Memory Hierarchy Limitations in Multiple-Instruction-Issue Processor Design

Lucian Vintan
University of Sibiu, Romania
email: vintan@sibiu.ro

Gordon Steven
University of Hertfordshire, UK
email: comqgbs@herts.ac.uk

Abstract

Most research on multiple-instruction-issue processor architecture assumes a perfect memory hierarchy and concentrates on increasing the instruction issue rate of the processor. In contrast, this paper assumes an ideal processor model and seeks to quantify the limitations placed on superscalar processor performance by the memory hierarchy. The paper concludes that sustaining processor issue rates of four or more will probably ultimately require systematic pre-loading of cache blocks and the use of trace caches.

Keywords

Memory Hierarchy, Caches, Multiple-Instruction-Issue

1. Introduction

Current superscalar processor designs attempt to increase performance by issuing multiple instructions in each processor cycle. Multiple-instruction-issue can be achieved through aggressive out-of-order instruction issue, through static instruction scheduling at compile time or through a combination of the two.

Inevitably in order to limit the problem space much of the research on multiple-instruction-issue tends to assume a processor with a perfect memory hierarchy which delivers 100% cache hit rates. This paper reverses this situation by assuming a perfect processor model with an imperfect memory system. This allows us to explore the limits placed on the instruction issue rate by the cache miss rate, cache port limitations and other imperfections in the cache hierarchy. As well as the more usual separate data and instruction cache organisation, we also examine the impact of using a unified cache.

Our investigation uses the Hatfield Superscalar Architecture (HSA) [1, 2] as a convenient architectural vehicle. HSA was developed at the University of Hertfordshire to investigate multiple-instruction-issue in a high-performance superscalar architecture. To provide an idealised processor model, all instruction dependencies are ignored and all instructions are assigned unit latencies. Perfect run-time branch prediction is also assumed.

2. The HSA architectural model

HSA is a load and store architecture with a simple RISC instruction set. Distinctive features include guarded execution of all instructions and a generalised delayed branch mechanism [1].

During each processor cycle a group of instructions is fetched from the instruction cache into an Instruction Buffer. Multiple instructions are then issued in order from the Instruction Buffer to multiple functional units for parallel execution.

In this study an instruction issue rate of four is adopted throughout. Instruction issue is further limited by the availability of instructions in the Instruction Buffer and by the operation of the data cache which is only provided with two read/write ports. Significantly, while data dependencies between instructions that reference the data cache are respected, all other data dependencies between instructions are ignored. We are therefore simulating an idealised processor model with an imperfect memory hierarchy.

A cache access time of one cycle and a cache miss penalty of ten cycles 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. Trace driven simulation

This study uses trace driven simulation to investigate the impact of caches on processor performance. HSA code is first generated for our chosen benchmarks using a GCC compiler generated specifically for our architecture. The code is then executed on the HSA instruction-level simulator which in turn produces an instruction trace that includes every memory reference and every taken branch instruction that is executed. Finally, this trace is used by a trace driven simulation program developed at the University of Sibiu that simulates a range of cache models. The following major parameters provided by the trace driven simulator are of particular interest in this study:

FR (Fetch Rate): 4, 8 or 16 instructions per cycle

IBS (Instruction Buffer Size): 4 - 64 instructions

Cache Size: 64 - 16K words.

Issue Rate: Up to 4 words per cycle

All caches are direct mapped. In the case of the split cache models, the total cache size is divided equally between the data and instruction caches unless otherwise specified.

4. Simulation results

A variety of cache configurations were simulated. Initially, we set the Fetch Rate (FR) to 8, the Issue Rate (IR) to 4 and the Instruction Buffer Size (IBS) to 32. The total cache size was restricted to 64 entries (Fig.1). This resulted in separate instruction and data cache sizes of a mere 32 entries. In this case the inadequate cache size completely dominates performance and restricts the average issue rate to under 0.5 instructions per cycle. Under these circumstances separate instruction and data caches yield no overall performance benefit. In some benchmarks the higher hit rate provided by the unified cache resulted in the higher performance, while in others the independent access provided by two separate caches proved to be more effective (Fig.1). Overall there is little to chose between the two organisations with a harmonic mean instruction issue rate of $IR_u = 0.48$ instructions per cycle for a unified cache and $IR_s = 0.49$ instructions per cycle for separate caches.

In the next two configurations, the size of the cache was progressively increased to 512 (Fig.2) and then 2K locations (Fig.3). Each successive increase in cache size strongly boosts the issue rate, with the separate cache configuration outperforming the unified cache once an adequate cache size is provided. With 512 locations the mean instruction issue rates are 0.99 with a unified cache and 0.97 with a separate cache, rising to 1.2 for a unified cache and 1.52 with a separate cache with 2K locations.

Interestingly, the performance of the models with small caches (64 and 512 locations) can be boosted by increasing the Fetch Rate and Instruction Buffer Sizes (Fig.4 and Fig.5), partially compensating for the inadequate cache sizes. In Fig.4, the Fetch Rate is increased from 8 to 16 and the Instruction Buffer Size from 32 to 64. This in turn boosts the harmonic mean Instruction Issue Rate with the unified cache model from 0.48 to 0.52 instructions per cycle and with the separate cache model from 0.49 to 0.58 instructions per cycle. In Fig.5, the same Fetch Rate and Instruction Buffer Size increases are applied to the models with 512 word caches. With a unified cache, the issue rate increases from 0.99 to 1.10 instructions per cycle and with separate caches from 0.97 to 1.21 instructions per cycle. Nonetheless, Fig.6 and Fig.7 emphasise the overwhelming dominance of cache size in determining the effective issue rate.

Further performance improvements can be secured by further increasing the cache sizes. Fig.8 shows the benefits of varying the instruction cache size given a large 16K data cache. A harmonic mean issue rate of 2.36 is achieved. Fig.9 reverses the roles of the two caches, providing a large 16K instruction cache and varying the size of the data cache. Here the highest mean issue rate is 2.06. Depressingly, with a maximum issue rate of four, the memory hierarchy is limiting performance to slightly over two instructions per cycle, even though we assume a perfect CPU with no data dependencies, unit latency instructions and perfect branch prediction.

Fig.10 suggests one reason for this low issue rate. Here miss rates in the data cache are plotted for cache sizes of 64, 512 and 4K. As the data cache size is increased, miss rates decline from 50% to 24% and finally to 0.3%. However, while the final figure of 0.3% is encouraging, several programs still feature stubbornly high miss rates with the largest cache size.

5. Conclusions and discussion

In this paper we have assumed a perfect processor model and simulated the impact of the memory hierarchy on the instruction issue rate. Our results suggest that with an issue rate of four, the maximum sustained issue rate can be effectively halved by the memory system alone. This is a depressing result for researchers striving to realise high instruction issue rates. Furthermore increasing the issue rate beyond four can only exacerbate the problem.

We also show that although some individual programs sometimes yield superior performance with a unified cache model, separate data and instruction caches are generally desirable with an issue rate of four. This conclusion contrasts with the results from our earlier study [3] which suggested that, with instruction issue rates of only one or two, the use of a unified cache was a viable design option.

The performance loss observed comes from several sources besides the obvious one of cache misses. First, our model can only execute two memory reference instructions in each cycle. An average of 30% of the instructions in our benchmarks reference memory. Assuming a uniform random distribution of instructions, a simple probability calculation suggests that this limitation alone restricts the issue rate to an upper limit of 3.69 instructions per cycle.

A second factor is instruction fetch limitations caused by cache alignment problems. For example, with a fetch rate of four, if a branch targets the last word of a cache line, only one instruction will be fetched into the Instruction Buffer. Similarly, if the first word fetched turns out to be a taken branch, the effective fetch rate is again reduced to one. In this paper we have assumed a perfect processor model and simulated the impact of the memory hierarchy on the instruction issue rate. Our results suggest that with an issue rate of four, the maximum sustained issue rate can be effectively halved by the memory system alone. This is a depressing result for researchers striving to realise high instruction issue rates. Furthermore increasing the issue rate beyond four can only exacerbate the problem.

Several approaches can be taken to reduce the performance gap. Perhaps the most obvious is to introduce a second-level cache to reduce the cache miss penalty. However, in the longer term, systematic pre-fetching of data and instructions into primary level caches will probably be required if high instruction issue rates are to be sustained. In addition, the fetch rate is ultimately limited by the number of instructions between successive taken branches since this number also represents the number of instructions which can be fetched in parallel in a single cache access. Either multiple instruction cache ports will therefore be required or techniques to cache longer instruction traces [4] will have to be more fully developed.

Our future work in this area will first be directed at quantifying the components of memory hierarchy performance losses. We then hope to explore one of the lines of development briefly mentioned above.

References

1. Steven G B and Collins R, A Superscalar Architecture to Exploit Instruction-Level Parallelism, *Euromicro96*, 2-5 September, Prague, 1996.
2. 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, pp 391-400.
3. Steven G B and Vintan L Modelling Superscalar Pipelines with Finite State Machines, *Proceedings of Euromicro96*, IEEE Computer Society Press, 1997.
4. Rotenberg E, Bennett S and Smith J E Trace Cache: A Low Latency Approach to High Bandwidth Instruction Fetching, *Micro29*, Dec 2-4. 1996, Paris, pp 24-34.

Figure 1 Instruction issue rate with 64 word cache

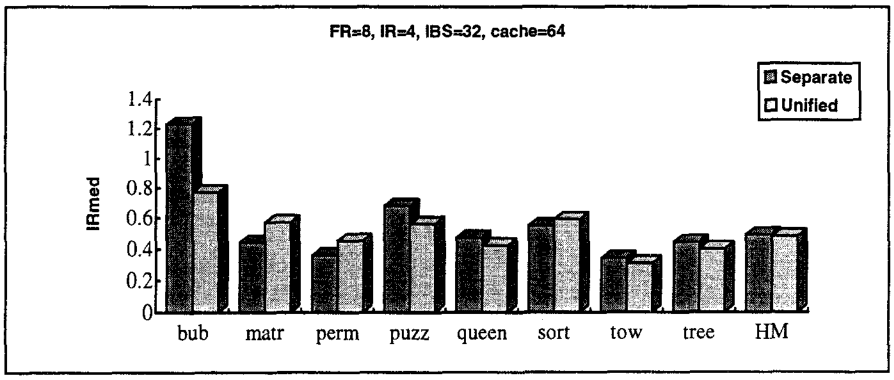


Figure 2 Instruction issue rate with 512 word cache

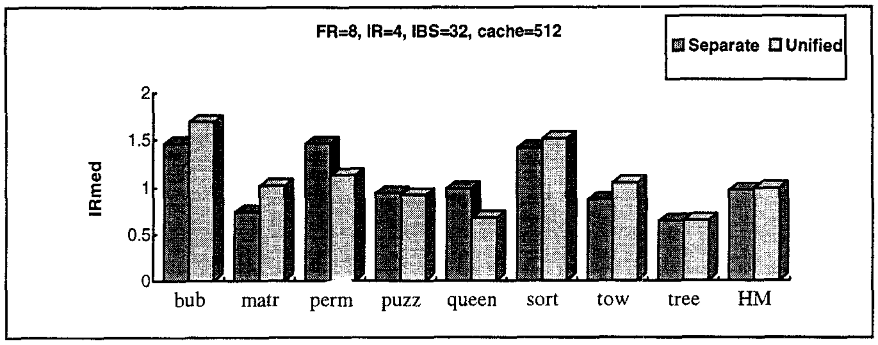


Figure 3 Instruction Issue Rate with 2K Word Cache

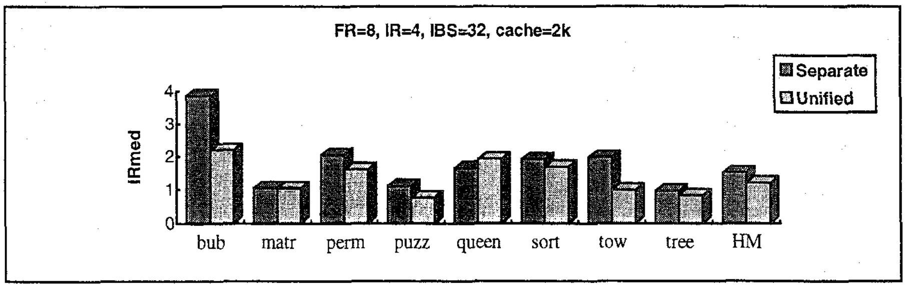


Figure 4 64 Word Cache with Increased Fetch Rate & Instruction Buffer Size

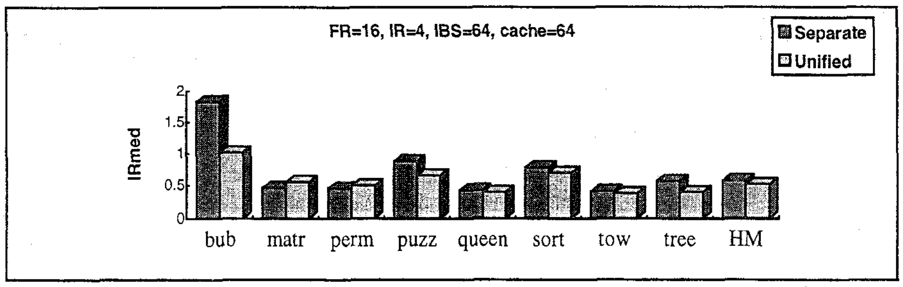


Figure 5 512 Word Cache with Increased Fetch Rate & Instruction Buffer Size

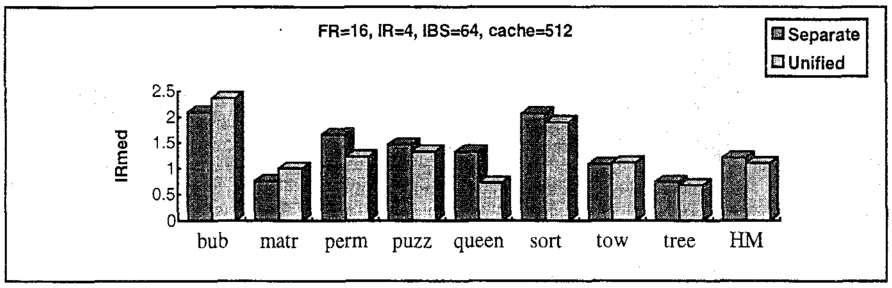


Figure 6 Issue Rate versus Cache Size (Separate Caches)

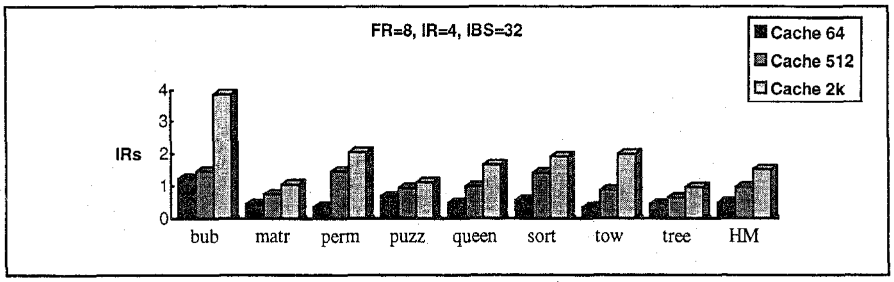


Figure 7 Issue Rate versus Cache Size (Unified Cache)

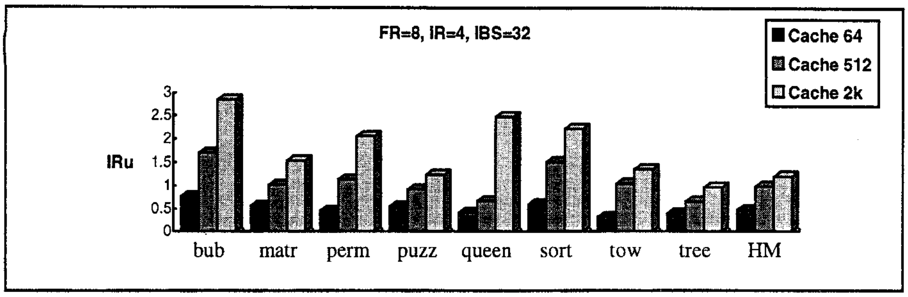


Figure 8 Impact of Instruction Cache Size

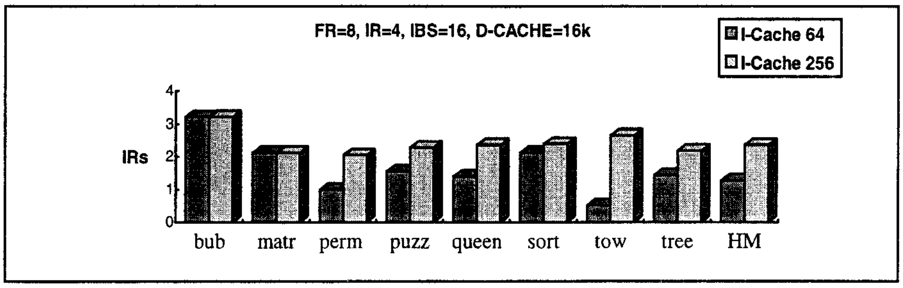


Figure 9 Impact of Data Cache Size

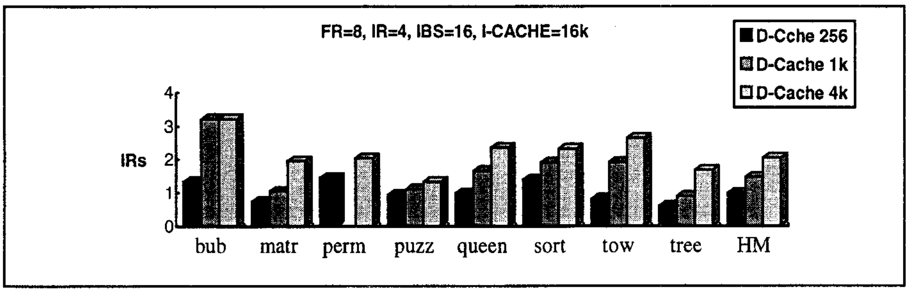
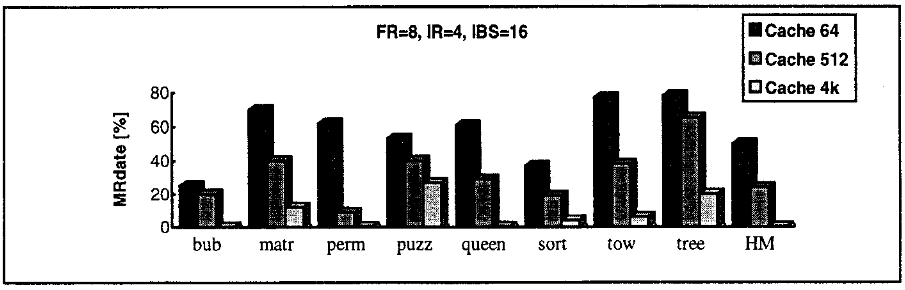


Figure 10 Hit Rate versus Cache Size


FR=8, IR=4, IBS=16
■ Cache 64
▨ Cache 512
□ Cache 4k
MRdate [%]
bub
matr
perm
puzz
queen
sort
tow
tree
HM