance at many cases, especially for *lucas-mgrid* and *gzip-ammp*. Due to the larger shared L1 cache and branch predictor, CASH outperforms CMP also for many threads. One important thing that we could draw from the figure is the unfairness of SMT which comes from the ICOUNT fetch policy that favors the fastest thread when fetching instructions, whereas CASH and CMP have the fairness for the threads in a workload. At 4-way and 8-way of total issue width, a same trend can be obtained as at 2-way case.

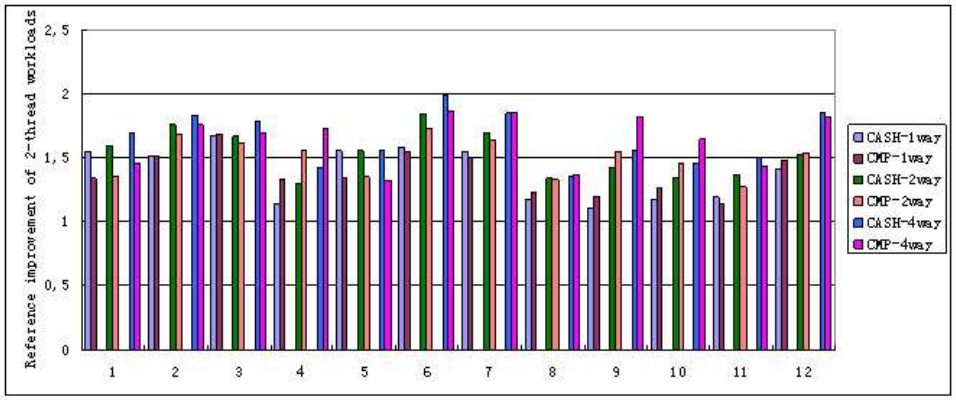
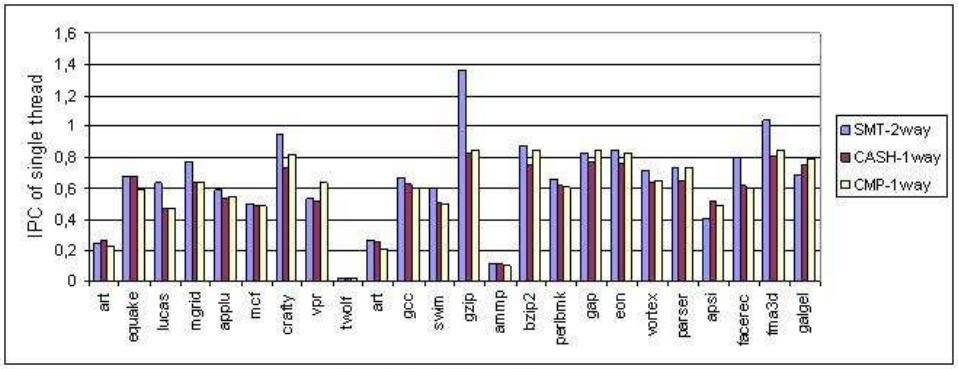


Figure 8: Reference improvement of 2-thread workloads on CASH and CMP.

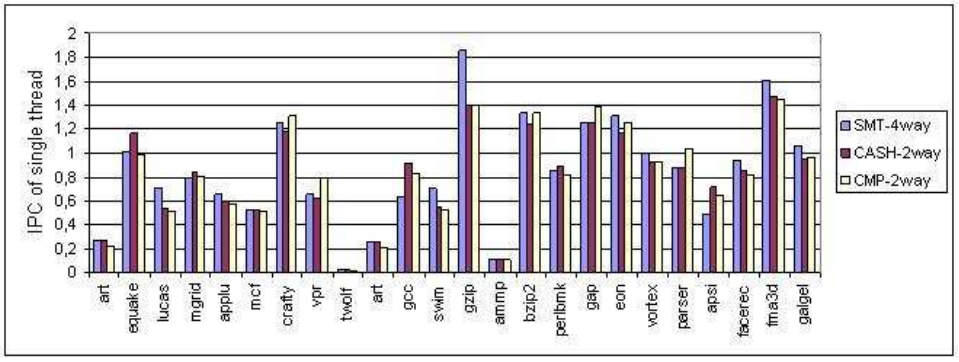
We analysis the performance of the single thread on CASH and CMP more detail at here. They can be classified into 4 classes. 1). For *art-equake*, *gcc-swim* and *fma3d-galgel*, the *equake*, *gcc* and *fma3d* programs benefit a larger L1 cache on CASH and get higher IPC than that on CMP, but do not compete the long latency unit with the co-running thread, *art*, *swim* and *galgel*, at the same time, such that cause a whole performance improvement. 2). For *bzip2-perlbmk* and *vortex-parser*, the *bzip2* and *vortex* are both memory intensive and integer intensive programs, the added one cycle access time of L1 D-Cache causes them degrading the performances. 3). For *gap-eon*, all the two programs have a relative high frequency of long latency instructions and compete to use the shared long latency unit, thus cause a total performance degrading. 4). For *lucas-mgrid* and *apsi-facerec*, all the programs compete to use the shared long latency unit which may cause performance degrading, but benefit more from the larger shared L1 cache and branch predictor at the same time, the latter factor makes the total performance improving. Due to the same shared L1 cache and branch predictor on SMT, the performance of single thread in it is like that on CASH. With the issue width increasing, SMT suffers more from the ICOUNT fetch policy which causes a lower improving of performance than CASH and CMP.

We give the branch prediction hit rate of single thread in Figure 10. SMT and CASH benefit from a shared larger branch predictor and get higher hit rate at near half of the cases, for the other cases a lower hit rate than on CMP normally comes from a high forward speed causes more branch instructions to be predicted and more mis-predictions occurring. With the issue width increasing, the differences of the forward speed among three architectures are smaller and smaller, and the behaviors of thread on CASH and CMP are more likely and differ from that on SMT. This cause the hit rate on CASH and CMP become similar and are higher than that on SMT.

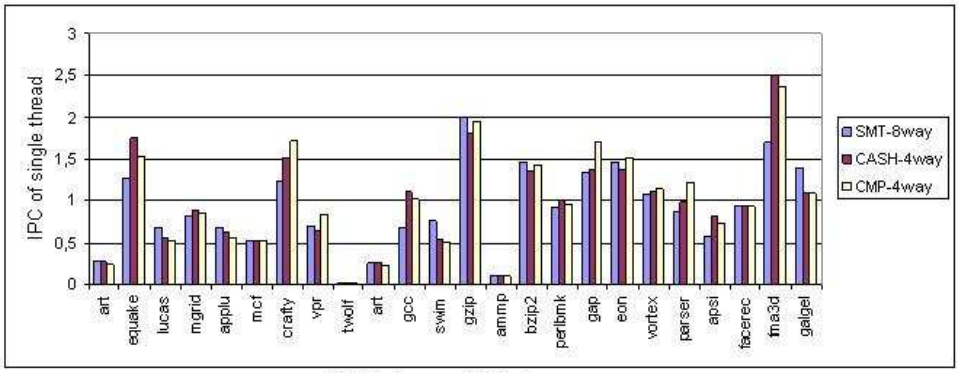
The I-Cache hit rates of single thread with different issue widths are presented in Figure 11. At most cases, CASH and SMT obtain a higher hit rate than CMP due to the shared



(1) Total issue width is 2-way



(2) Total issue width is 4-way



(3) Total issue width is 8-way

Figure 9: Single thread performance in three architectures with different issue widths.

structure, but on the contrary if the shared structure is competed very much by the co-existed threads in a workload, *applu-mcf* and *crafty-vpr* for example. In addition, some threads reach the highest point of I-Cache hit rate (*art*, *lucas*, *mgrid*, *twolf*, *gzip*, *facerec*) which means that 32KB I-Cache size is sufficient for them during the simulation. So if the two threads in a workload have similar behaviors the I-Cache hit rate of them on CASH will like that on CMP, otherwise the hit rate maybe higher for one thread and lower for the other. With the issue width increasing, the increasing of hit rate on CASH and CMP for most of the threads are higher than that on SMT, CMP increases more at 2-way and 4-way of total issue width, and CASH increases more at 8-way of total issue width.

The D-Cache hit rates of single thread in two-thread workloads are given in Figure 12. SMT and CASH obtain higher hit rate than CMP for most of the threads. With issue width increasing, the hit rate of thread decreases in all the three architectures, but CMP at a lower speed than CASH and SMT due to the one cycle faster access latency in it. In some cases, the thread (*art*, *applu*, *mcf* and *swim*) gets a lower hit rate even with a larger shared D-Cache. The differences of hit rate among three architectures are very small in absolute values, but these small differences cause the final IPC performance of threads in them larger differences.

In all, the shared structure on SMT and CASH can bring a higher branch prediction hit rate and L1 cache hit rate at most cases, and then causes a higher single performance. But due to the one cycle longer access latency in D-Cache, some threads get a lower hit rate and a lower performance. On SMT, the threads in a workload compete to use the shared structures, and get a higher performance than on CASH and CMP if the competence is really a benefit of sharing, and get a lower performance on the contrary if the competence causes a thread obtaining more resources and the other obtaining less.

6.2.2 performances of four-thread workloads

As mentioned before, when running four-thread workloads, we assume a four-core CASH and a four-core CMP, and the SMT features four threaded and only 4-way and 8-way of SMT are considered.

In 4-thread experiments, the size of I-Cache, D-Cache and branch predictor are all 16KB in every CMP core, whereas CASH and SMT still provide a total 64KB shared L1 cache and branch predictor for all the threads in a workload. The 4 times of larger L1 cache and branch predictor lets CASH performing better than SMT and CMP at most cases.

Figure 13 gives the experiment results of four-thread workload. When the total issue width is 4-way, CASH and CMP perform better than SMT, and all the workloads obtain higher performance in them than on SMT. When comparing CASH and CMP, two workloads perform better on CASH, and three workloads perform better on CMP. When the total issue width is 8-way, all the workloads perform better on CASH than on SMT and CMP. And SMT only have a small improvement for all the workloads when the issue width increases from 4 instructions per cycle to 8 instructions per cycle. For *vortex-galgel-bzip2-equake* and *apsi-vpr-fma3d-eon*, there are large improvements on CASH and CMP than on SMT.

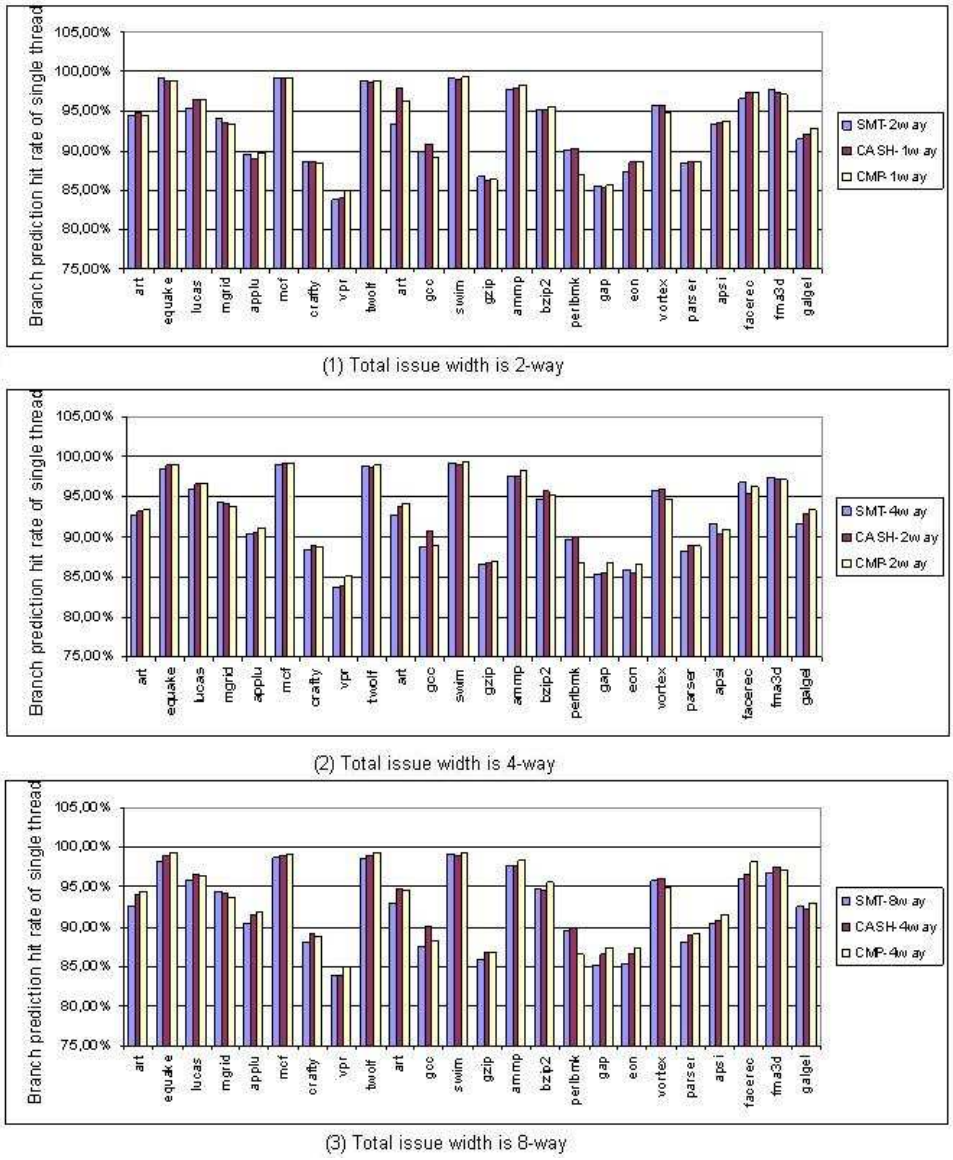


Figure 10: Branch prediction hit rate in three architectures with different issue widths.

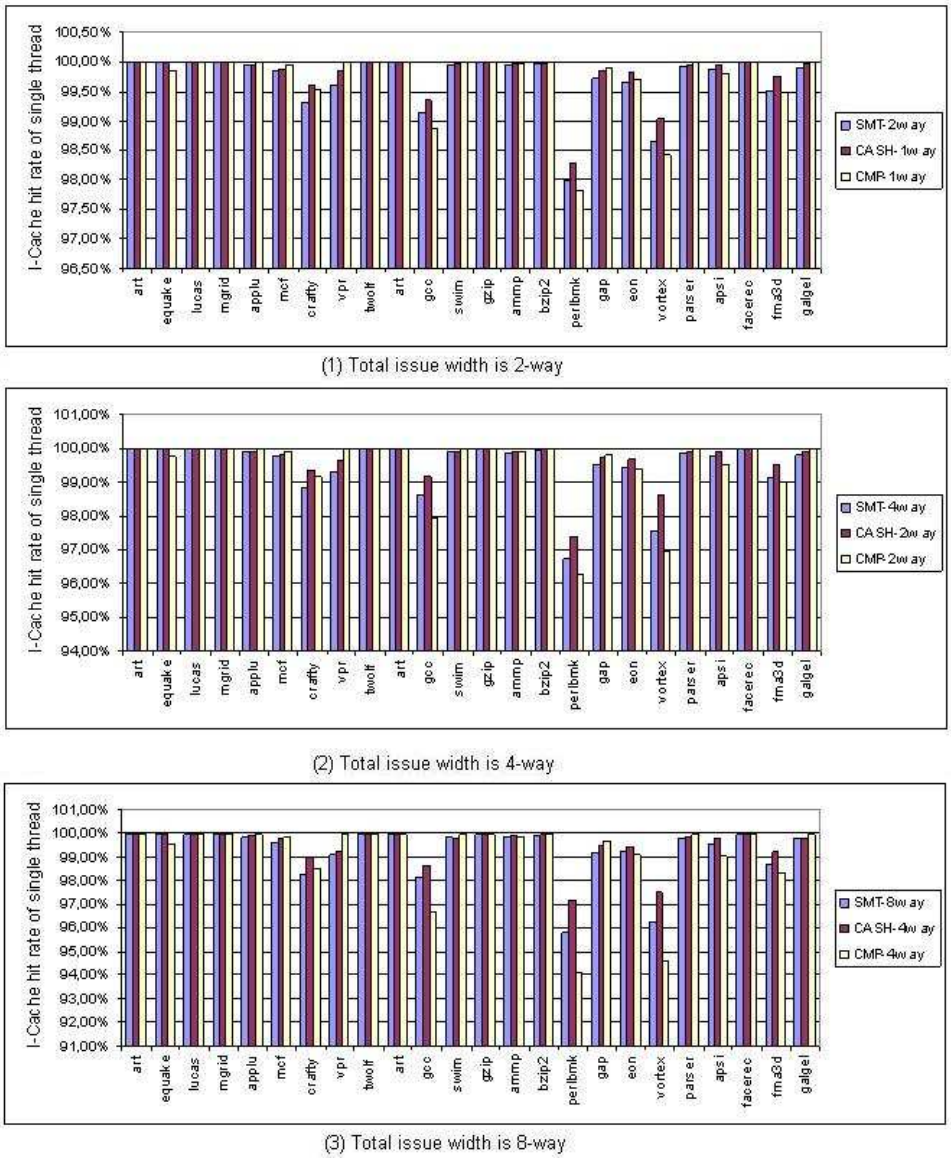


Figure 11: I-Cache hit rate in three architectures with different issue widths.

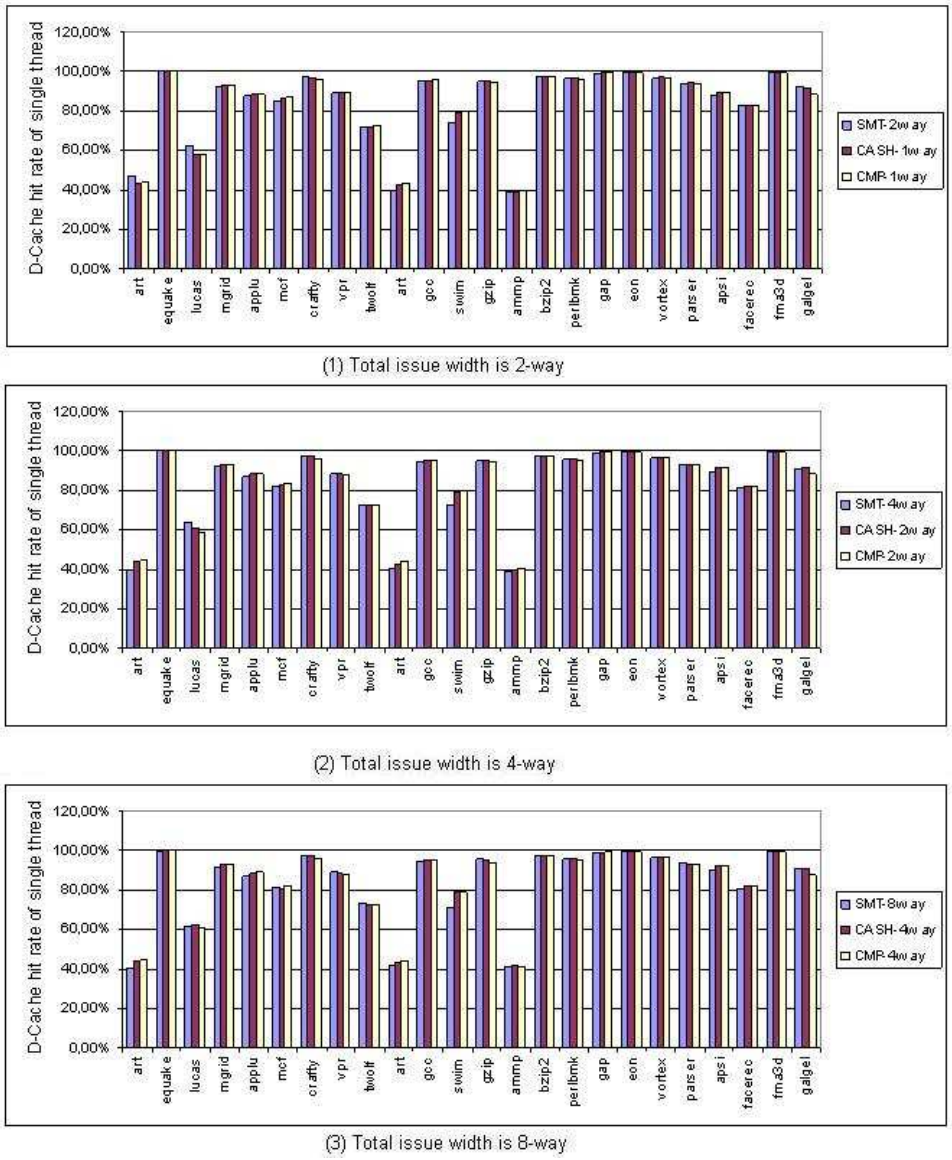


Figure 12: D-Cache hit rate in three architectures with different issue widths.

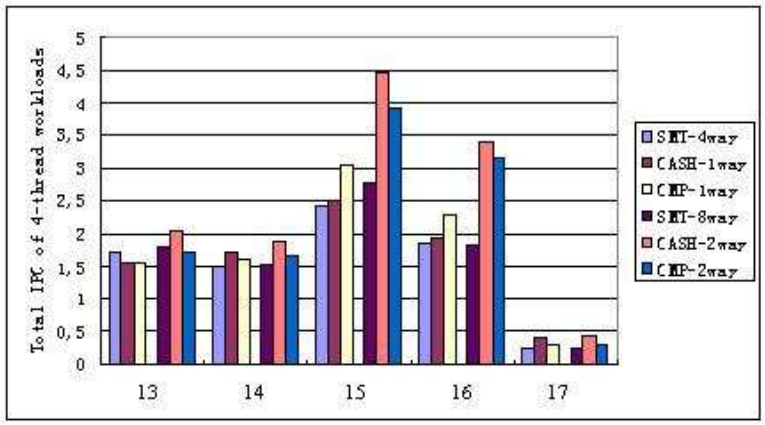


Figure 13: Overall IPC performances of 4-thread workloads on SMT, CMP and CASH.

Figure 14 gives the single thread performance in 4-thread experiments. For single thread, there are same tread of performance as the overall performance. With four threads running together, SMT suffers from the high competence of using shared structure among threads, and gets a worse single performance for the threads. And with issue width increasing, CASH and CMP get better performance from the unshared single cores structure, and CASH still benefits a little from the larger shared L1 cache and branch predictor.

Using relative improvement as metric, we show the results in Figure 15. As mentioned before, we use the single performance of thread with 64KB base configuration as the IPC_{alone} in the equation (1). When the total issue width is 4-way, CASH obtains highest relative improvement for two workloads, and CMP obtains highest relative improvement for three workloads. And SMT always has the lowest relative improvement for all the workloads. With total issue width increasing to 8-way, CASH gets highest relative improvement for four workloads, and CMP gets the highest one for one workload. SMT in 8-way configuration even does not outperform itself in 4-way configuration for three workloads. The average relative improvements for 4-thread workloads are shown in Table 8. SMT always gets the lowest average improvement, and CASH gets the highest relative improvement when the total issue width is 8-way, and CMP gets the highest one when the total issue width is 4-way. The difference of average improvements between CASH and CMP is not very large, but the difference between SMT and the other two is very large (from 41% at 4-way configuration to 70% at 8-way configuration).

Still, we use reference improvement ***Ref*** as metric to compare CASH and CMP, and also use the single thread performance running alone on CMP as the reference. The result is shown in Figure 16. In 1-way configuration, CASH outperforms CMP for three workloads, and CMP outperforms CASH for two workloads. And in 2-way configuration, CASH outperforms CMP for all the workloads. The average value of Ref is shown in Table 9.

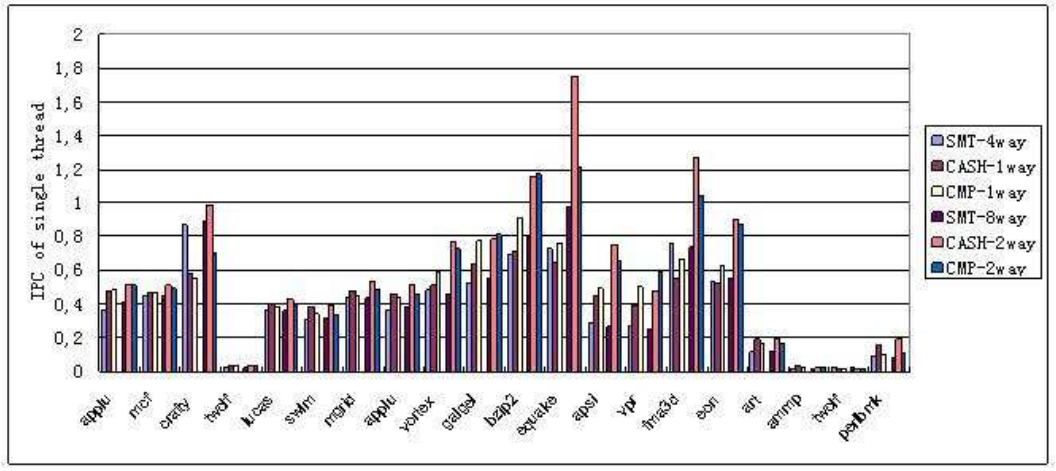


Figure 14: Single thread performance in three architectures with different issue widths.

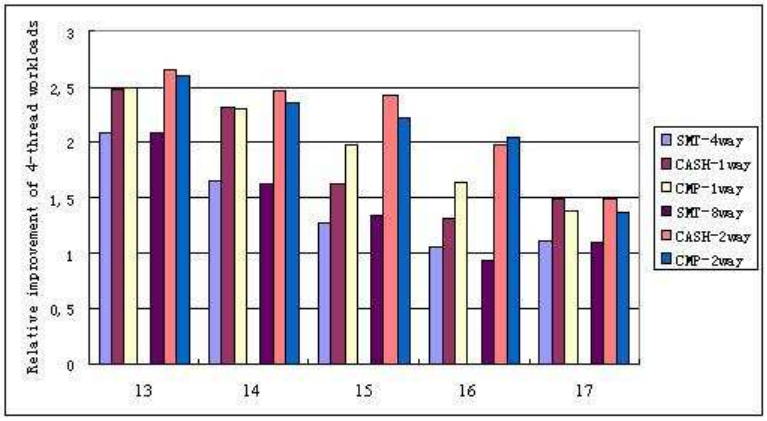


Figure 15: Relative improvement of 4-thread workloads in three architectures with different issue widths.

Table 8: Average relative improvements of three architectures with different issue widths.

Total issue width	SMT	CASH	CMP
4-way	1.43556	1.840746	1.959233
8-way	1.41602	2.19833	2.11194

Table 9: Average reference improvement of 4-thread workloads on CASH and CMP.

Issue width / Total issue width	CASH	CMP
1-way / 4-way	1.934431	1.959233
2-way / 8-way	2.30994	2.11194

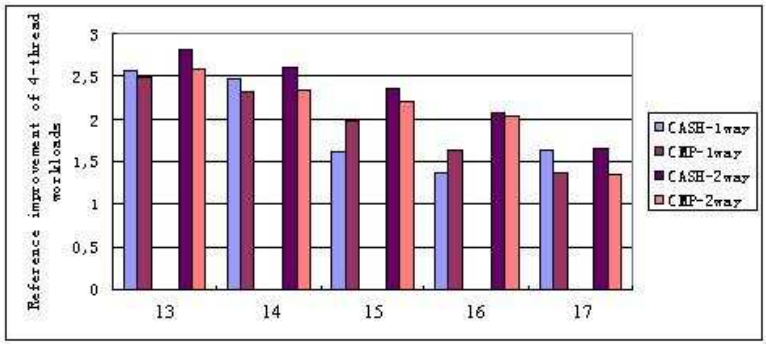


Figure 16: Average improvement of 4-thread workloads on CASH and CMP.

We also present the hit rate of branch prediction and L1 cache at Figure 17, 18, and 19. Figure 17 shows the branch prediction hit rate of single thread in 4-thread experiments. With a shared branch predictor, SMT obtains higher hit rate than CMP at most cases. Although CASH also shares the branch predictor among threads, the hit rates of threads in a workloads are lower than that on SMT and CMP. This is because the CASH has faster forward speed than the two others, and causes more branch instructions to be predicted and more mis-predictions to be produced. The difference of hit rate among the three architectures is very small because they use the same branch predictor and only the size serviced for single thread has little difference.

Figure 18 presents the I-Cache hit rate of single thread in 4-thread experiments. With a shared L1 I-Cache, SMT and CASH get higher hit rates than CMP at most cases. And with issue width increasing, the difference between the shared structure and non-shared structure becomes larger and larger. For memory-intensive programs (*crafty*, *vortex*, *apsi*,

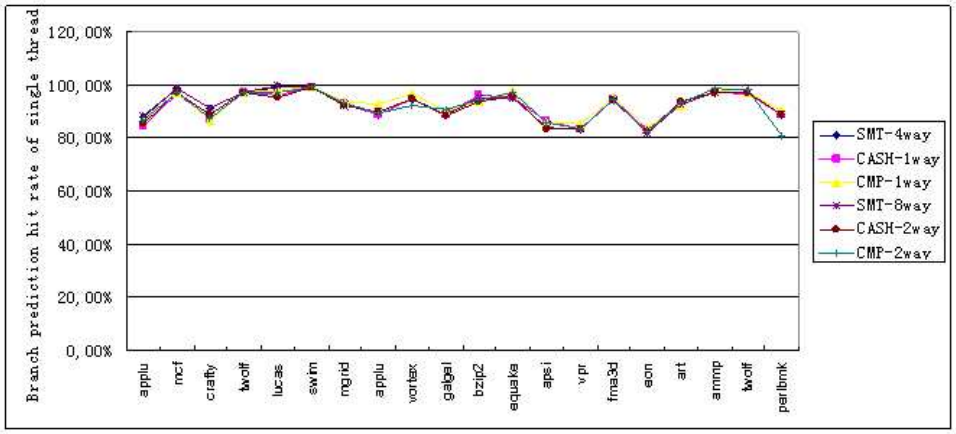


Figure 17: Branch prediction hit rate of 4-thread workloads in three architectures with different issue widths.

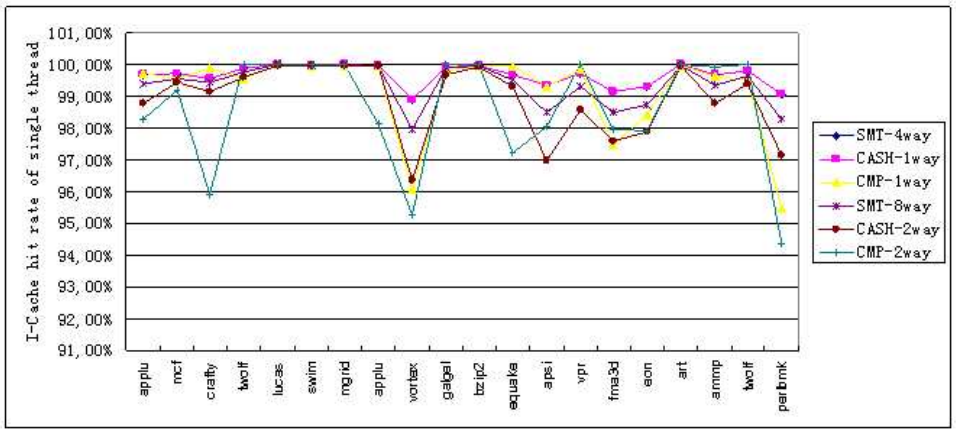


Figure 18: I-Cache hit rate of 4-thread workloads in three architectures with different issue widths.

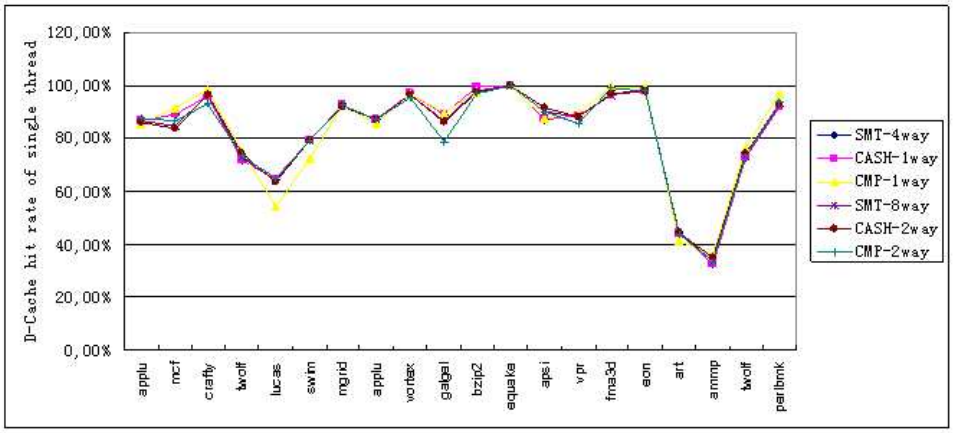


Figure 19: D-Cache hit rate of 4-thread workloads in three architectures with different issue widths.

and *perlbmk*) the shared structure on SMT and CASH permits the threads in the same workload compensate to use the total I-Cache space and gets higher hit rates for them than on CMP. And for *fma3d*, it prefers a larger I-Cache, so the shared structure on SMT and CASH also provide the space to increase the I-Cache hit rate of it. The hit rates of *applu*, *eon* and *ammp* vary very much according to the usable space with other co-exist threads in a workload.

Figure 19 gives the D-Cache hit rate of single thread in 4-thread experiments. SMT and CASH benefit from a larger shared D-Cache space, but suffer from one cycle longer access latency than CMP. The difference of absolute hit rate values of every single thread between two processors is not very larger except for *mcf*, *lucas* and *galgel*. For *mcf*, it is a memory-bounded program, and prefers to a larger D-Cache space. The hit rate of it is affected very much by the co-existed threads running simultaneously. And for *galgel*, it has a relative high frequency of long latency instructions, the D-Cache hit rate of it is also affected very much by the co-existed running threads. For *lucas*, it is included in a workload with high frequency of long latency instructions, the hit rate varies according to the usable long latency unit and D-Cache size.

In all, CASH shows the potential to improve the overall performance and single thread performance ether at two threads or at four thread cases. For four thread workloads, it outperforms SMT and CMP at most cases, and for two thread workloads it outperforms SMT and CMP when the total issue width is larger then four. At the same time, it obtains a similar performance as SMT at lower issue widths with a lower implementation complexity.

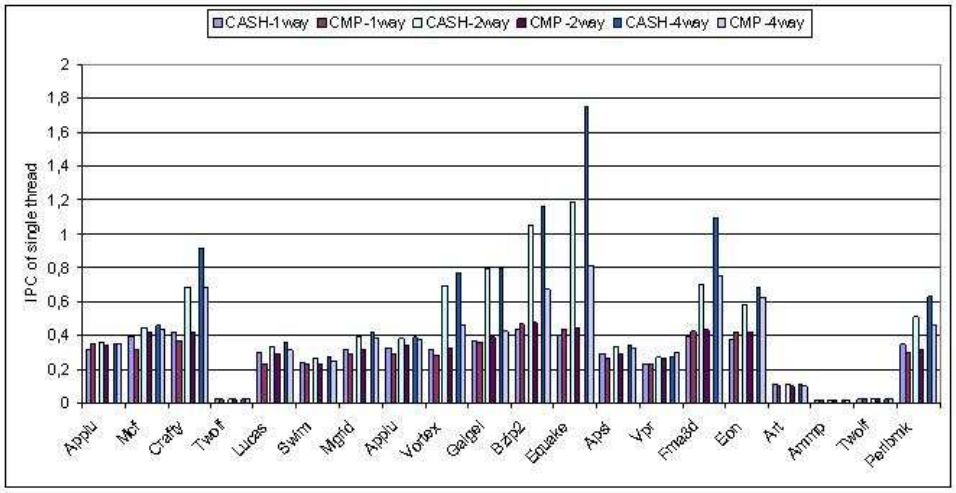


Figure 20: Single thread performance on CASH and CMP with context switch.

6.3 Context switching workload performance

We also compare the performance of CASH and CMP with context switching. Due to the same shared L1 cache structure and branch predictor, we do not compare the behaviors of context switching on SMT and CASH. We simulate the usual four thread workloads in two cores configurations of CASH and CMP. The sizes of I-Cache, D-Cache and branch predictor are all 32KB in every core of CMP, and 64KB on CASH with a shared fashion. The access latency of D-Cache on CASH is one cycle longer than on CMP. We test one time-slice intervals, 15 million cycles. The context switch is in a round-robin fashion. That is the first two threads in the four-thread workload run in the two cores firstly, and the other two threads wait until the interval is reached and then move in. The simulated number of instructions for all the workloads are 800 millions.

Figure 20 shows the single thread IPC performance in the workloads. As showed in [1], with context switch CASH outperforms CMP at most cases because the shared structures do not lose information for the threads when they switch into the other cores. In 1-way configuration, most of programs get better performance on CASH than on CMP, and only *applu*, *bzip2*, *equake*, *fma3d*, and *eon* get better performance on CMP. In 2-way and 4-way configurations, CASH outperforms CMP at near all the programs (except *vpr* at 4-way configuration). For memory intensive programs, *crafty*, *vortex*, *galgel*, *bzip2*, *equake*, *fma3d* and *perlbmk*, CASH improves the performance very much comparing with CMP.

Figure 21 shows the branch prediction hit rate of single thread on CASH and CMP with context switch. For all the issue widths, the hit rates of programs on CASH and CMP are very near. For program *applu*, *crafty*, *mgrid*, *vortex*, and *perlbmk*, the hit rates on CASH

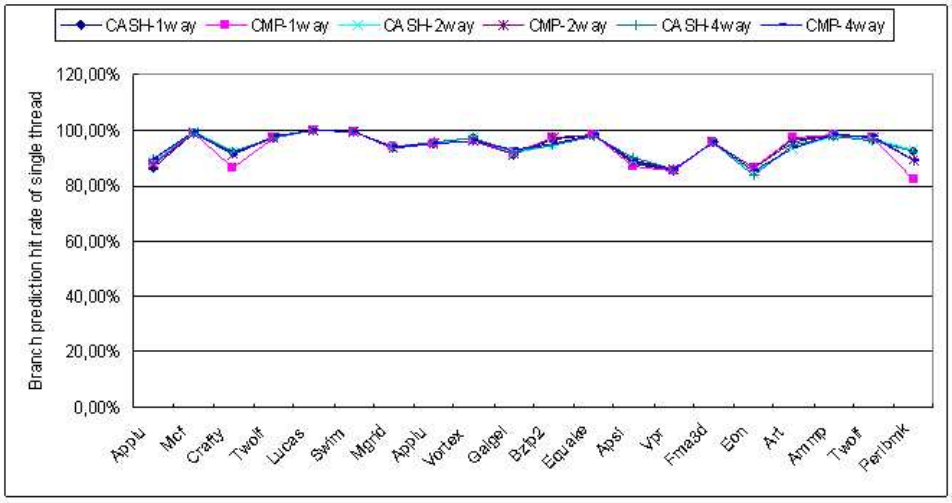


Figure 21: Branch prediction hit rate of single thread on CASH and CMP with context switch.

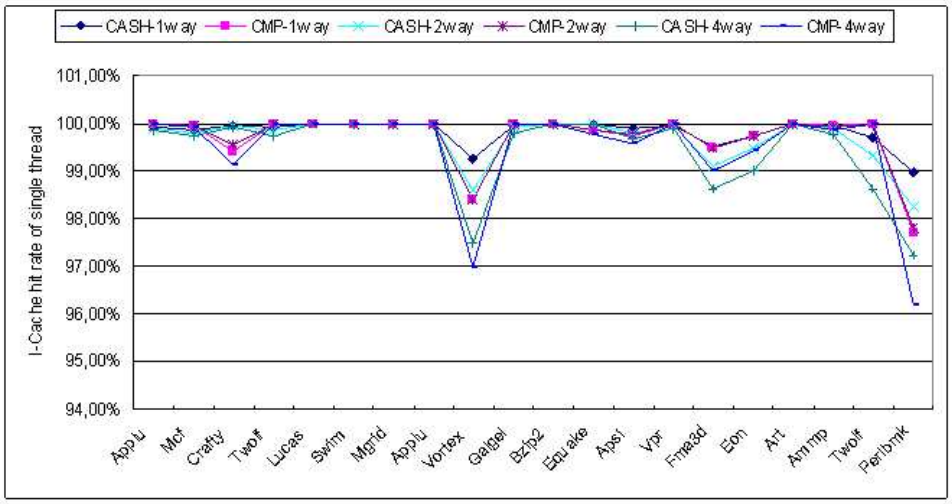


Figure 22: I-Cache hit rate of single thread on CASH and CMP with context switch.

are a littler higher than that on CMP, and for program *mcf*, *fma3d*, *ammp*, *twolf*, the hit rates on CMP are a little better than on CASH.

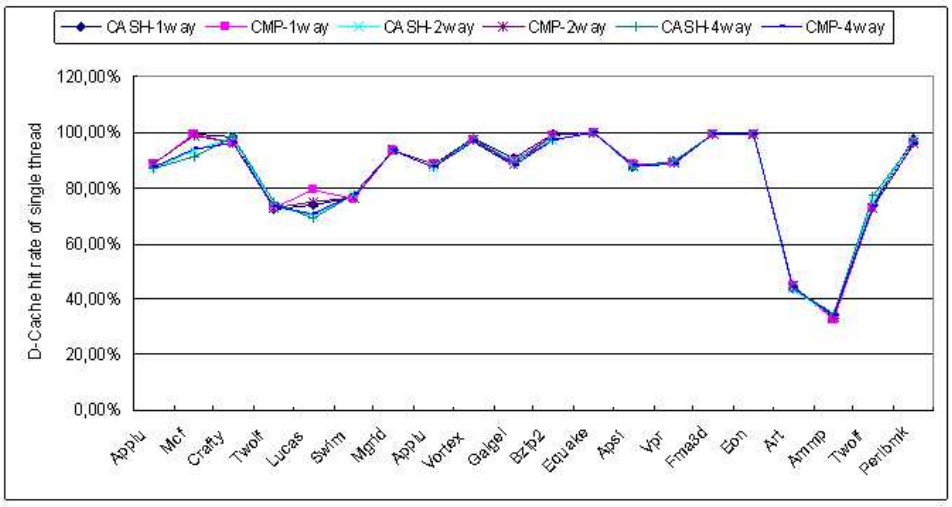


Figure 23: D-Cache hit rate of single thread on CASH and CMP with context switch.

Figure 22 shows the I-Cache hit rate of single thread. Comparing with CASH, CMP gets lower hit rate for a large degree for program *crafty*, *vortex*, *equake*, *apsi*, and *perlbmk*. And for program *applu*, *mcf*, *twolf*, *vpr*, *fma3d*, *eon*, and *ammp*, CMP gets marginal higher hit rate than CASH. Among them, *twolf* and *ammp* have very low IPC speeds, such that the lost data information does not cause much affect on their hit rates. The other programs get near hit rates on CASH and CMP. With issue width increasing, the hit rate of memory intensive thread decreases on CASH and CMP continually, but CASH still gets higher one than CMP.

Figure 23 shows the D-Cache hit rate of single thread. Similar as the I-Cache hit rate, there are only marginal difference between the hit rates of programs on CASH and on CMP. For program *crafty*, *swim*, *galgel*, *vpr*, *ammp*, *twolf*, and *perlbmk*, CASH gets a little higher hit rate than CMP, and for program *lucas*, *mgrid*, *applu*, *apsi*, *art*, CMP gets a little higher one than CASH.

6.4 Parallel workload performance

We compare the performance of parallel workloads using *barnes* and *ocean* in SPLASH-2[9] benchmark suite. SMT, CASH, and CMP all use the base hardware configuration with 64KB I-Cache, 64KB D-Cache, 64KB branch predictor. In these two programs, after the parallel sub-threads are created, the threads exhibit a similar behavior such that the contention on the hardware will occur. Whether the program improves or degrades performance on CASH than on CMP or SMT depends on its own behavior and the hardware architecture. Comparing with CMP, CASH benefits performance from 1) the larger L1 cache, and 2) the

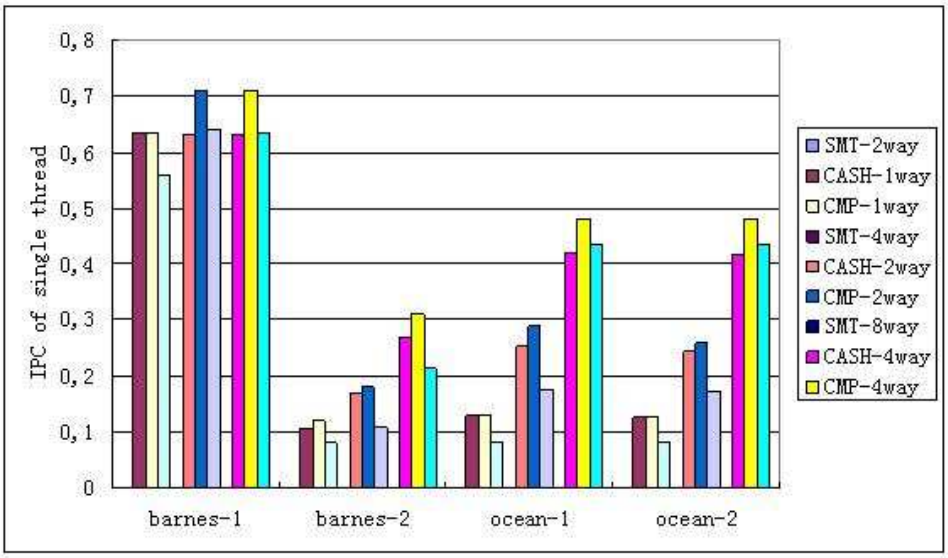


Figure 24: Single thread performance of parallel workloads in 2-thread mode with different issue widths.

larger branch predictor, but also suffers from longer access time in L1 D-Cache. The SMT exploits a full-shared policy with its hardware resources and a different ICOUNT fetch policy from the RR on CASH and CMP, and it shows a middle performance between CASH and CMP. To less the contention of shared hardware, CASH exploits the Round-Robin float-point or long latency unit issue policy which avoids many conflicts on the queue between the threads.

We show the 2-thread performance of parallel workloads in Figure 24. *Barnes-1* and *Ocean-1* are the primary threads, and *barnes-2* and *ocean-2* are the sub-threads. When the total issue width is 2-way, SMT and CASH benefit from larger L1 cache and branch predictor, and obtain better performance than CMP for all the primary threads and sub-threads. When the total issue width is 4-way and 8-way, CASH improves both the primary thread and sub-thread performances largely. It outperforms SMT and CMP for a large degree. For *barnes*, SMT even gets a little worse performance than CMP for the primary thread in 4-way and 8-way configurations. When considering the numbers and frequencies of float-point instructions or long latency instructions, they are very near in three architectures after the sub-threads are created. Due to sharing the float-point unit or long latency unit among the cores, CASH has higher conflict frequency of such instructions than SMT and CMP, but the absolute values are not very high. With the help of Round-Robin issue policy, CASH gets better average performance than the two others.

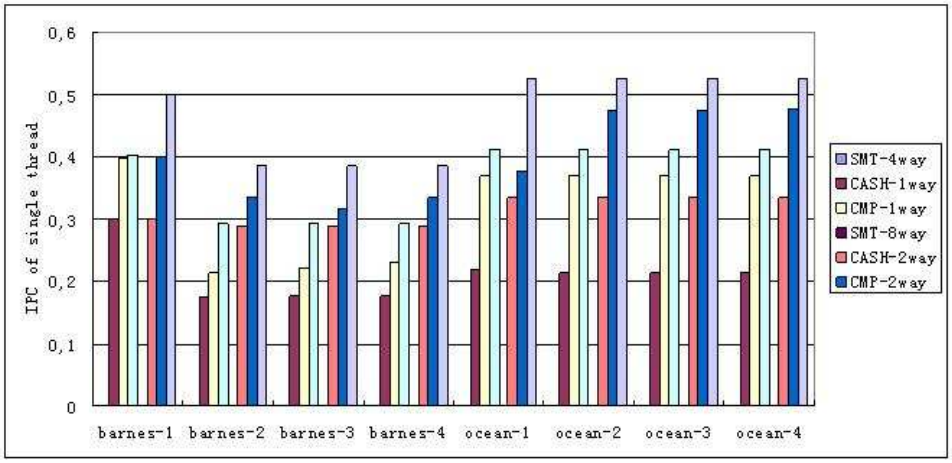


Figure 25: Single thread performance of parallel workloads in 4-thread mode with different issue widths.

The branch prediction hit rate and L1 D-Cache hit rate of the parallel threads are shown in Table 10. The branch hit rate of *barnes* on CMP is a little higher than on SMT and CASH, but the hit rate of *ocean* is a little lower than that on SMT and CASH. CASH gets higher branch prediction hit rate than SMT at most cases. For D-Cache hit rate, *barnes* on SMT and CASH benefits more from the larger cache space than the sufferance from the longer access latency, and gets higher hit rate than on CMP. *Ocean* gets lower hit rate on SMT and CASH than on CMP when the total issue width is 2-way, but gets higher hit rates on SMT and CASH when the total issue width is 4-way and 8-way. Same as branch prediction, CASH obtains higher D-Cache hit rate than SMT at most cases. The I-Cache hit rates in three architectures have little difference. Only CMP shows marginally lower hit rate in *barenes* program.

The performances of parallel workloads running in four threads mode are shown in Figure 25. *Barnes-1* and *ocean-1* are the primary threads, the others are sub-threads created by the primary threads. Different with two threads mode, SMT and CASH get worse average performance than CMP. For SMT, the fully-shared policy provides little space to compensate using hardware resources among threads when the number of threads larger than 4, and longer access latency in D-Cache plays the key role to affect the performance of threads, as a result the IPC of single thread in it is lower on CMP. For CASH, sharing float-point unit or long latency unit causes more conflicts of using the hardware among threads, and with the added effect of longer access latency in D-Cache the performance of single thread also lower than that on CMP. With the unshared policy of functional units, CASH gets better single thread performance than SMT.

Table 10: Branch prediction hit rate and D-Cache hit rate of single thread in parallel workloads in 2-thread mode.

	Total issue width =2way			**Total issue width =4way**			**Total issue width =8way**		
Br. pred. hit rate (%)	SMT	CASH	CMP	SMT	CASH	CMP	SMT	CASH	CMP
Barnes-1	92.72	92.75	94.55	91.76	91.52	93.57	91.26	91.82	92.67
Barnes-2	91.59	90.79	91.57	92.41	91.69	91.92	92.83	91.94	92.68
Ocean-1	99.19	99.23	99.15	99.18	99.31	99.17	99.28	99.31	99.23
Ocean-2	99.26	99.25	99.23	99.28	99.32	99.19	99.34	99.33	99.27
D-Cache hit rate	SMT	CASH	CMP	SMT	CASH	CMP	SMT	CASH	CMP
Barnes-1	98.33	98.27	97.53	98.23	96.17	97.37	98.08	96.12	96.35
Barnes-2	96.88	96.44	97.70	97.28	96.28	97.88	97.45	96.16	97.77
Ocean-1	83.88	87.32	89.25	81.17	83.91	88.45	79.55	83.75	85.58
Ocean-2	92.37	92.26	93.02	91.98	83.80	92.91	90.09	83.63	91.73

With the issue width increasing at 4-thread mode, the difference of performances among SMT, CASH, and CMP becomes larger and larger. CMP improves performance faster than SMT and CASH. The branch prediction hit rate and D-Cache hit rate of single thread in 4-thread mode are listed in Table 11. CMP gets higher branch prediction hit rate than SMT and CASH for *barnes* program, but lower hit rate for *ocean* program at most cases. And SMT gets lower hit rate than CASH for *barnes*, and higher hit rate for *ocean* at most cases. For D-Cache hit rate, CASH gets higher hit rate than SMT and CMP for *ocean*, and lower hit rate for *barnes*. SMT sits at the middle of CASH and CMP at most cases.

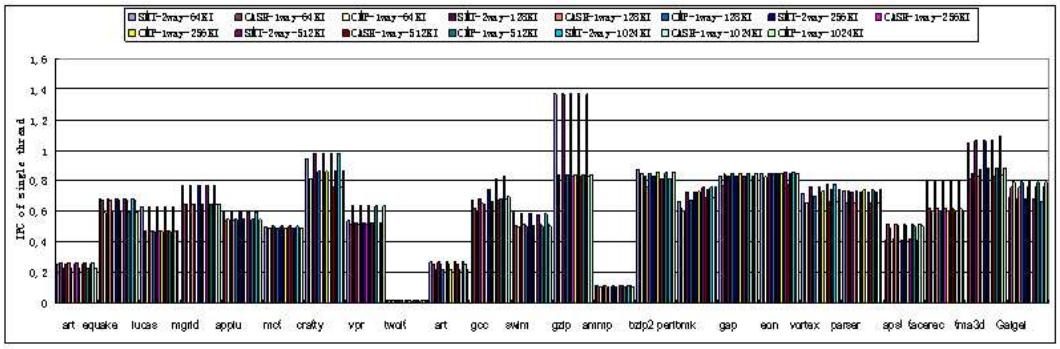
6.5 Varying cache size in CASH processor

6.5.1 L1 I-Cache size

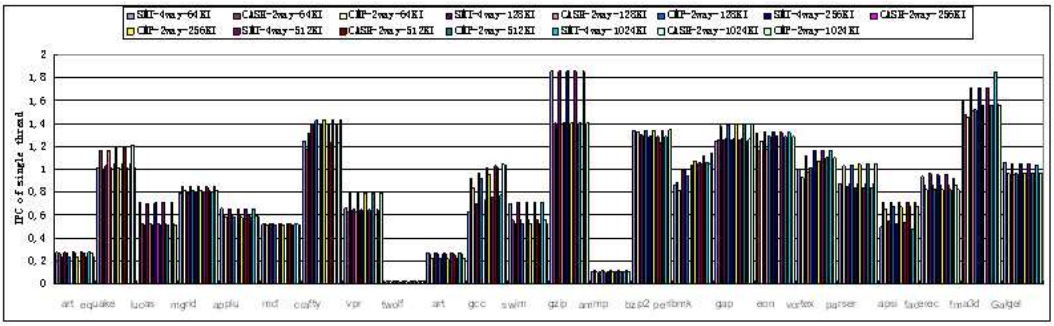
We compare the performance of SMT, CASH and CMP with different I-Cache sizes, the other hardware parameters in them are same as the base configurations.

a. 2-thread workload performance

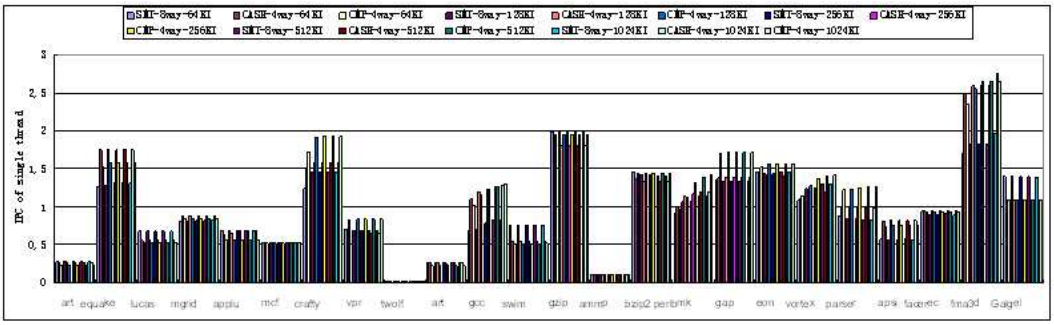
Figure 26 gives the single thread performance of 2-thread workloads in three architectures with different I-Cache sizes and different issue widths. For most of programs, the performance does not change when the I-Cache size enlarges from 64KB to 1MB. Four programs, *gcc*, *perlbmk*, *apsi*, and *fma3d*, prefer to a larger I-Cache size and improve the performances when the size increasing. When the total issue width is 2-way, SMT benefits



(1) Total issue width is 2-way



(2) Total issue width is 4-way



(3) Total issue width is 8-way

Figure 26: Performance of single thread in 2-thread workloads with different I-Cache sizes.

Table 11: Branch prediction hit rate and D-Cache hit rate of single thread in parallel workloads in 4-thread mode.

	Total issue width=4way			**Total issue width=8way**		
Bp hit rate (%)	SMT	CASH	CMP	SMT	CASH	CMP
Barnes-1	90.09%	90.51%	91.77%	89.44%	90.69%	90.64%
Barnes-2	88.60%	90.86%	87.91%	89.68%	91.46%	89.04%
Barnes-3	89.50%	90.83%	89.51%	89.36%	91.01%	89.80%
Barnes-4	89.10%	91.31%	88.74%	90.00%	91.02%	89.48%
Ocean-1	98.72%	98.74%	98.67%	98.72%	98.74%	98.82%
Ocean-2	98.96%	98.82%	98.62%	98.82%	98.82%	98.90%
Ocean-3	99.02%	98.77%	98.60%	98.82%	98.77%	98.84%
Ocean-4	99.11%	98.82%	98.69%	98.89%	98.82%	98.90%
D-Cache hit rate	SMT	CASH	CMP	SMT	CASH	CMP
Barnes-1	96.37%	89.24%	96.80%	96.23%	89.23%	96.40%
Barnes-2	90.96%	89.51%	97.80%	91.02%	89.83%	97.62%
Barnes-3	90.96%	89.05%	97.67%	90.95%	89.40%	97.56%
Barnes-4	90.86%	89.69%	97.62%	90.84%	88.89%	97.45%
Ocean-1	77.56%	89.24%	86.55%	75.29%	89.23%	83.88%
Ocean-2	75.22%	89.51%	92.24%	72.66%	89.83%	90.27%
Ocean-3	75.54%	89.05%	92.25%	72.83%	89.40%	90.18%
Ocean-4	75.23%	89.69%	92.24%	72.55%	88.89%	89.94%

from the larger shared I-Cache and fully-shared functional units. It gets higher performance than CASH and CMP for *equake*, *mgrid*, *crafty*, *gcc*, *gzip*, *facerec*, and *fma3d* programs. Among them, *crafty* and *gcc* are both memory-intensive and cpu-intensive program, the fully-shared structure of SMT permits them obtaining high performance. The program *gzip* and *fma3d* are memory-intensive, and also can benefit from the shared cache structure of SMT. In addition, program *equke*, *mgrid*, and *gzip* also get benefit from compensate usage of fully-shared functional units on SMT. Due to the same shared L1 cache structure, CASH shows the similar behavior as SMT but with a lower degree. Not sharing functional units as SMT, *mgrid*, *crafty*, *gcc*, *gzip*, *facerec* and *fma3d* on CASH get lower performance than on SMT, but on the contrary the co-exist programs, *art*, *apsi* and *galgel*, get higher performance on CASH than on SMT. CMP do not share resources as SMT and CASH, it outperforms SMT and CASH for some program due to the one cycle less access latency in D-Cache, and performs worse for others due to the lower hit rate in branch predictor and L1 cache.

With issue width increasing, the difference between SMT and CASH becomes less. This is mainly because CASH and CMP improve performance for the threads more than SMT. For some programs, *equake*, *mgrid*, *crafty*, *gcc*, *perlbmk*, and *fma3d*, CASH gets better

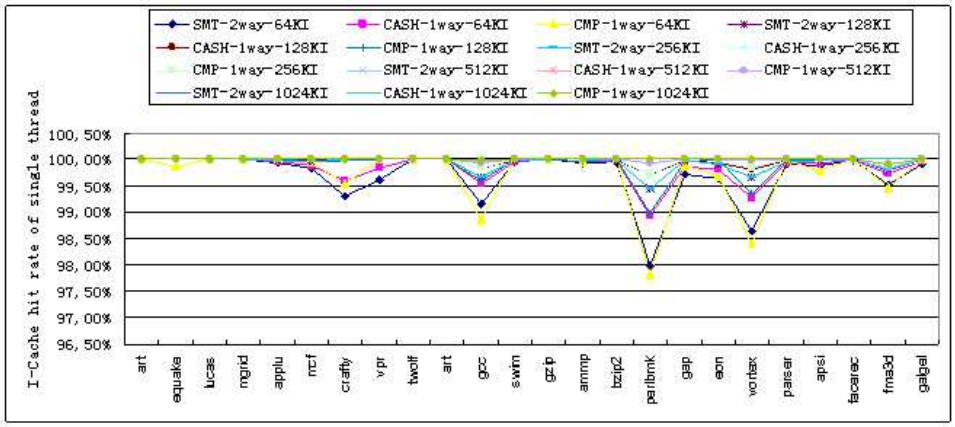
performance than SMT. The fully-shared method on SMT provides less benefit at high issue width than the low one.

Figure 27 gives the I-Cache hit rate of single thread in 2-thread workloads with different cache sizes. When changing the cache size, the hit rate changes mainly on some programs. These programs are *crafty*, *gcc*, *perlbmk*, *vortex*, *apsi*, and *fma3d*. The other programs reach the top point of hit rate when the size is total 64KB or 128KB, so they can not benefit from the size increasing. For the above programs they continue increasing their hit rates with the cache size enlarging, and as a result their IPC performances also increase continually that have been show in Figure 26. This trend fits in the all different issue widths. When comparing SMT, CASH, and CMP, CASH gets higher average hit rate than SMT when the total issue width is 4-way and 8-way, but lower average hit rate than SMT when the total issue width is 2-way. CMP shows different trend of hit rate with SMT and CASH. Due to unshared I-Cache in every single core, the hit rate of single thread only determined by the size of cache. In 2-thread running mode, the size dedicated to one thread on CMP is 32KB. For the programs that reach the top point of hit rate, all the three architectures get same hit rate for them. For the memory-intensive programs, *crafty*, *gcc*, *perlbmk*, *vortex*, *apsi*, and *fma3d*, CMP gets lower hit rate than SMT and CASH. But as a compensate, the co-exist programs, *vpr*, *swim*, *bzip2*, *parser*, *facerec*, and *galgel*, all get higher hit rates on CMP than on SMT and CASH. It is clearly that the shared policy on SMT and CASH helps the memory-intensive programs getting more space and higher hit rate, but the space for the co-exist programs are decreased and the hit rate of them also decreased.

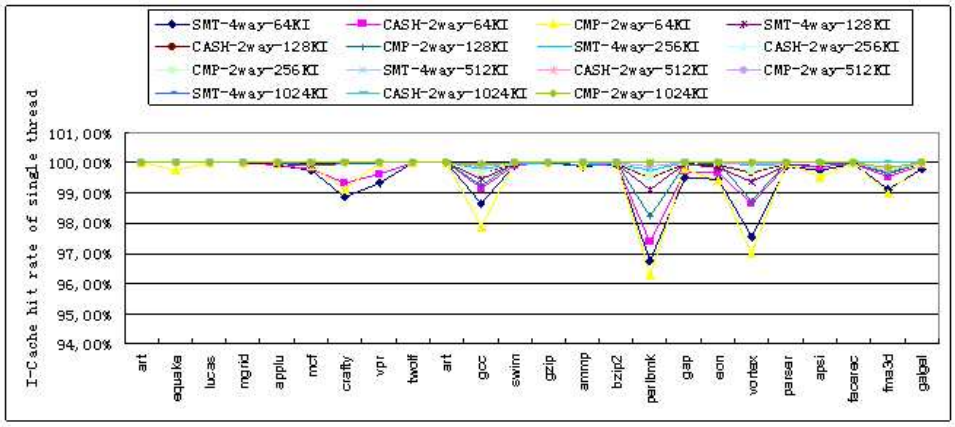
b. 4-thread workload performance

Figure 28 shows the single thread performance in 4-thread running mode with different I-Cache sizes. The difference between CMP and SMT, CASH is the 4-times less space of I-Cache dedicated to the thread in every single core on CMP. When the total size is 64KB, the size on CMP is only 16KB for one thread. According to the analysis results from above, many programs reach the top I-Cache hit rate when the size belonging to them equal or larger than 32KB, so for these programs the performance only has some difference when the total size is 64KB and 128KB. When the total size is larger than 128KB, the cache space belonging to one thread on CMP will larger than 32KB, such that there will no performance difference between the different larger sizes. For the programs that prefer to larger I-Cache size, *crafty*, *vortex*, *apsi*, *fma3d* and *perlbmk*, there will be some differences between different sizes. Normally, these programs improve their performance continually with the size increasing.

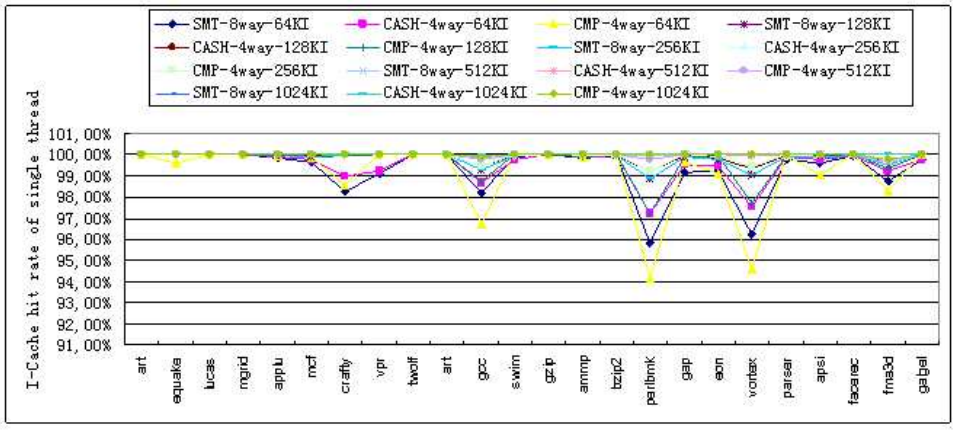
When the total issue width is 4-way, SMT can still benefits from the fully-shared policy for some programs, (applu in workload 13, *lucas* and *swim* in workload 14), but for the other programs it has little space to compensate using the total cache among threads. CASH has similar behavior with CMP for most of the programs. When the larger shared I-Cache provides a high hit rate and do not suffer much from the longer latency in D-Cache, CASH



(1) Total issue width is 2-way

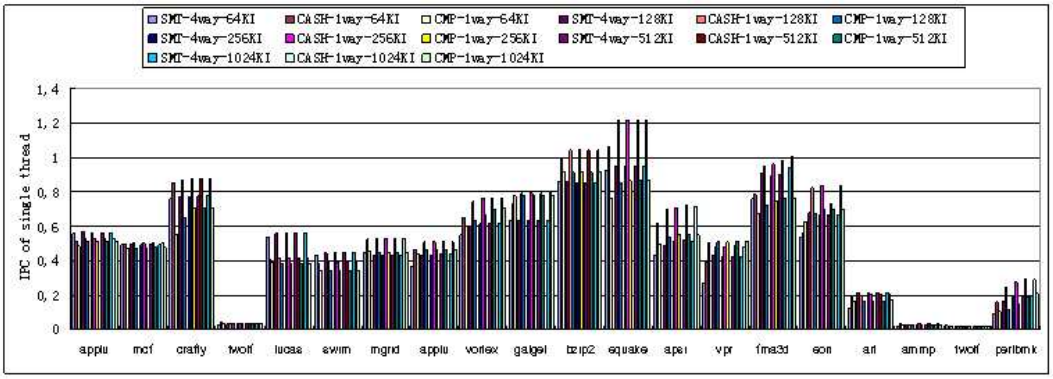


(2) Total issue width is 4-way

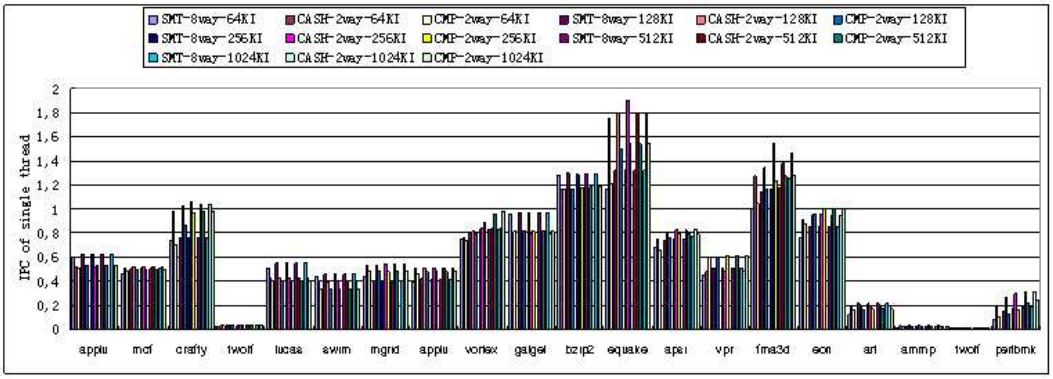


(3) Total issue width is 8-way

Figure 27: I-Cache hit rate of single thread in 2-thread workloads when changing the I-Cache size.



(1) Total issue width is 4-way



(2) Total issue width is 8-way

Figure 28: Performance of single thread in 4-thread workloads with different I-Cache sizes.

will get better performance than CMP, on the contrary if the longer latency plays the key role CASH will get worse performance.

When the total issue width is 8-way, every thread improves the performance in three architectures, but CASH and CMP have a higher degree than SMT. Due to the four times larger I-Cache on CASH, some programs, *crafty*, *applu* (workload 14), *vortex*, *equake*, *fma3d*, and *perlbmk*, get highest performance among the three architectures.

In average, CASH gets best performance, than CMP, and SMT is the last one.

Figure 29 shows the I-Cache hit rate of single thread running in 4-thread mode. Same as in 2-thread mode, the hit rate of different I-Cache size and different issue widths mainly occurs in some programs, *crafty*, *vortex*, *apsi*, *fma3d*, *eon*, and *perlbmk*. Among them, *eon* and *ammp* are effected very much by the co-existed *fma3d* and *perlbmk*. The other programs near reach to the top of hit rate at 64KB or 128KB size, such that they do not vary the hit rate with the cache size increasing. For the above mentioned programs, they increase their hit rate with the I-Cache enlargers in all the three architectures. Same as the 2-thread mode, these programs gets higher hit rate on SMT and CASH, but the co-exist programs get lower hit rate in them than on CMP if the dedicated size is too small to fulfill their requirements. When considering the issue width, the hit rate changes more at high issue width than that at lower one, but still hold the same trend as the lower one.

6.5.2 L1 D-Cache size

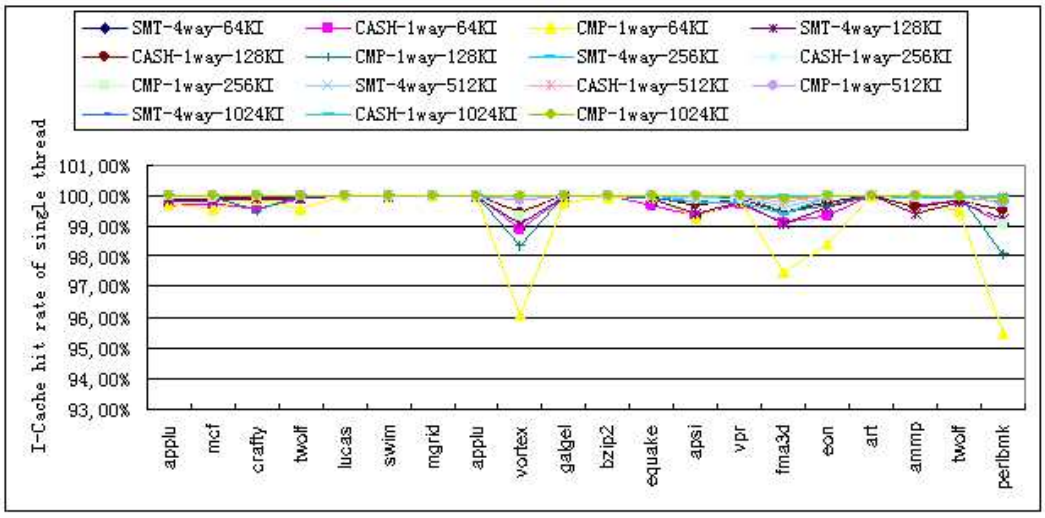
We also compare the performance of SMT, CASH and CMP with different D-Cache sizes. Same as the I-Cache experiment, the other hardware parameters in the three architectures are same as the base configurations.

a. 2-thread workload performance

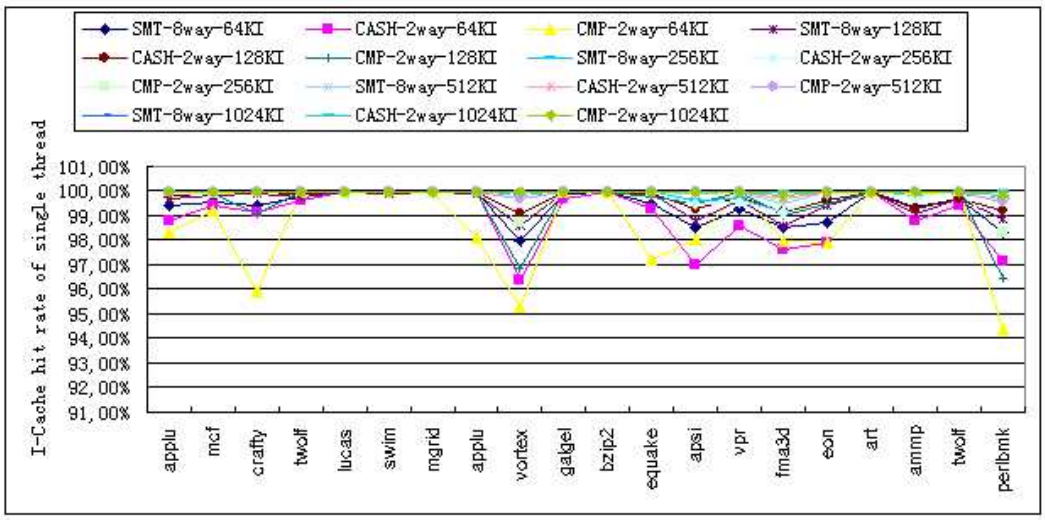
Figure 30 presents the single thread performance in 2-thread mode with different D-Cache size. We vary the size from 64KB to 1MB. On SMT and CASH, a totally shared D-Cache is provided for all the running threads in a workload, but on CMP a half of the total size is dedicated to every thread in every single core. In addition, different with on CMP, the access latency of D-Cache on SMT and CASH is one cycle longer than that on CMP.

With the cache size increasing, some of the programs (*art*, *equake*, *crafty*, *vpr*, *gcc*, *gzip*, *perlbmk*, *vortex*, *parser*, *apsi*, *facerec*, *fma3d*) improve their performance continually, the other programs change marginally or even no change. As mentioned before, except *equake* and *facerec*, all the other mentioned programs are memory-intensive programs, they prefer to a large D-Cache size. The shared D-Cache on SMT and CASH may give them opportunity to improve performance, but the longer access latency also may decrease their performance from that on CMP. Whether improving or decreasing determined by which part plays the key role during their executing.

In different issue widths, *art*, *lucas*, *applu*, *swim* and *fma3d* benefit more from shared structure than the sufferance from longer latency, and get better performance on SMT and

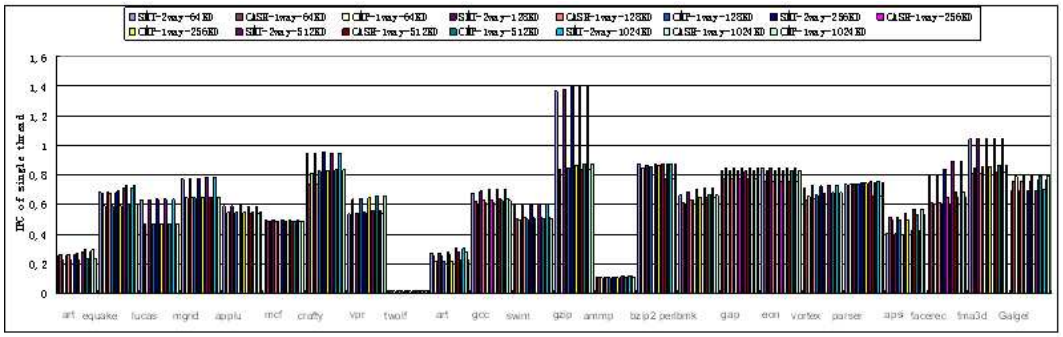


(1) Total issue width is 4-way

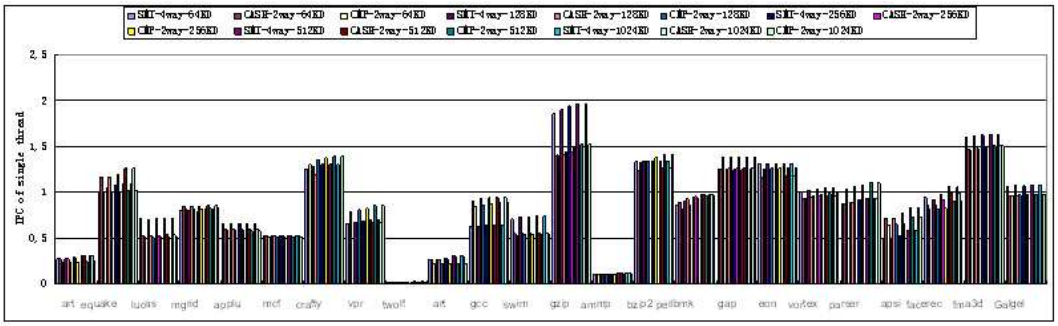


(2) Total issue width is 8-way

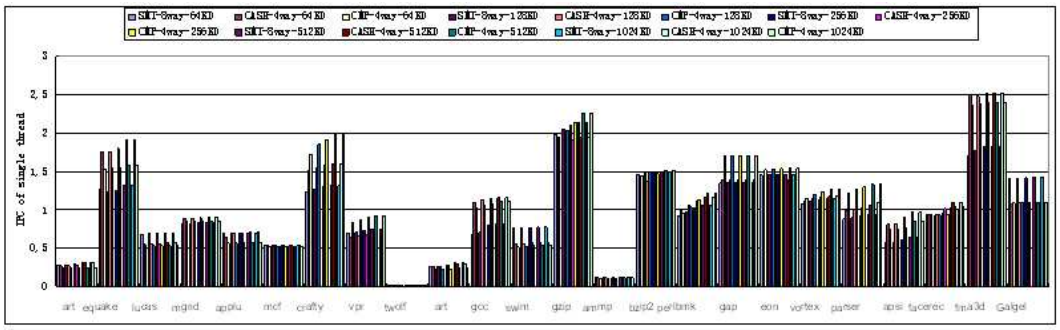
Figure 29: I-Cache hit rate of single thread in 4-thread workloads when changing the I-Cache size.



(1) Total issue width is 2-way



(2) Total issue width is 4-way



(3) Total issue width is 8-way

Figure 30: Performance of single thread in 2-thread workloads with different D-Cache sizes.

CASH than on CMP at most cases, and SMT gets better performance than CASH due to the fully-shared functional units. The program *crafty*, *gap*, *vortex*, and *parser* suffer more from the latency on SMT and CASH, and get worse performance in them than on CMP when the total issue width is 4-way or 8-way, and on the contrary when the total issue width is 2-way. The program *equake*, *mgrid*, *gcc*, *perlbmk*, and *apsi* get best performance on SMT 2-way and on CASH when the total issue width is 4-way or 8-way. And the program *vpr*, *gzip*, *bzip2*, and *eon* get worst performance on CASH.

When the total issue width is 2-way, *equake*, *mgrid*, *crafty*, *gcc*, *gzip*, *facerec*, and *fma3d* benefit very much from the fully-shared policy of hardware resources on SMT, and get best performance than CASH and CMP, but with the total issue width increasing to 4-way and 8-way the benefit becomes less or diminishes.

Figure 31 shows the D-Cache hit rate of these experiments. At all issue widths configurations all the three architectures have very near D-Cache hit rate for most of the programs. There are only a little differences of hit rate for program *art*, *swim*, *perlbmk*, *apsi* and *galgel* among different architectures and among different cache sizes. For *swim* and *galgel*, it is clear that the variance of hit rate comes from the co-exist programs *gcc* and *fma3d*.

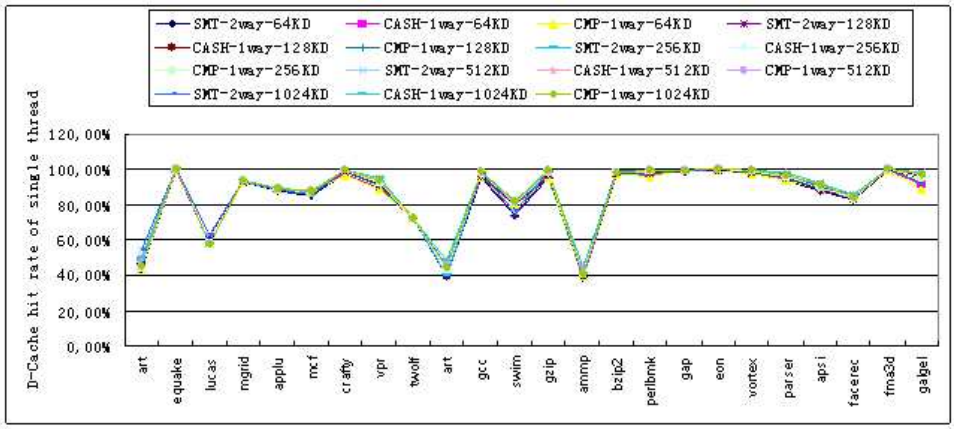
b. 4-thread workload performance

Figure 32 shows the single thread performance running in 4-thread mode with different D-Cache sizes. As mentioned before, the D-Cache size dedicated to every thread in a CMP core is four times less than the total size of cache on SMT and CASH, and the access latency in the last two is one cycle longer than on CMP.

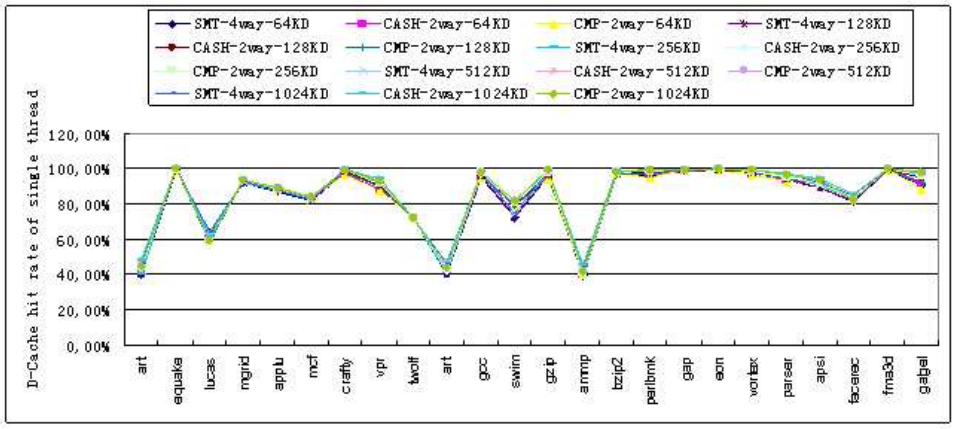
With the size increasing, most of the programs, *crafty*, *vortex*, *bzip2*, *equake*, *apsi*, *vpr*, *fam3d*, *eon*, *art*, and *perlbmk*, improve the performance continually in three architectures. Other programs only change a little with the size varies.

With issue width increasing, all the programs in three architectures improve performance continually. Among the three architectures, CASH improves the performance with highest degree for most of the programs. For workloads *vortex-galgel-bzip2-equake*, there are seldom conflicts on the using of shared float-point or long latency unit (only *galgel* has relative high frequency of long latency instructions) among threads. And *vortex* and *bzip2* are both memory-intensive and cpu-intensive programs. The shared larger D-Cache permits CASH improving the performance of them dramatically. At the same time, with the memory and cpu intensive programs running together SMT could not benefit much from the fully-shared policy of hardware but still suffers from the longer access latency, as a result the performance of workload is lower than CASH and CMP. With a non-shared policy, CMP gets the mid performance for the workloads between CASH and SMT. For the other four-thread workloads, there is a similar explanation of the performance variance in three architectures.

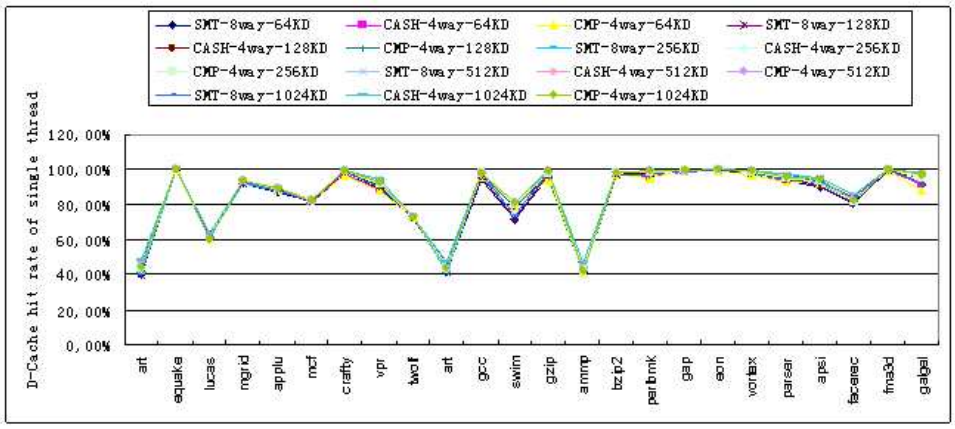
Figure 33 shows the D-Cache hit rate of single thread in 4-thread running mode with different cache size. With different cache sizes and different issue widths, the hit rate changes mainly on some programs, *mcf*, *lucas*, *galgel*, *apsi*, *vpr*, *art*, and *perlbmk*. For the other programs, the hit rate only varies a little or do not vary at all when change the D-Cache



(1) Total issue width is 2-way

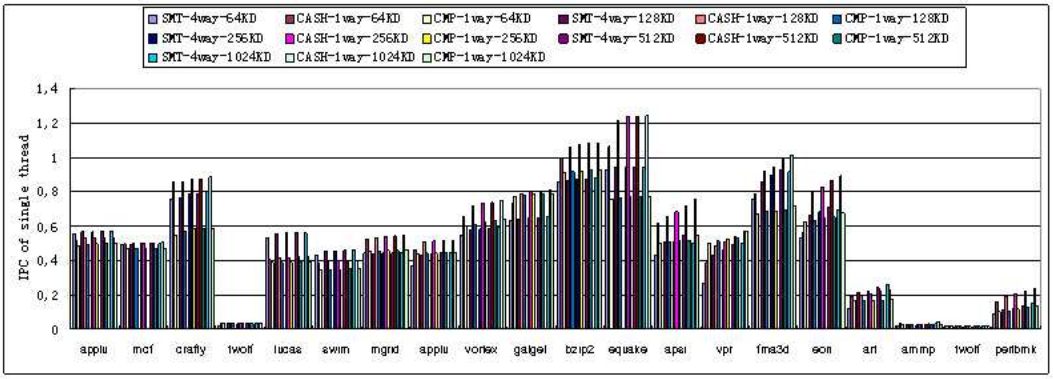


(2) Total issue width is 4-way

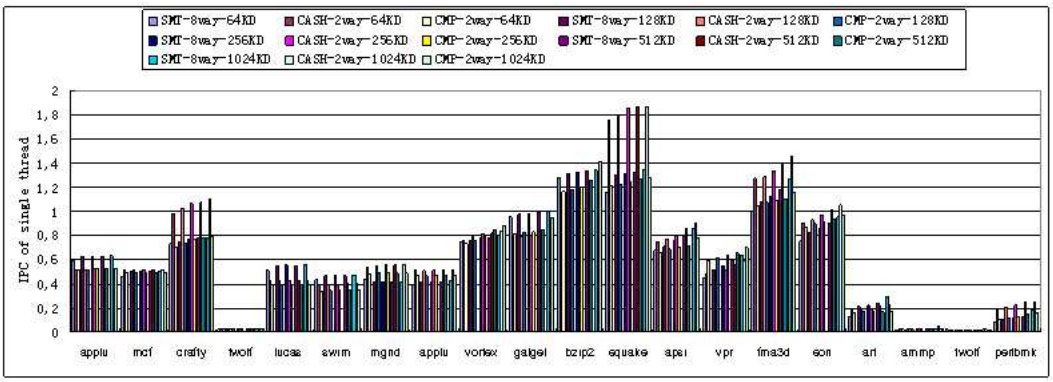


(3) Total issue width is 8-way

Figure 31: Single thread D-Cache hit rate in the workloads when changing the D-Cache size.

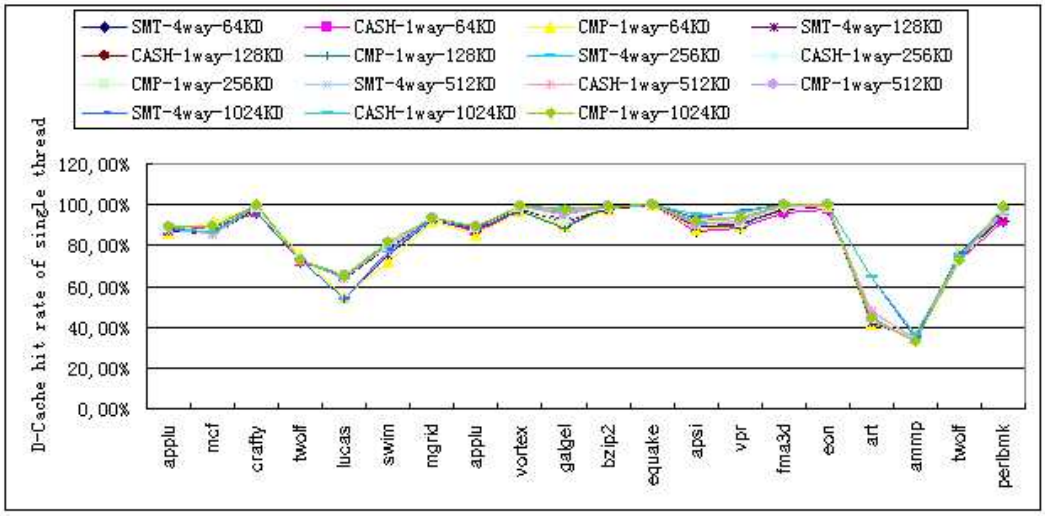


(1) Total issue width is 4-way

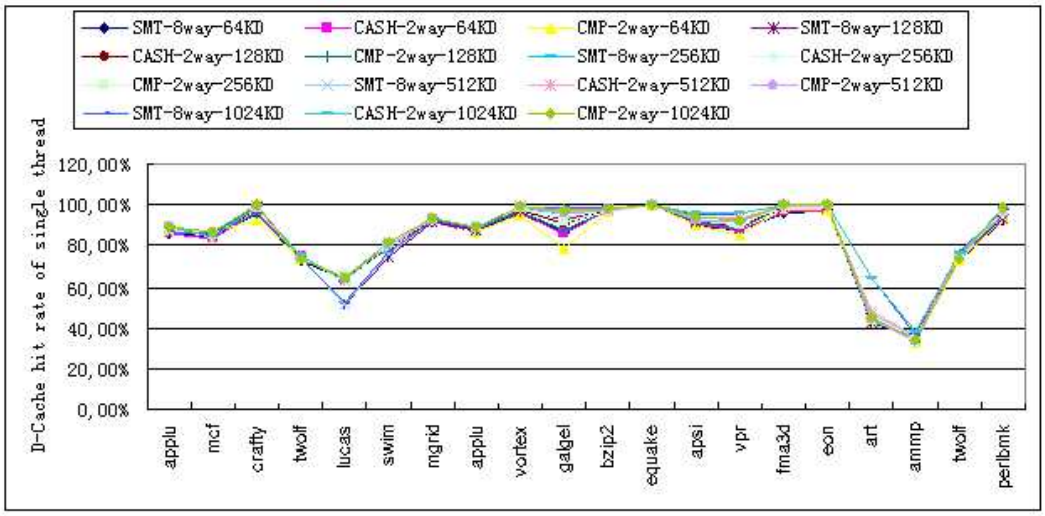


(2) Total issue width is 8-way

Figure 32: Performance of single thread in 4-thread workloads with different D-Cache sizes.



(1) Total issue width is 4-way



(2) Total issue width is 8-way

Figure 33: I-Cache hit rate of single thread in 4-thread workloads when changing the D-Cache size.

size or issue widths. Same as before, *mcf*, *apsi*, *vpr*, *art*, and *perlbmk* are memory intensive programs, they prefer to a larger cache size and their hit rates are also affected by the size.

When comparing the hit rate of SMT, CASH and CMP, CMP gets highest hit rate for program *applu*, *mcf*, *lucas*, *mgrid*, *equake*, *fma3d*, and *perlbmk*; CASH gets highest one for program *crafty*, *twolf*, *swim*, *apsi*, *art*, and *ammp*; then SMT gets the highest one for program *vortex*, *galgel*, *bzip2*, *vpr* and *eon*. Most of the memory intensive programs have higher hit rate on CASH or SMT than on CMP. And on the contrary the other programs have higher hit rates on CMP than on CASH and SMT.

6.6 Varying cache associativity in CASH processor

6.6.1 L1 I-Cache associativity

We investigate the performance variance with different I-Cache associativity on CASH, and also compare it with SMT and CMP. All the three architectures use the same base configuration, except the associativity of L1 I-Cache changes.

a. 2-thread workload performance

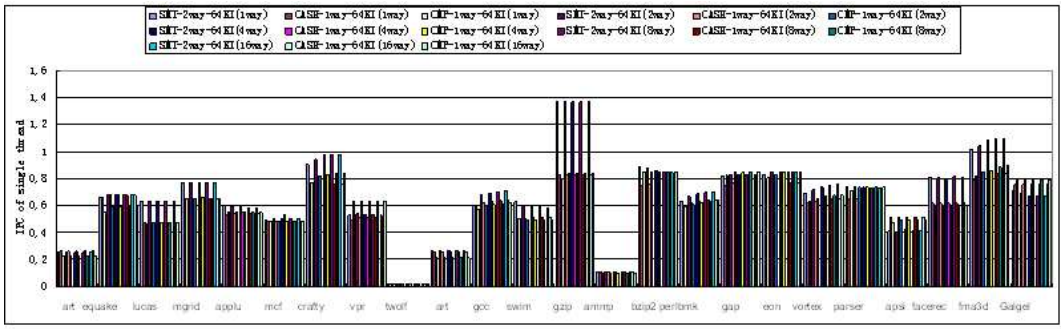
Figure 34 shows the single thread performance in 2-thread running mode with different I-Cache associativities on SMT, CASH, and CMP. Similar as the experiment results of cache size, with the I-Cache associativity varying only some programs, *crafty*, *gcc*, *perlbmk*, *gap*, *eon*, *vortex*, and *fma3d*, change the performance continually, and the other programs have little change or no change at all. Most of above mentioned programs (except *gap* and *eon*) are memory intensive, thus a higher cache associativity causes a higher hit rate for the programs in all the three architectures, and the performances are also improved as a result. The *gap* and *eon* are cpu intensive programs, and the varying of cache associativity also cause their hit rate and performance changing continually.

With issue width increasing, all the programs improve the performance at three architectures, but CASH and CMP at a faster speed.

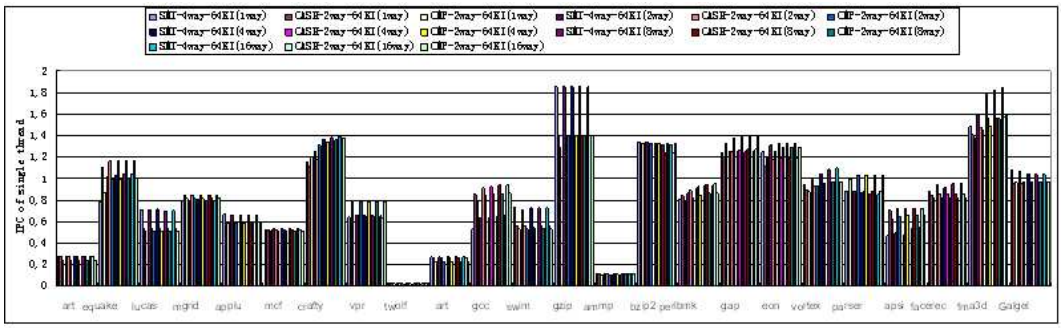
Figure 35 shows the I-Cache hit rate of single thread with different cache associativities. Corresponding to the performance variance in Figure 34, cache hit rate varies very much at the above mentioned programs. Normally, a higher associativity leads to a higher hit rate. In addition, program *equake*, *applu*, *mcf*, *vpr*, *gzip*, *parser*, and *apsi* also change their hit rate with the associativity varying although the final performance of them do not change very much. Among them, except *equake* and *applu*, the other programs are also memory intensive, and the varying of associativity also causes the hit rate changing continually.

With issue width increasing, the hit rate of single program decreases continually at most cases. This is because the more instructions at a larger issue width can be finished than at a small one, such that causes more instructions to be fetched in and leads to more cache misses and lower hit rate.

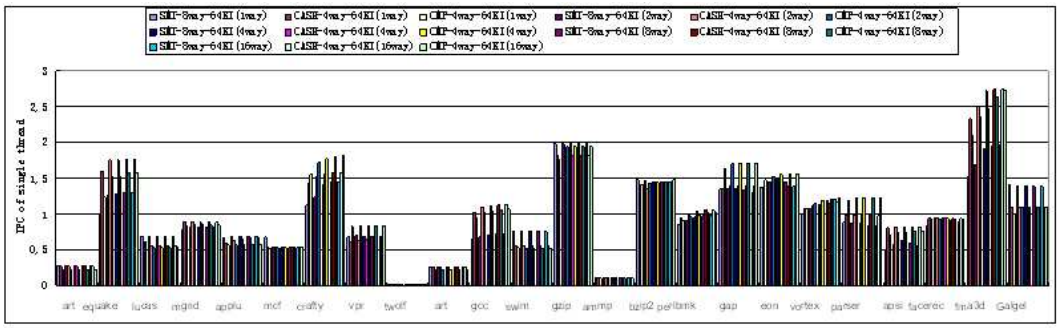
When comparing the hit rate of SMT, CASH, and CMP, CASH gets highest hit rate at near half of the programs (and most of them are memory intensive), and CMP is also (most



(1) Total issue width is 2-way

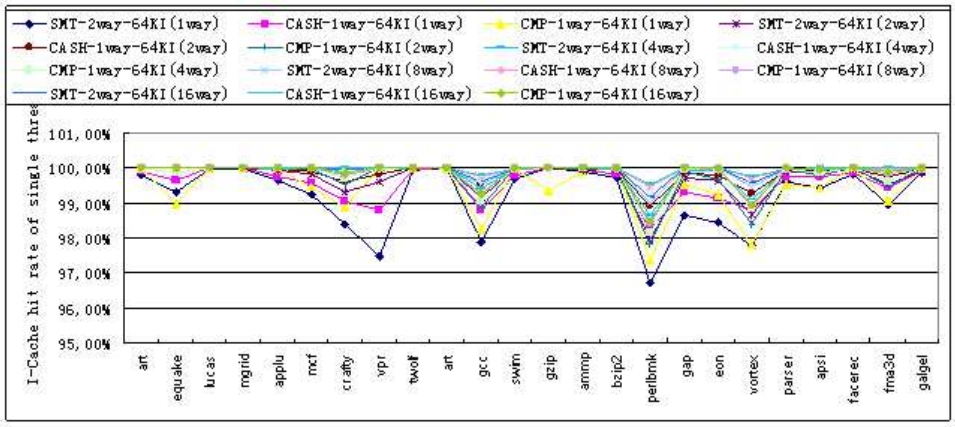


(2) Total issue width is 4-way

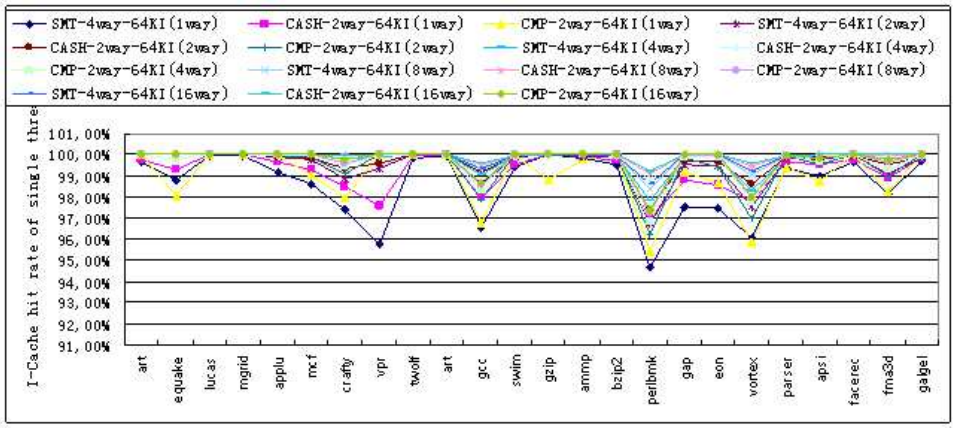


(3) Total issue width is 8-way

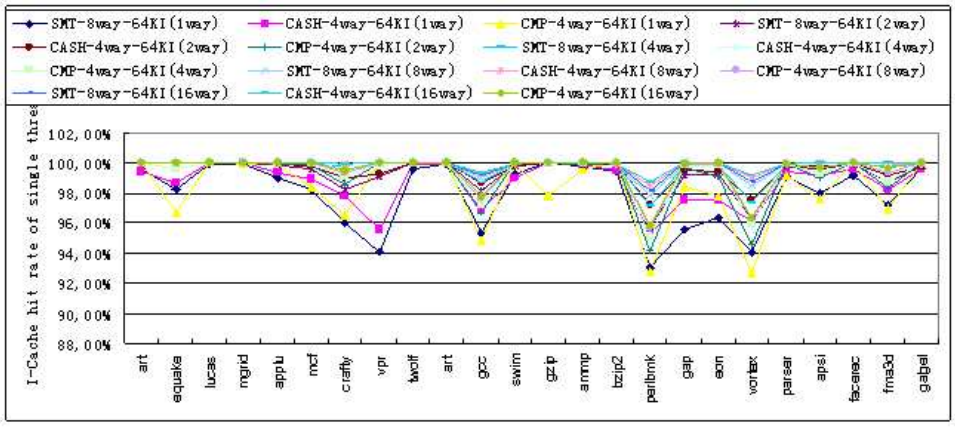
Figure 34: Performance of single thread in 2-thread workloads with different I-Cache associativities.



(1) Total issue width is 2-way

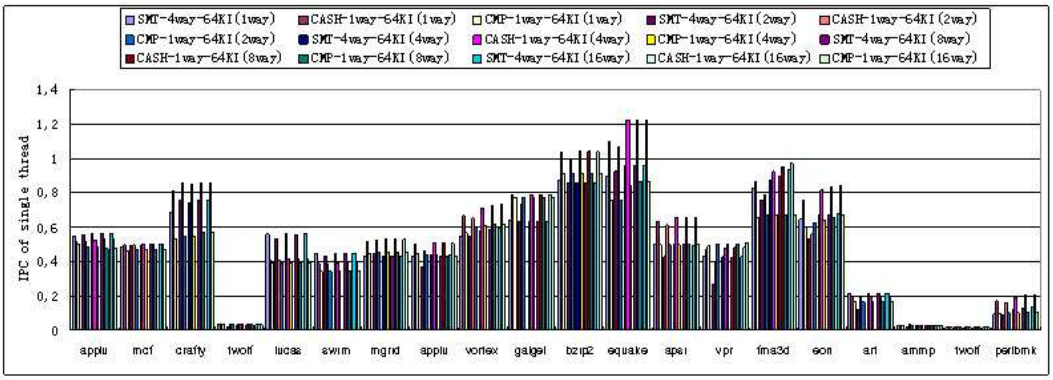


(2) Total issue width is 4-way

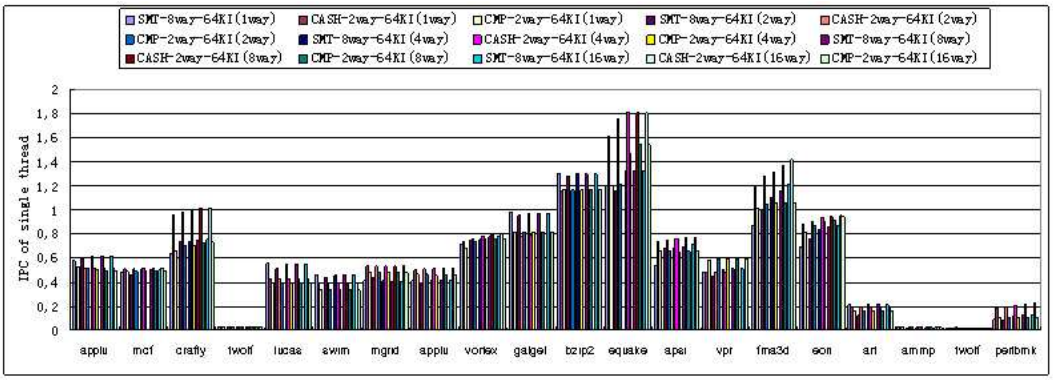


(3) Total issue width is 8-way

Figure 35: I-Cache hit rate of single thread in 2-thread workloads when changing the I-Cache associativity.



(1) Total issue width is 4-way



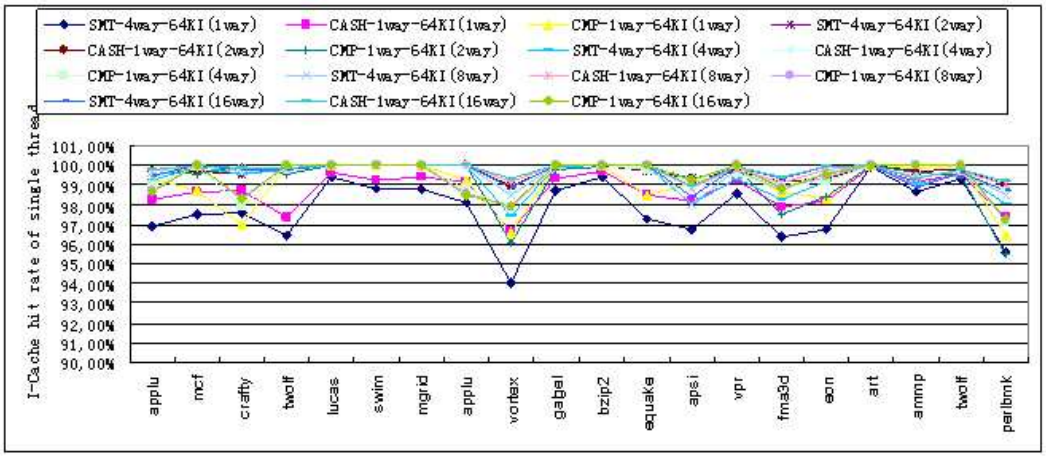
(2) Total issue width is 8-way

Figure 36: Performance of single thread in 4-thread workloads with different I-Cache associativities.

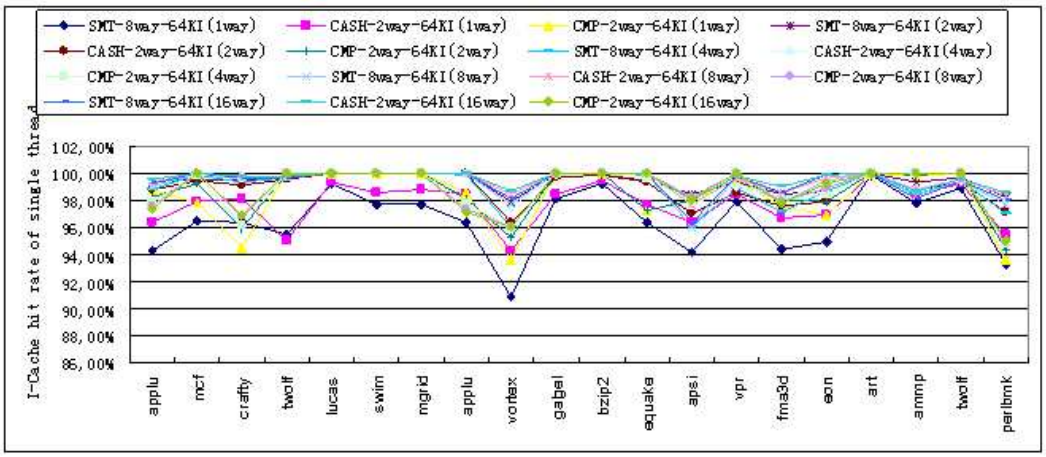
of the other programs), SMT gets the lowest hit rate at most cases.

b. 4-thread workload performance

Figure 36 shows the single thread performance in 4-thread running mode with different I-Cache associativities. With the associativity varying, program *applu*, *mcf*, *crafty*, *vortex*, *equake*, *fma3d*, *eon* and *perlbmk* change their performances continually. The other programs change a little or do not change with the different associativities. Among the above programs, except *applu*, *equake*, and *eon*, the programs are all memory intensive, and the different associativity causes different hit rate of them, and the performance varies as a results. Normally, with the associativity increasing, the performance of memory intensive programs increases continually. For *applu*, with the performance increasing of *mcf* and *crafty* in the



(1) Total issue width is 4-way



(2) Total issue width is 8-way

Figure 37: I-Cache hit rate of single thread in 4-thread workloads when changing the I-Cache associativity.

same workload, it decreases its performance continually. The *eon* is a cpu intensive program, the associativity changing in I-Cache also causes its hit rate and performance varying.

With issue width increasing, all the programs improve their performances in three architectures, but CASH and CMP improve the performance with a faster speed than SMT. In average, CASH gets best performance in the two different total issue widths.

Figure 37 shows the I-Cache hit rate of single thread with different cache associativities. Same as the performance, most changes of hit rate occur at the memory intensive programs and some co-existed programs. With the associativity increasing, the hit rates of most programs increase corresponding. And with issue width increasing, the hit rates of most programs decrease. When comparing the hit rates on CASH and CMP, CASH gets the better one for all the workloads. At most cases, SMT gets the lowest hit rate among the three architectures.

6.7 L1 D-Cache associated sets

a. 2-thread workload performance

Figure 38 shows the single performance in 2-thread mode with different D-Cache associativities. When increasing the associativity, only a few programs increase their performances on CMP, and only crafty increases its performance on SMT and CASH. For the other programs, the varying of D-Cache associativity does not affect their performance in three architectures.

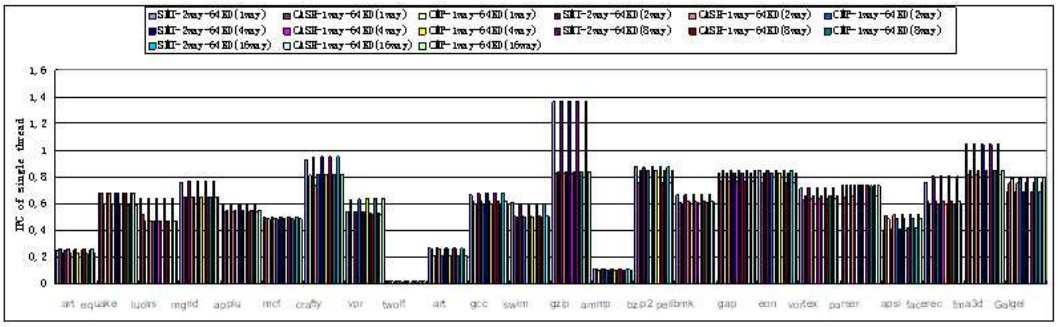
Figure 39 presents the hit rate of single thread in 2-thread running mode. Corresponding to the performance, there are seldom variances in hit rate for most of the programs when the associativity increases. With different issue widths, the hit rate also only changes marginally or does not change at all. When comparing the three architectures, the difference of hit rate among them is very little for most of programs, there are only small difference in program *lucas*, *mcf*, *art*, *swim*, and *galgel*.

b. 4-thread workload performance

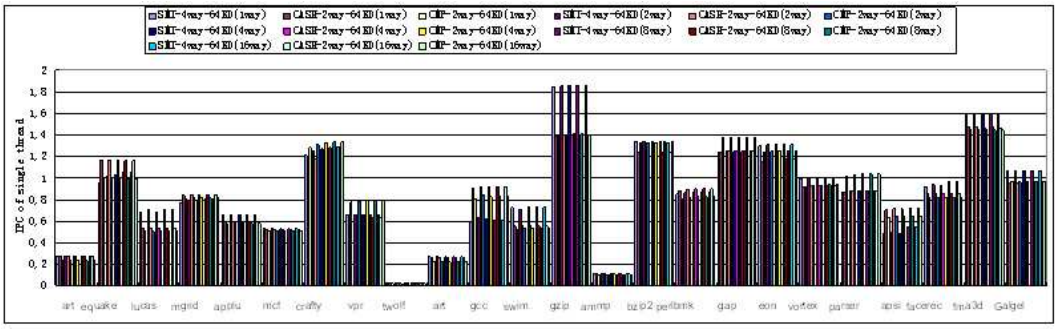
In figure 40, it gives the single thread performance of 4-thread workloads when change the D-Cache associativity. Same as 2-thread cases, there are only a little performances improvement in program *crafty*, *equake*, *apsi*, *vpr*, *fma3d* and *eon* at high issue widths. For the other programs, there are near no change when the associativity of D-Cache increases. In figure 41, we show the cache hit rate of single thread in these experiments. When the associativity is increasing, the hit rates of near half of the programs increase in all the three architectures. And the differences of hit rate in three architectures are very small. With different issue widths, the hit rate shows a same trend of differences. Comparing the hit rate of three architectures, CMP gets a little higher hit rate than the other two at directly-mapping method, but has little difference at other set associated mapping methods.

7 Conclusion

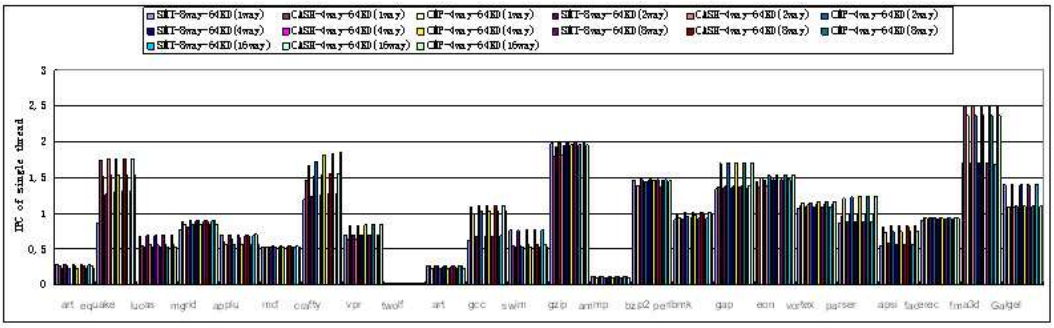
CASH parallel processor (for *CMP And SMT Hybird*) is a possible intermediate design points for on-chip thread parallelism in terms of design complexity and hardware sharing. It retains resource sharing as SMT when such a sharing can be made non-critical for implementation,



(1) Total issue width is 2-way

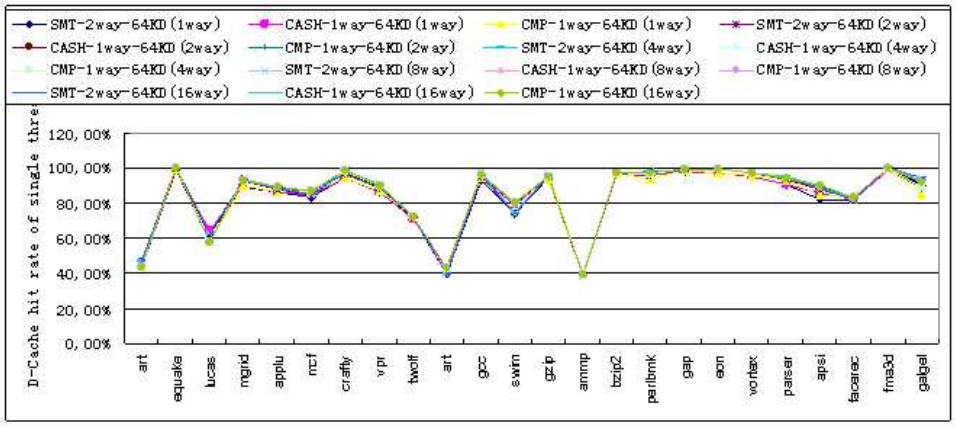


(2) Total issue width is 4-way

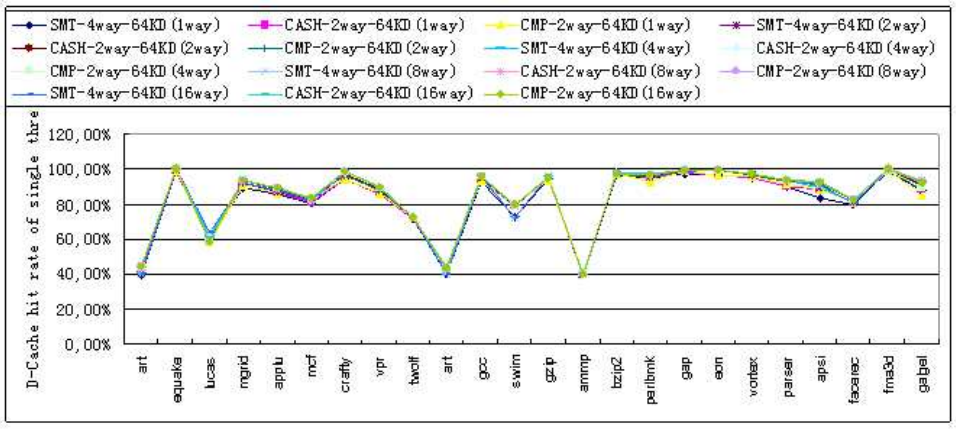


(3) Total issue width is 8-way

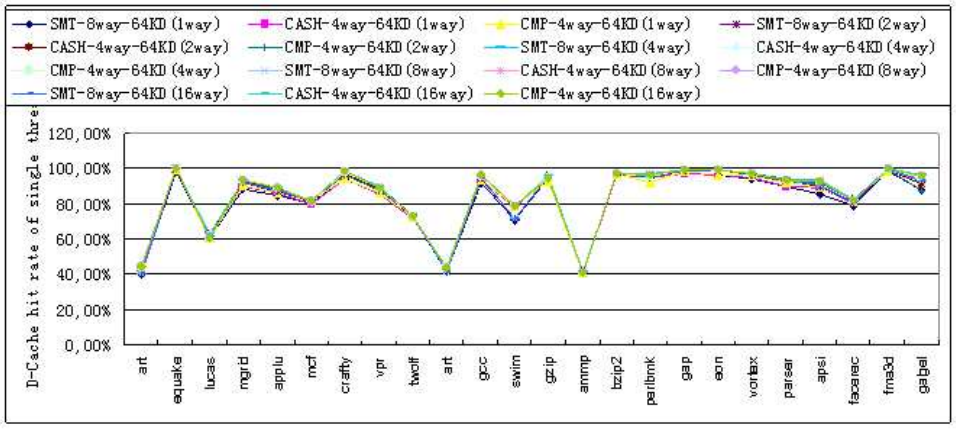
Figure 38: Performance of single thread in 2-thread workloads with different D-Cache associativity.



(1) Total issue width is 2-way

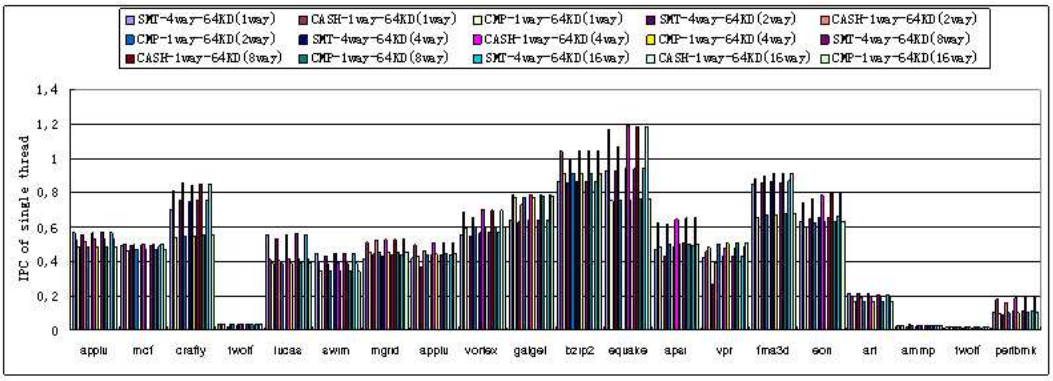


(2) Total issue width is 4-way

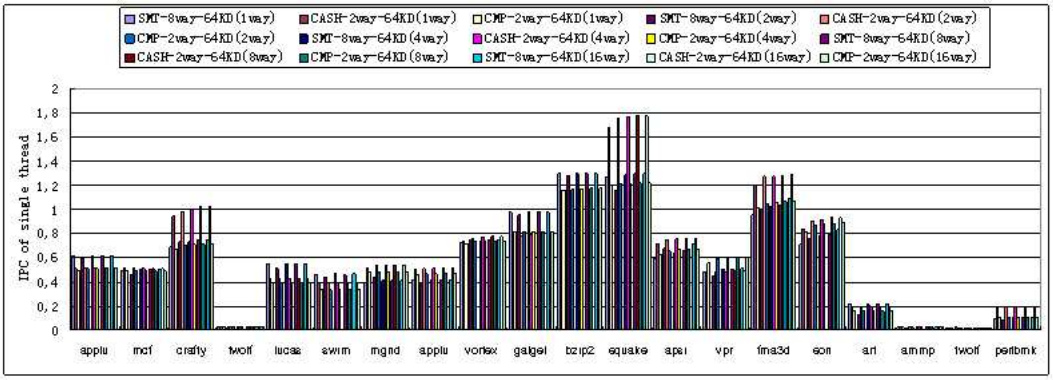


(3) Total issue width is 8-way

Figure 39: D-Cache hit rate of single thread in 2-thread workloads when changing the D-Cache associativity.

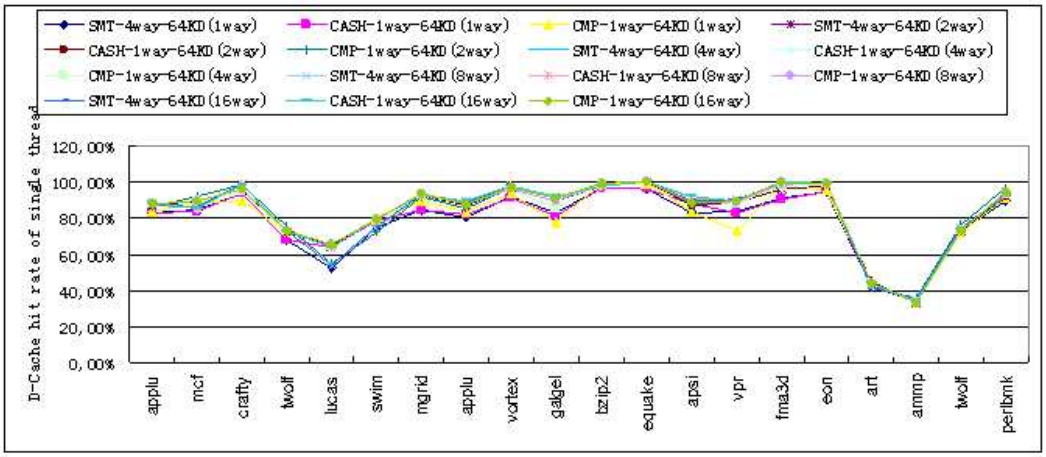


(1) Total issue width is 4-way

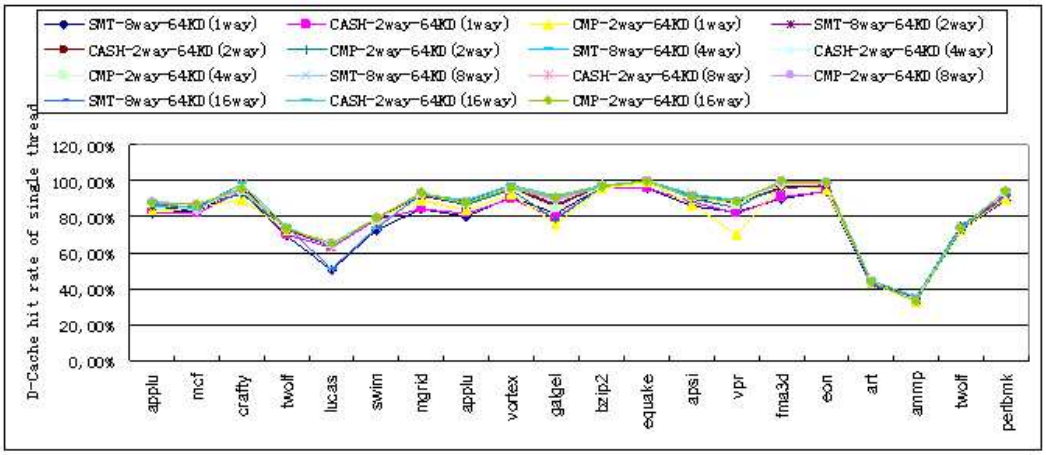


(2) Total issue width is 8-way

Figure 40: Performance of single thread in 4-thread workloads with different D-Cache associativity.



(1) Total issue width is 4-way



(2) Total issue width is 8-way

Figure 41: D-Cache hit rate of single thread in 4-thread workloads when changing the D-Cache associativity.

but resource splitting as CMP wherever resource sharing leads to a superlinear increase of the implementation hardware complexity.

In this paper, we explore the CASH design space in details using an execution-driven simulator. To compare the performance of similar architectures, we also implement a CMP and SMT simulator using the same base simulator. The workloads include twelve two threads and five four threads multi-program benchmarks, and two parallel workloads from SPLASH-2.

The simulation results show that for single program workload a CASH processor with a shared L1 cache and branch predictor can obtain a better performance than the CMP processor. When comparing CASH with SMT, the latter can obtain a more higher performance than the former because the fully-shared hardware resources.

At most cases, SMT can benefit from the fully-shared policy of hardware resources and gets better performance than CASH and CMP at lower issue width; but with higher issue width, CASH and CMP improve the performance of programs more higher than SMT, and CASH gets the best average performance, then CMP, and SMT is the last;

For multi-program workloads, the CASH processor shows a great potential to improve the performance from the CMP processor. This includes three cases. 1) The workload which seldom uses and competes the shared unit among the threads performs better on CASH than on CMP when the hardware resource in a single core can provide a sufficient service for the thread running in it and the shared L1 cache can give more data to the total threads, but performs worse when 2) the threads in a workloads that do not compete the shared unit but suffer from the longer access time in L1 D-Cache; 3) the threads in a workload that get a sufficient service from the shared unit although competing to use among them, and benefit more from the larger branch predictor and L1 shared cache at the same, get a higher relative improvement than CMP; 4) the threads in a workload, that compete seriously to use the shared unit and the benefit from the shared branch predictor and L1 cache can not compensate the performance losing from the competence and the long access time in L1 cache, get a worse performance than the CMP.

When comparing the performance with context switching, CASH performs better than CMP as one can expect because the shared L1 cache structure and branch predictor do not lose information for the threads when they moves from one core to another, whereas CMP suffers from information losing with the context switching.

For parallel workloads, normally the behaviors of the parallel threads are similar. They compete to use the shared unit at the same time and suffer from the longer access time of L1 cache, but benefit from the shared branch predictor and L1 cache very much. If the number of long latency instructions belong to a workload is small, it gets a higher relative improvement on CASH than on CMP; On the contrary, a lower improvement is obtained.

When increasing the cache size, the performance improves continually on CASH. The difference of performance between two different associativities of L1 cache is small.

References

[1] R. Dolbeau and A. Seznec. CASH: Revisiting hardware sharing in single-chip parallel processor. IRISA-IRINA technical report PI-1491.

[2] M. D. Powell, M. Gomaa, and T.N. Vijaykumar. Heat-and-Run: Leveraging SMT and CMP to manage power density through the operating system. In Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems, pages: 260-270, October 2004.

[3] V. Krishnan and J. Torrellas. A clustered approach to multithreaded processors. In Proceedings of International Parallel Processing Symposium, pages 627-634. Mar. 1998

[4] J. Burns and J. L. Gaudiot. Area and system clock effects on SMT/CMP processors. In Proceedings of International Conference on Parallel Architecture and Compilation Techniques, pages 211-218. 2001

[5] R. Kumar, N. P. Jouppi, and D. M. Tullsen. Conjoined-core chip multiprocessing. In Proceedings of the 37th International Symposium on Microarchitecture, pages: 195-206. Dec. 2004

[6] J. Collins and D. M. Tullsen. Clustered multithreaded architectures - pursuing both IPC and cycle time. In Proceedings of IPDPS. Apr. 2004

[7] A. Seznec. Analysis of the O-GEometric History Length branch predictor, In Proceedings of 32nd Annual International Symposium on Computer Architecture (ISCA'05), pages: 394-405. 2005.

[8] D.M. Tullsen. Simulation and Modeling of a Simultaneous Multithreading processor. In the 22nd Annual Computer Measurement Group Conference. December. 1996

[9] John L. henning. SPEC CPU2000: Measuring CPU Performance in the New Millennium. IEEE/COMPUTER, COMPUTER, pages: 28-35. July 2000 Issue.

[10] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: characterization and methodological considerations. In Proceedings of the 22nd International Symposium on Computer Architecture, pages 24-36. Santa Margherita Ligure. Italy. June 1995.

[11] Y. Sazeides and T. Juan. How to compare the performance of two SMT microarchitectures. In Proceedings of IEEE International Symposium on Performance Analysis of System and Software, pages:180-183. November 2001.

Contents