The Impact of Cache Organisation on the Instruction Issue Rate of a Superscalar Processor

Lucian Vintan   Cristian Armat
University "Lucian Braga" of Sibiu
Sibiu-2400, Romania
vintan, armat@cs.sibiu.ro

Gordon Steven
University of Hertfordshire, UK.
Hatfield, Hertfordshire, U.K. AL10 9AB
email: G.B.Steven@herts.ac.uk

Abstract

Much of the research on multiple-instruction-issue processor architecture assumes a perfect memory hierarchy and concentrates on increasing the instruction issue rate of the processor either through aggressive out-of-order instruction issue or through static instruction scheduling. In this paper we describe a trace driven simulation tool that we have developed to quantify the impact of the memory hierarchy on the performance of a superscalar processor that we have developed to support static instruction scheduling. We describe some initial studies performed using our simulator. As well as examining the more conventional split cache configurations, we also quantify the performance impact of using a unified cache. Finally, we examine the benefits of using two-level caches and victim caches.

keywords

Memory Hierarchy, Caches, Superscalars, Multiple Instruction Issue

1. Introduction

High-performance superscalar processors attempt to increase performance by issuing multiple instructions in each processor cycle. Parallel instruction issue can be realised either dynamically through hardware at run time or statically through software at compile time.

The hardware approach is characterised by aggressive out-of-order instruction issue, dynamic branch prediction and escalating hardware complexity [1]. The software approach involves systematically reordering the code at compile time to assemble groups of independent instructions that can be safely issued and executed in parallel at run time [2]. Although this instruction scheduling approach is often associated with VLIW architectures [3], it is equally applicable to superscalars. Furthermore, since independent instructions are already grouped together at compile time, the processor can now adopt a simple in-order instruction issue policy. The cost of this hardware simplification is the increased compilation time and the increased code size that is an inherent by-product of the instruction scheduling process.

A high instruction issue rate requires a high memory bandwidth. A sustained issue rate of four, for example, is likely to require peak instruction fetch and issue rates of eight or more. In addition, multi-port data cache structures are required to provide sufficient data operand bandwidth. The performance of a multiple-instruction-issue superscalar processor therefore crucially depends on the performance of its memory hierarchy. In this paper we describe the development of a trace driven simulation system that we will use to quantify the impact of the memory hierarchy on the performance of a superscalar processor designed to support static instruction scheduling.

The trace driven simulator has been developed as part of the HSA (Hatfield Superscalar Architecture) project [4]. The ambitious objective of this project is to use static instruction scheduling to achieve an order of magnitude speed up over a traditional RISC processor that issues a maximum of one instruction in each processor cycle. Encouragingly, numerous studies [5, 6] conclude that programs typically contain significantly more inherent parallelism than we will need to achieve our objective.

Unfortunately, aggressive instruction scheduling inevitably leads to code expansion. Typically, a factor of between 1.5 and two is involved [7], but ill-chosen heuristics or scheduling techniques can lead to more spectacular code explosion. This code expansion is the software equivalent of the escalating hardware complexities associated with aggressive out-of-order instruction issue. One of the uses of our trace driven simulator will be to quantify the impact of this code expansion on performance.

In this paper we present preliminary studies that quantify the impact of the memory hierarchy on the performance of the HSA architecture. Traditionally, the cache hit rate is used to measure memory hierarchy performance. However, this metric gives no direct indication of processor performance. Since we are interested in the overall execution time, we use instruction issue rate as our principle metric. This metric measures directly how successful our processor is in issuing multiple instructions in each cycle. As well as the usual split cache arrangement, we also investigate the impact of using a unified cache system. We also recognise that it is often impractical to design a first-level cache system that is both fast enough to satisfy the processor timing and bandwidth requirements, yet large enough to ensure a satisfactory hit rate. For this reason, we investigate the benefits of using two-level cache structures and victim caches, both of which minimise the impact of misses in a first-level cache of inadequate size.

2. The HSA architectural model

HSA is a load and store architecture with a simple RISC instruction set [4]. Distinctive features include guarded execution of all instructions and a simplified addressing mechanism. The following four-stage pipeline is assumed:

IF:	Instruction Fetch
ID/RF:	Instruction Decode / Register Fetch
EX:	Execute
WB:	Write Back

In the first pipeline stage, a group of instructions is fetched from the instruction cache into an Instruction Buffer. In the second stage, instructions in the Instruction Buffer are decoded and register operands are obtained from the general-purpose register file. At the end of the cycle, multiple instructions are issued to functional units for execution. Instructions are always issued in program order. In the third stage, instructions are executed. Finally, in the fourth stage, results are returned to the register file or are bypassed directly to functional units for immediate re-use.

All instructions are assigned unit latencies and all pipeline stages are assumed to complete in one cycle. The simplified addressing mechanism used in HSA [8] does not require an addition to generate an effective memory address. As a result LD and ST instructions can be sent directly to the data cache at the end of the Instruction Decode stage.

In HSA a generalised branch delay mechanism is provided. Since branches are normally resolved in the ID/RF stage, a minimum of one branch delay slot is therefore required. However, a recent study [9] suggested that dynamic branch prediction could usefully be added to the HSA architecture. In this paper we therefore assume that perfect branch prediction, with a 100% success rate, is provided.

As well as the more usual separate data and instruction cache organisation, we also quantify the impact of using a unified cache organisation. A cache access time of one cycle is assumed throughout. If a data access is made to a unified cache, the instruction fetch process will be stalled for one cycle. However, since instruction issue can continue from the Instruction Buffer, the rest of the pipeline is not necessarily stalled.

3. Tace driven simulation

This study uses trace driven simulation to investigate the impact of caches on processor performance and, in particular, on the instruction issue rate. Direct mapped caches are used throughout. HSA code is first generated for our benchmarks using an optimising Gcc compiler generated specifically for our architecture. The code produced is then executed on the HSA instruction-level simulator which in turn produces an instruction trace. This trace consists of every taken branch and every memory reference instruction executed by the processor.

The trace driven simulator, developed at the University of Sibiu by two of the authors, requires two input files: the assembly language file produced by our Gcc compiler and the trace file generated by the instruction-level simulator. The branches in the traces allow the program path to be reconstructed, while the memory reference instructions record the memory address of every load and store instruction executed and allow dependencies between memory references to be respected.

Each cache configuration investigated is characterised by the following five parameters:

FR:	Fetch Rate
IBS:	Instruction Buffer Size
Cache Size:	Size of the direct mapped cache in bytes.
IR_{max}:	Maximum Instruction Issue Rate
MP	Miss Penalty; cycles required to load a cache block from main memory

The Fetch Rate (FR) specifies the maximum number of instructions that can be fetched into the Instruction Buffer in one cycle. FR is also taken as the default block size. Since each cache read may only access data from a single cache block, FR instructions will only be fetched when the first word of a block is addressed. In contrast, after a branch to the last instruction in a block only a single instruction will be obtained. Throughout the study, block allocation is performed on write misses, and writes to the cache follow a write back policy.

The IBS size determines the maximum number of instructions that can be buffered pending issue. A new group of instructions is only loaded into the Instruction Buffer if the whole group can be accommodated.

In this study, the instruction issue rate replaces the cache miss rate as the primary performance metric. IR_{max} places a limit on the number of instructions that can be issued in parallel to functional units. The simulator will also issue a maximum of two memory reference instructions per cycle. As is usual in HSA configurations, the Fetch Rate is de-coupled from the Issue Rate. In this paper we use a high fetch rate in conjunction with a unified cache to reduce the impact of instruction fetch stalls. Elsewhere, we have used a Fetch Rate of twice the Issue Rate to improve the performance of our static instruction scheduler [7].

RAW (Read after Write) or true data dependencies are respected throughout by the simulator. In contrast, output dependencies and anti-dependencies are ignored. Neither of these dependencies is likely to cause a stall in an in-order issue processor with unit instruction latencies.

4. Simulation Results

4.1 Benchmarks Characteristics

Eight integer benchmarks - *bubble*, *sort*, *perm*, *puzzle*, *queens*, *matrix*, *tree* and *tower* - are used throughout. Although relatively short, these programs are computationally intensive and have high dynamic instruction counts. A number of the benchmarks use recursion, with *perm*, *tower* and *tree*, in particular, being heavily recursive. The dynamic instruction distribution of the benchmarks is fairly typical: 40% of the instructions are arithmetic, logical or shifts; 13% are relational; 18% are loads; 12% are stores and 17% are branches.

While small caches are completely filled by the benchmarks, the cache block utilisation falls steadily as the cache size is increased. For example, with a 512 byte unified cache, on average 98% of the cache blocks are used. This figure falls steadily to 73% with an 8k cache and to only 30% with a 64K byte cache. *Matrix* and *tree* are notable exceptions and still use 49% and 53% of the available blocks in a 64K byte cache.

One disadvantage of a unified cache is that useful instructions will be overwritten in the cache by data and vice versa increasing the miss rate. We define a cache interference ratio as the percentage of misses caused by this overwriting process. With a cache size of 256 bytes, the average interference ratio is as high as 28.5, with both *queens* and *tower* exhibiting interference ratios over 40%. However, the interference ratio falls sharply as the cache size is increased, to 3.8% with a cache size of 4K bytes and to only 0.27% with 8K bytes.

4.2 Comparison between unified and separate caches

In this section we compare the use of a unified cache with the more usual separate instruction and data cache configuration. Since the provision of two caches allows parallel access to instructions and data, this configuration might be expected to yield higher performance in all cases. However, a unified cache has several advantages. Firstly, it will yield a slightly higher hit rate than a split cache [10]. Essentially a unified cache can vary the proportion of instructions and data cached, giving it a slight edge. Secondly, instructions can still be issued from the Instruction Buffer while the data cache is being accessed. Finally, a single unified cache is cheaper to implement than two separate caches.

To ensure a fair comparison, a unified cache with N entries is compared with a split cache where both the instruction and data caches have N/2 entries. The following parameters are used throughout: FR = 8, IR = 4, IBS = 32 and MP = 10. Surprisingly, with small caches of 256 bytes and 2K bytes, a split cache yields no performance advantage (Fig.1). With 256 bytes, the harmonic mean instruction issue rate is 0.49 instructions per cycle with a split cache and 0.48 instructions per cycle with a unified cache, while with a 2K byte cache the issue rates are 0.97 instructions per cycle and 0.99 instructions per cycle respectively. With small cache sizes, the higher hit rate of the unified caches is therefore able to compensate for stalls in the instruction fetch process. Not unexpectedly, however, the benefits of separate caches dominate with larger cache sizes. For example, with an 8K byte cache, the split cache model delivers an instruction issue rate of 1.52, while the unified cache can only deliver an issue rate of 1.2. Nonetheless, even with 8K caches, four programs, *matrix*, *queens*, *puzzle* and *sort*, still perform better with a unified cache.

4.3 Unified Cache Models

In this section we evaluate various unified cache models. First we explore the instruction issue rate as a function of cache size (Fig. 2). Parameters are as follows: FR = 4, IR = 4, IBS = 8 and MP =10. As expected the harmonic mean instruction issue rate rises steadily from 0.37 instructions per cycle with a 256 byte cache to 1.32 instructions per cycle with a 16K byte cache. At the same time the average cache miss rate falls from 47.5% to 3.22%. Nonetheless, the maximum issue rate of 1.32 instructions per cycle is disappointingly low for a model with a cache miss rate of around 3% and with a maximum instruction issue rate of four.

Increasing the fetch rate, FR, improves performance slightly. For example, with a cache size of 512 bytes and IBS = 32, increasing FR from 4 to 8 boosts the harmonic mean issue rate from 0.49 to 0.56. A further increase of FR to 16, however, has no impact, with the issue rate falling back again very slightly. Since FR also determines the cache block size, these changes largely reflect the impact of increasing the block size and the resultant improvement of the hit rate. Reducing the instruction buffer size, IBS, from 32 to 8 has little impact with the issue rate falling marginally from 0.486 to 0.478.

Issue rates also fall steadily as the miss penalty, MP, is increased. For example, with a cache size of 4K bytes, the issue rate falls from 1.28 to 0.91 and then to 0.77 as MP is increased from 10, first to 15, and finally to 20. Other parameters are: FR = 8, IBS = 16 and IR_{max} = 4

Finally, we examine two examples that illustrate the inherent drawbacks of the unified cache model. First, a low performance model has the following parameters: FR = 4, IBS = 32, IR = 4, MP = 10 and a unified cache size of 256 bytes. With this small cache size, the performance of a unified cache model is virtually identical to an equivalent split cache model. However, on average the processor is only able to fetch instructions successfully in 13.25% of the available cycles. The high miss rate of the cache is therefore crippling performance.

In the second model, a high performance model, the cache size is increased to 16K bytes and FR to 8. However, although the miss rate is around 3%, the processor still only fetches instructions in half of the available cycles. Data cache conflicts ensure that the performance will be significantly poorer than a comparable split cache model. As a result, unified caches are only competitive with split caches when the inadequate size of the cache is already severely limiting performance.

4.4 Split Cache Models

In this section we turn our attention to the split cache model, often known as a Harvard architecture. First we examine the impact of the instruction cache size using a fixed data cache size of 2K bytes (Fig.3). Other parameters are FR = 8, IBS = 16, IR_{max} = 4 and MP = 15. As expected, the harmonic mean instruction issue rate increases steadily from 0.51 with a 128 byte instruction cache to 1.07 with a 1K byte cache. At the same time, the average instruction cache miss rate falls from 34.42% to 0.06%.

Further improvements can be achieved by increasing the size of the data cache. The instruction cache size is fixed at 512 bytes while other parameters are unchanged (Fig.4). Issue rates now range from 0.51 with a 256 byte data cache to 1.46 with a 16K byte data cache, while the average data cache miss rate falls from 47.8% to 4.0%. In spite of the low average final miss rate, *puzzle*, with a miss rate of 22% and *tree*, with a miss rate of 7.2%, continue to perform badly. It is therefore no surprise that the issue rate can be boosted to 2.36 by using a 1K byte instruction cache in conjunction with a large 64K byte data cache. Significantly, similar issue rates were never achieved with a unified cache configuration.

As in the case of the unified cache model, increasing the fetch rate, FR, tends to increase the instruction issue rate. For example, with an instruction cache of 512 bytes, and a data cache of 2K bytes, increasing FR from 4 to 8 and then to 16 increases the issue rate from 0.69 to 0.91 and finally to 1.10. Again these results reflect the use of a larger block size. Raising the maximum issue rate from 4 to 8 also improves performance slightly. For example, with an instruction cache of 512 bytes and a data cache of 2K bytes, the instruction issue rate increases from 0.90 to 0.93. Without instruction scheduling, the opportunities to issue more than four instructions in parallel are therefore limited. Finally, the issue rate is also strongly affected by the miss penalty, MP. For example, with an instruction cache of 512 bytes, and a data cache of 2K bytes, the instruction issue rate falls from 1.13 to 0.90 and finally to 0.75 as MP is increased from 10 cycles to 15 cycles and finally to 20 cycles.

4.5 Two-Level Cache Structures

Although increasing the size of the instruction and data caches reduces the miss rate, it also increases the cache access time. This may in turn increase the processor cycle time or even increase the cache access time to two cycles. Caches with two cycle access times must themselves be pipelined if they are to deliver data effectively. Furthermore, all branch mispredictions suffer an additional penalty cycle, and instructions wishing to operate on data loaded from the data cache must wait an additional cycle before they are issued.

Two-level caches are increasingly seen as a way of avoiding these difficulties [10]. The first-level caches can then be small enough to allow one-cycle access times and can meet the data bandwidth requirements of the processor. At the same time larger second-level caches, also placed on the processor chip, allow the miss penalty to be drastically reduced - as long as there is a hit at the second level - and therefore allow the processor to maintain a reasonable instruction issue rate.

We investigated various two-level cache configurations using a first-level instruction cache size of 256 bytes, a first-level data cache size of 1K and a unified second level cache with a latency of 3 cycles (Fig.5). Other parameters are: FR = 8, IBS = 16, IR_{max} = 4 and MP = 15. The harmonic mean instruction issue rate increased from 0.77 to 1.27 as the size of the second-level cache was increased from 2K bytes to 32K bytes. In the previous section with an instruction cache size of 1K bytes and a data cache size of 2K bytes - both larger than in the above configuration - we were only able to achieve an issue rate of 1.07 instructions per cycle. Similarly with an instruction cache of 512 bytes and a data cache of 8K bytes, an issue rate of only 1.26 was achieved. Clearly, a carefully designed two-level cache can be used to limit the size and therefore access time of the first-level caches without significantly degrading processor performance.

4.6 Victim Caches

Finally, we simulated the impact of adding a victim cache to our models. A victim cache [11] is a small fully associative cache located between the cache and main memory A block displaced from the main cache is initially moved to the victim cache instead of being returned immediately to memory. A data access will therefore achieve a hit whenever the required data is held in either the main cache or the victim cache. Hits in the victim cache result in the matching block being restored to the main cache. Since the displaced block is transferred to the victim cache in the usual way, a hit in the victim cache will result in an exchange of blocks between the victim cache and the main cache.

A victim cache is likely to reduce the number of conflict misses [12] in a cache. It is therefore likely to achieve greater success with one of our unified cache models. We therefore added a variable sized victim cache to a unified cache of 2K bytes (Fig.6). Both LRU and random replacement in the victim cache were simulated . On balance random replacement proved slightly more effective and is therefore used here. Other parameters are: FR = 8, IBS =16 and IR = 4. As the size of the victim cache was increased from 0 to 512 bytes, the harmonic mean instruction issue rate increased steadily from 0.98 to 1.23. At the same time the average cache hit rate improved from 82% to 88.9%.

The highest issue rate of 1.23 is marginally better than the issue rate achieved in section 4.2 with a cache size of 8K bytes, four times the cache size used here. Even allowing for the additional system complexity and the overhead of associative access to the victim cache, a victim cache promises to be a cost effective method of improving the cache hit rate.

5. Conclusion and discussion

In this paper we have used trace driven simulation to quantify the impact of the memory hierarchy on the performance of an in-order superscalar processor. Throughout we have used the instruction issue rate as our main metric in preference to the more traditional cache miss rate.

As well as the more usual Harvard architecture, we also studied unified cache models. Surprisingly, with a small cache size, there was little performance difference between a split cache and a unified cache. However, more detailed examination suggested that this unexpected behaviour arose because the issue rate was severely restricted by an inadequate cache size and consequent low hit rate. With larger cache sizes, a split cache convincingly outperformed a unified cache.

With either cache configuration, inadequate cache sizes can cripple processor throughput, and issue rates of under 0.4 instructions per cycle were recorded with both models. However, in many superscalar designs it may be undesirable to increase the size of the cache, since doing so will either increase the processor cycle time or will lead to multi-cycle cache access times. In these cases we show that both two-level caches and victim caches can be effective alternatives to larger caches.

Traditionally, superscalar designs provide instruction and data caches of equal size. In contrast, this study suggests that - at least with these benchmarks - a larger data cache would be more effective. Since other studies confirm that the traditional equal partitioning results in a higher miss rate in the data cache [12], an unbalanced design may be worth considering for more general use.

Although a maximum instruction issue rate of four was used throughout, the highest sustained issue rate recorded was a disappointing 2.36. Furthermore increasing the issue rate beyond four will only increase this gap. Clearly higher instruction issue rates can only be achieved through either compile time instruction scheduling or through out-of-order instruction issue. In future, we therefore hope to use our trace driven scheduler to quantify the impact of cache misses on the execution of benchmarks that have been scheduled for parallel instruction issue by the HSA instruction scheduler. In particular, we wish to determine whether the speedups of between three and four currently being achieved by our instruction scheduler [9] can be sustained when the memory hierarchy is also modelled. We then wish to see if we can achieve comparable performance by adding out-of-order instruction issue to our trace driven simulator. Finally, we wish to determine if out-of-order issue can further improve the execution time of code that has already been scheduled for parallel execution.

References

1) Patt, Y N, Patel, S J, Evers, M, Friendly, D H and Stark, J "One Billion Transistors, One Uniprocessor, One Chip," Computer, September 1997, pp51-57.
2) Rau, B R and Fisher, J A "Instruction-Level Parallel Processing: History, Overview and Perspective," The Journal of Supercomputing, Vol.7, No. 1/2, 1993, pp9-50.
3) Fisher, J A "Very Long Instruction Set Architectures and the ELI-512," 10th Annual Symposium on Computer Architecture, June 1983, pp140-150.
4) Steven G B, Christianson D B, Collins R, Potter R. and Steven F L "A Superscalar Architecture to Exploit Instruction Level Parallelism," Microprocessors and Microsystems, Vol.20, No 7, March 1997, pp391-400.
5) Lam, M .S and Wilson, R P "Limits of Control Flow on Parallelism," 19th Annual International Symposium on Computer Architecture, Gold Coast, Australia, May 1992, pp46-57.
6) Potter R and Steven G B "Investigating the Limits of Fine-Grained Parallelism in a Statically Scheduled Superscalar Architecture," 2nd International Euro-Par Conference Proceedings, Vol.2, Lyon, France, August 1996, pp779-788.
7) Steven G B and Collins R "Instruction Scheduling for a Superscalar Architecture," Proceedings of the 22nd Euromicro Conference, Prague, September 1996, pp643-650.
8) Steven F L, Adams R G, Steven G B, Wang L and Whale D J "Addressing Mechanisms for VLIW and Superscalar Processors," Microprocessing and Microprogramming, Vol.39, Numbers 2-5, December 1993, pp75-78.
9) Egan, C., Steven, F.L. and Steven, G.B. "Delayed Branches versus Dynamic Branch Prediction in a High-Performance Superscalar Architecture," Euromicro97, Budapest, September 1997.
10) Przybylski S A. "Cache Design A Performance Directed Approach," Morgan Kaufmann, 1990.
11) Jouppi N P "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully Associative Cache and Pre-Fetch Buffers," Proceedings of 17th Annual International Conference on Computer Architecture, Seattle, May 1990, pp364-373.
12) Patterson, D A and Hennessey J L "Computer Architecture A Quantitative Approach," Morgan Kaufmann, 2nd edition 1996.

Fig.1 Spilt versus Unified Instruction Caches

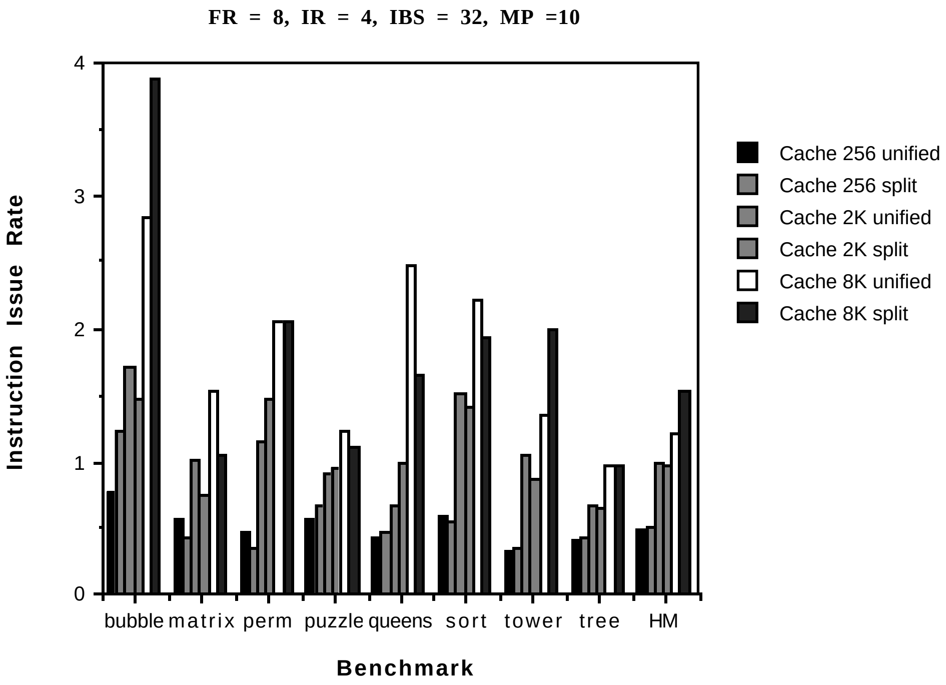


Fig.2 Unified Cache Instruction Issue Rate

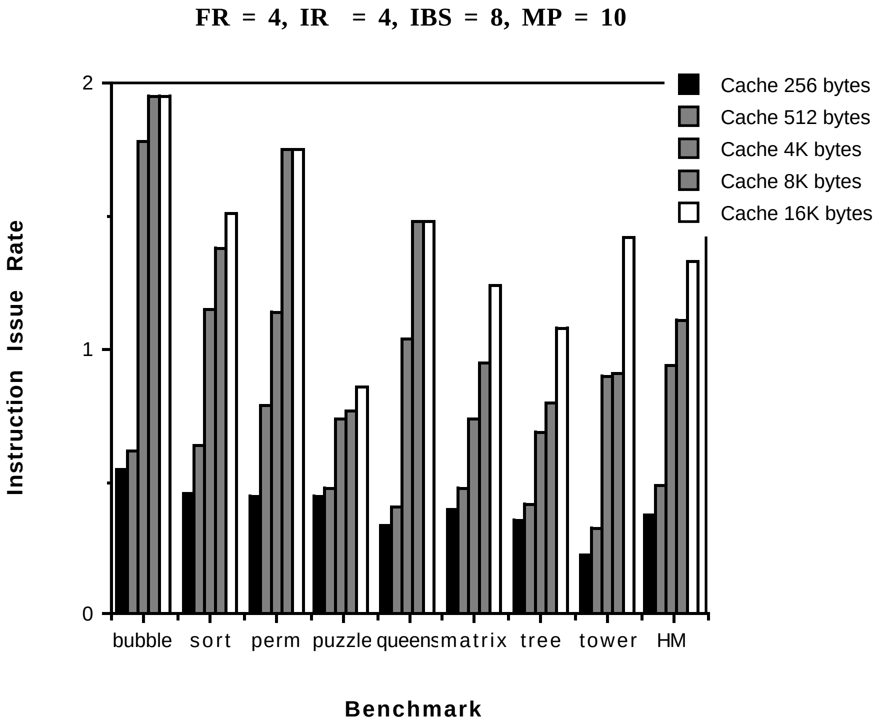


Fig. 3 Split Cache Model: Issue Rate versus Instruction Cache Size

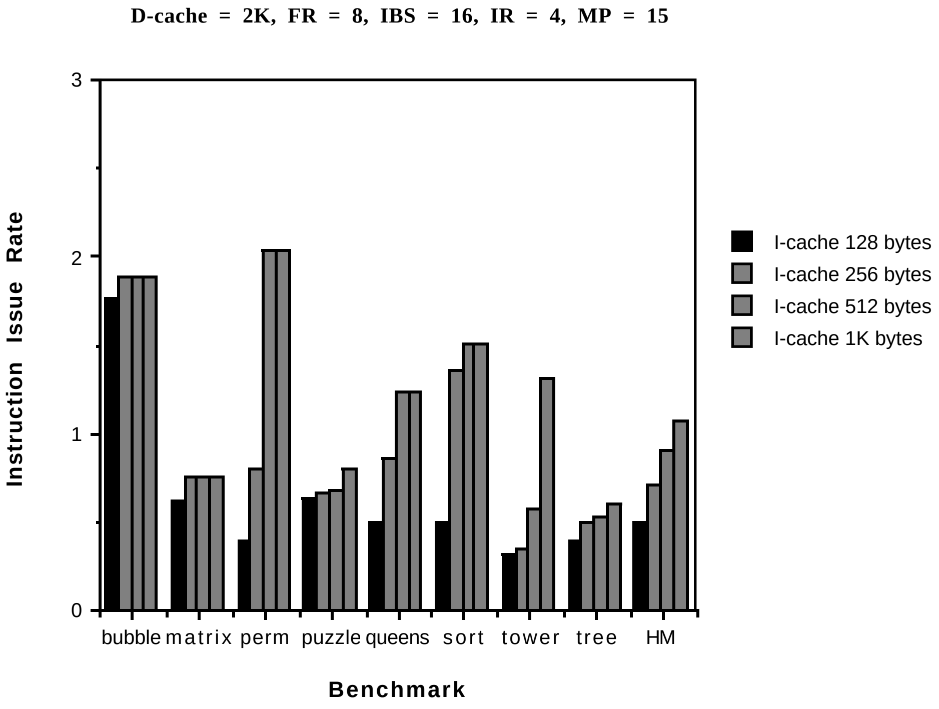


Fig. 4 Split Cache Model: Issue Rate versus Data Cache Size

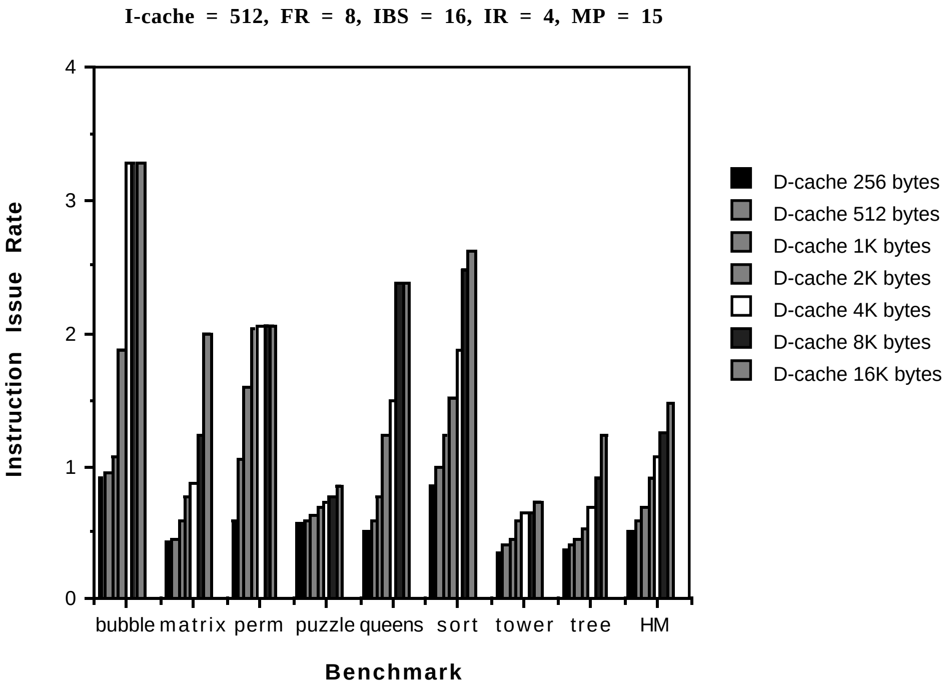


Fig. 5 Two-Level Cache: Issue Rate versus Cache Size

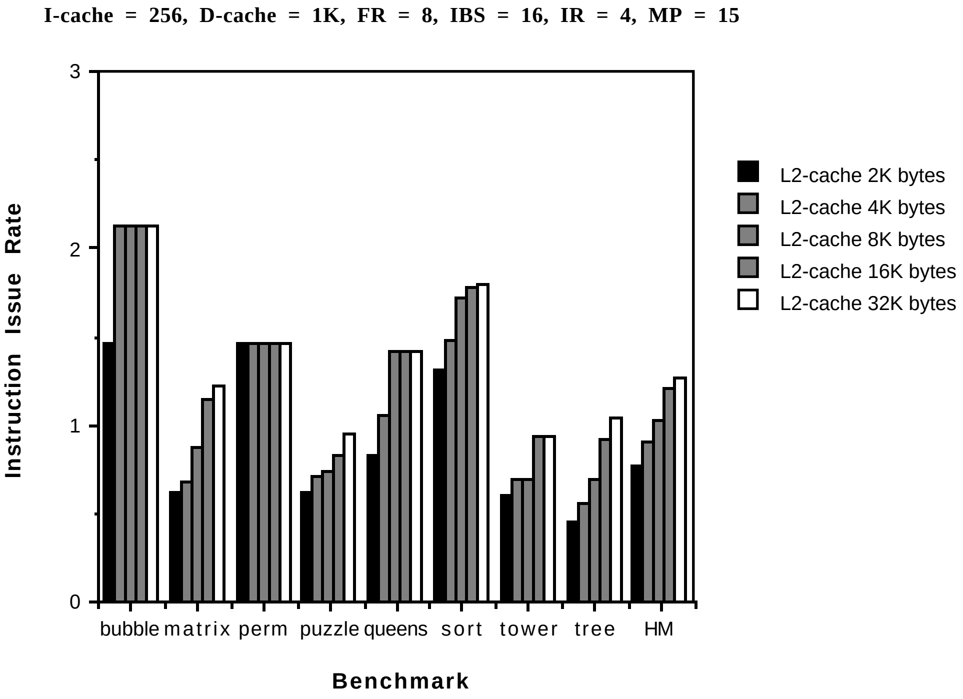
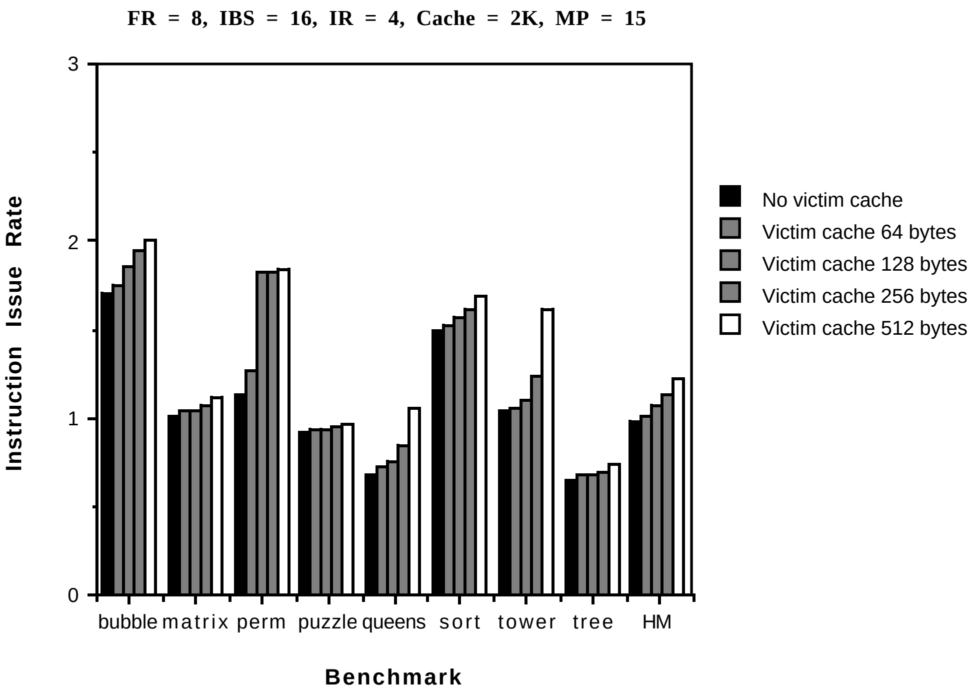


Fig. 6 Victim Cache: Issue Rate versus Victim Cache Size


FR = 8, IBS = 16, IR = 4, Cache = 2K, MP = 15
Instruction Issue Rate
bubble
matrix
perm
puzzle
queens
sort
tower
tree
HM
Benchmark
No victim cache
Victim cache 64 bytes
Victim cache 128 bytes
Victim cache 256 bytes
Victim cache 512 bytes