SIMULATING SOME ADVANCED PROCESSING TECHNIQUES INTO A SUPERSCALAR ARCHITECTURE

Adrian FLOREA, Lucian N. VINTAN,
University "L. Blaga", Department of Computer Science, Str. E. Cioran, No. 4, Sibiu-2400, ROMANIA,
E-mail: vintan@cs.sibiu.ro,

Abstract: The main aim of this short paper is to investigate multiple-instruction-issue in a high-performance superscalar architecture, illustrating the limits of some well-known technique (like dependence collapsing and instruction bypassing) or the best solution in that concern the cache architecture strategy. Also we propose a new technique, named Multiple – Load for improving processor performance in a superscalar architecture. Our research use trace driven simulation techniques to evaluate the processor performance.

Key words: Trace driven simulation, Superscalar, Cache, Write Back, Write Through, Instructions Collapsing

1. INTRODUCTION

The main simulation techniques used to evaluate and establish from a suite, the best processor pipeline configuration. The entering parameter for these techniques are benchmark programs (traces). We used the traces obtained based on the eight C Stanford integer benchmarks. These benchmarks were compiled through the HSA (Hatfield Superscalar Architecture) compiler, developed at the University of Hertfordshire, UK, by Dr. G.B. Steven's Research Group. Further, the traces were obtained using the HSA simulator, developed at the same university [Ste96]. Based on these tools, we have developed a trace driven simulator to investigate the limits of some well-known techniques (like dependence collapsing, instruction bypass using DWB – data write buffer) or the potential of some new developed techniques (like Multiple – Load) for improving processor's performance into a superscalar pipeline architecture (with four stages: **IF, ID, ALU/MEM, WB**). This technique called Multiple – Loads is based on an additional information (memory addresses of Load/Store instructions), available during the instruction decode stage.

2. SIMULATION WORK

2.1. BENCHMARK PROGRAMS

The simulation work has been centred on the Stanford integer benchmark suite, a collection of eight C programs designed by Professor John Hennessy, to be representative of non - numeric code while at the same time being compact. The benchmarks are computationally intensive with higher dynamic instruction counts (the cube packing problem, the eight queens problem, *bubble* sorts an array, *quick* sorts a randomised array, the *binary tree* sort, matrix multiplication, recursive computation of permutations, *Towers of Hanoi* - recursive problem). Although, many applications are not represented by the benchmarks, including graphics, multimedia, critical hand-coded operating system routines. All these benchmarks were compiled by the HSA Gnu C compiler, which targets the HSA instruction set. The resulted HSA object code was simulated by a dedicated HSA simulator [Ste96], which generates the corresponding traces.

The average instruction number is about 273.000. The average percentage of total instructions that are branches is about 13%, that are Load is still 18%, that are Store is about 12% and that are arithmetic is about 57% [Flo98].

2.2. THE SIMULATION METHOD

Following our aims, we developed a dedicated trace driven simulator [Vin99] that uses the above mentioned traces. The most important input parameters for this simulator are:

- FR – fetch rate – the instructions number that is fetched from Instruction Cache: up to 16 IPC (Instruction Per Cycle)
- IBS – instruction buffer size: up to 64 instructions
- Type and Size of Cache Memories (size of caches is measured in location; a location of Instruction Cache stores an instruction and a location from Data Cache stores memory addresses)
- BLOC_SIZE – the size of the data cache block: up to 32 location
- IRmax – parallel issue capability – the maximum number of instructions that can be dispatched concurrently from the Instruction Buffer: up to 8 IPC
- Instruction Latencies (measured in cycles): up to 20 cycles
- Type and Number of Functional Units

During this research we use a direct-mapped split (Instruction & Data) cache structure. Where not specified, the results were obtained using an optimal superscalar architecture [Flo98], with FR = 8, IBS = 16, IR_{max} = 4, BLOC_SIZE = 8, N_PEN = 10 cycles

(N_PEN - cycles required to load a block from main memory). We have also used a two-port data cache that can be accessed simultaneously by two non-aliasing Load/Store instructions.

The simulation results are presented in terms of instructions per cycle (IPC) and we summarised them by taking the harmonic mean over the benchmark set. During the simulation we have used as a main metric the average issue rate which encloses all simulated processes and generates a synthetic and realist performance indicator.

3. SOME RESULTS

A. Using **write back** strategy in Data Cache

There are two basic options when writing to the cache: **write back** and **write through**. Both of them have their own advantages. With write back, writes occur at the speed of the cache memory, and multiple write within a block require only one write to the lower level memory. With write through, read misses never resuit in writes to the lower level, and write through is easier to implement than write back. Write through also has the advantage that the next lower level has the most current copy of the data. Therefore, I/O and multiprocessors are fickle: they want write back for processor caches to reduce the memory traffic and write through to keep the cache consistent with lower levels of the memory hierarchy.

The simulation results show that write back strategy is with 44.69% more performant than write through. In our opinion, write back has a net advantage, eclipsed just by a harder implementation, especially related to cache coherence mechanisms into multiprocessor systems [Hen96]. Thus, following our aims, the simulations used the write back strategy.

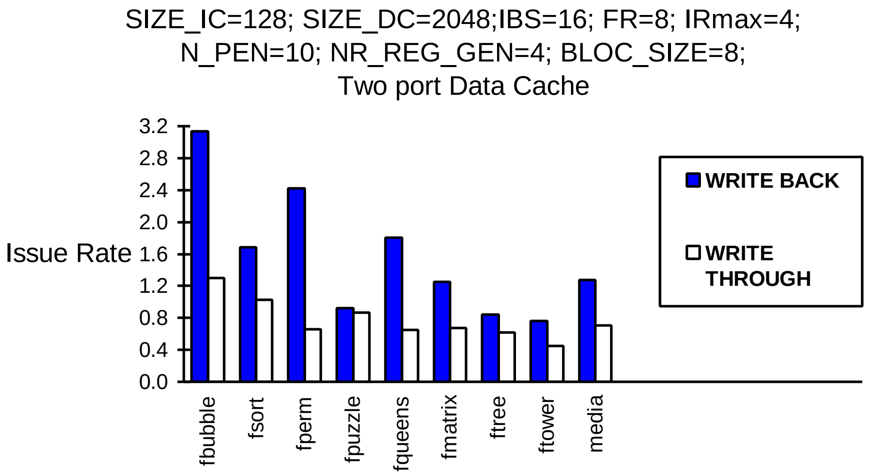

Figure 1 *Comparative study about processor performance depending on cache write strategy*

B. **Multiple Loads** – a new technique for improving processor performance

We are introducing now a new method for reducing the cache miss penalty, called "*multiple loads*". It is well known [Hen96] that reducing the cache miss penalty improves the processor's performance. Assuming that the target address of Load/Store instructions belonging to the instruction buffer are known, it could be issued multiple Load instructions, which are reading from the same fetched data cache block if there are a no intercalated Store instructions between these Loads. This method could be applied in an Out of Order mechanism with reservation stations for Load and Store instructions or even in a Trace Processor[Vin99b].

Using this technique, the processing rate could be involved by two factors: Size of Data Cache Memory (SIZE_DC parameter – see Figure 2) and Instruction Buffer Size (IBS parameter – see Figure 3). In a small Data Cache, sumarised by fewer data cache blocks, is likely to access the same block for several times; also, a bigger instruction buffer is likely to retain more Load instructions that access the same data cache block.

After simulation we got that the raising of SIZE_DC parameter improves processor's performance (Figure 2).

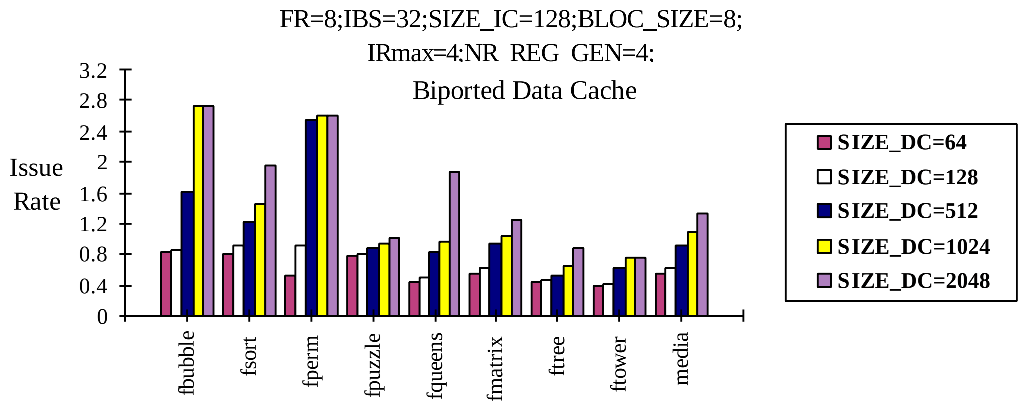

Figure 2 *Multi-Loads technique impact's about processor performance depending on Size_DC parameter varying*

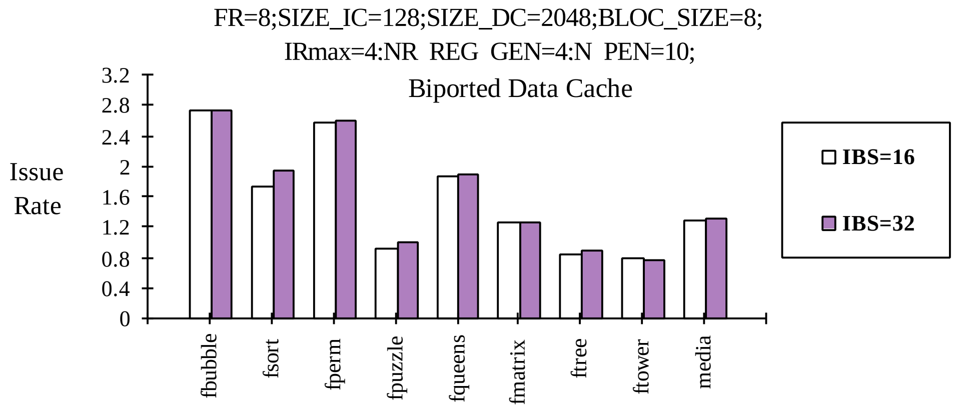

Figure 3 *Multi-Load technique impact's about processor performance depending on Instruction Buffer Size (IBS parameter) varying*

The raises of Instruction Buffer Size from 16 to 32 locations, combined with Multi - Loads technique involves a processor performance improvement with only about 3% percents (Figure 3).

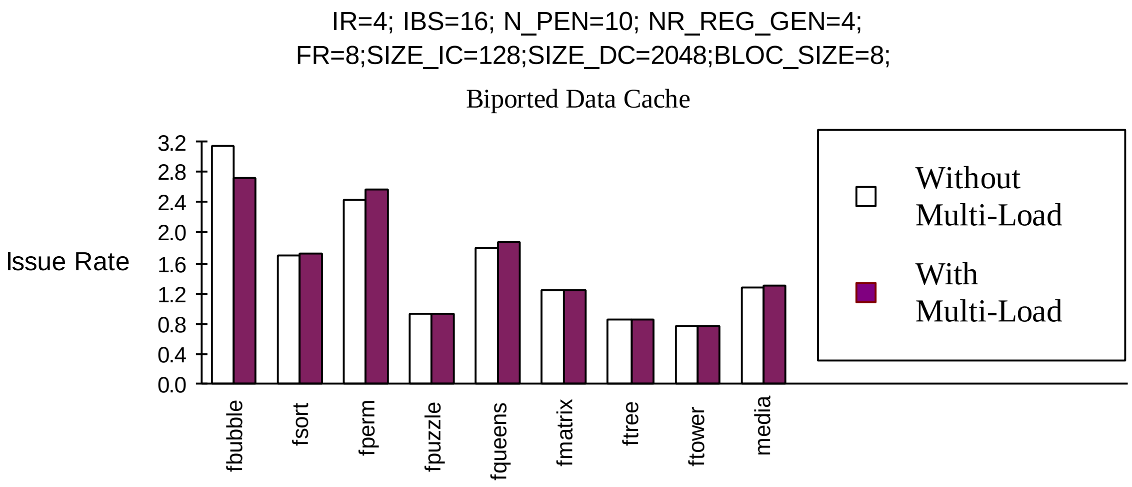

Figure 4 *Comparative study about processor performance depending on using or not of Multi-Load technique*

Surprisingly, this technique involves only a 2% percents growth about processing rate at average (very short). The anomaly appeared is due to *bubble*'s benchmark (see Figure 4). The explanation we got through NR_RAW parameter statistics – computed in program, parameter that represents the total number of Read After Write hazards. After trace driven simulation, on all benchmarks, except *bubble*, this parameter and implicit *the total number of execution cycles* is getting down using the Multi – Loads technique, but on *bubble* it raises approximately with 10000 hazards. Considering additionally at each RAW hazard at least one cycle penalty it gets for processing rate a diminution value than the case in which it doesn't use the Multi – Loads technique. Finally, this will have a negative effect about processing performance at average. It must notice that with this technique it doesn't gain much time because all Load instructions (sometimes except first) would execute with hit in Data Cache as they access same block, and no Store instruction doesn't write in this block, additionally appearing more RAW hazards. Although, the Multi – Loads advantage is the possibility to execute in a cycle more than IRmax instructions so that *the total number of execution cycles* will get down.

C. The limitation of instruction number that could be **collapsed**

Dependence collapsing represents a technique that resolves execution data Read After Write hazards for instructions requiring ALU operation. Dependence collapsing can reduce the latency eliminating data dependencies by combining dependencies among multiple instructions into one complex instruction. This technique improves the processor performance by "restructuring" the data dependence graph. A general scheme capable of collapsing involves arithmetic and logical instructions. Instructions that will be collapsed can be non-consecutive. The distance separating the collapsed instructions is nearly always less than 8. There are at least two possible implementation strategies [Vas93]: the first **run-time** strategy, based on a combining hardware mechanism of instructions from prefetch buffer, and second **static** strategy, a software combining realised by an instruction scheduler. Into a previous work [Vin99], we implement **run-time** this technique: in instruction buffer there are detected possibly data dependencies and if it is possible then the collapse is done under some certain rules (the dependence is generate by an arithmetic instruction). In the same paper [Vin99] we proved through trace driven simulation method that dependence collapsing and raising of instruction number that could be collapsed (**Comb_Instr** - parameter) from 2 to 3 improves the processor performance with about 22%. Although, this growth of Comb_Instr from 3 to 6 is significantly limited (under 3% – see Figure 6).

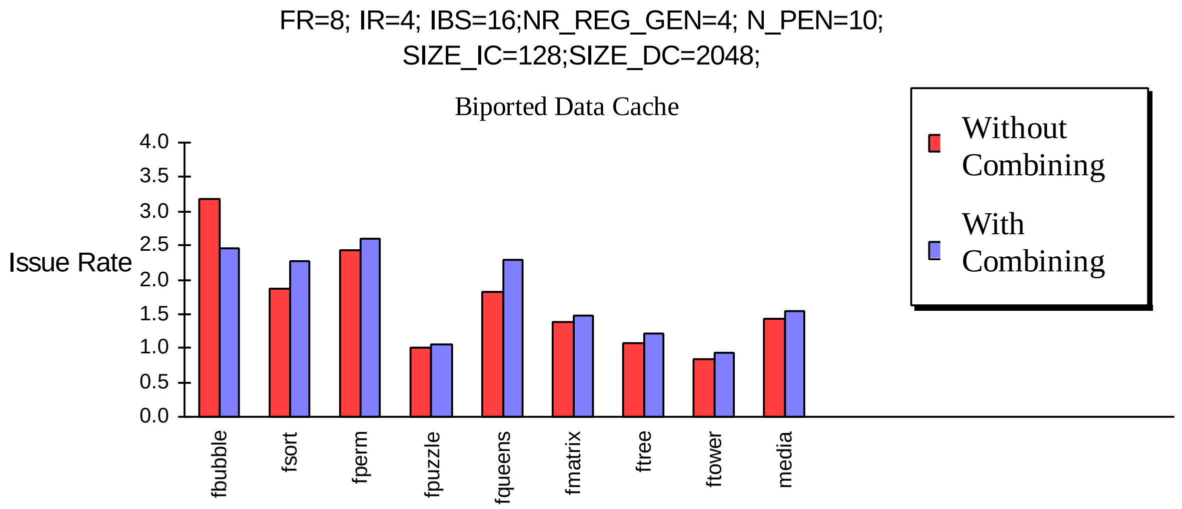

Figure 5 *Processing rate on a two-port data cache using or not dependence collapsing*

In the two-port data cache, dependence collapsing improves processor's performance with 8.51% at average. The maximum rise is 26.34% on *queens* benchmark (Figure 5).

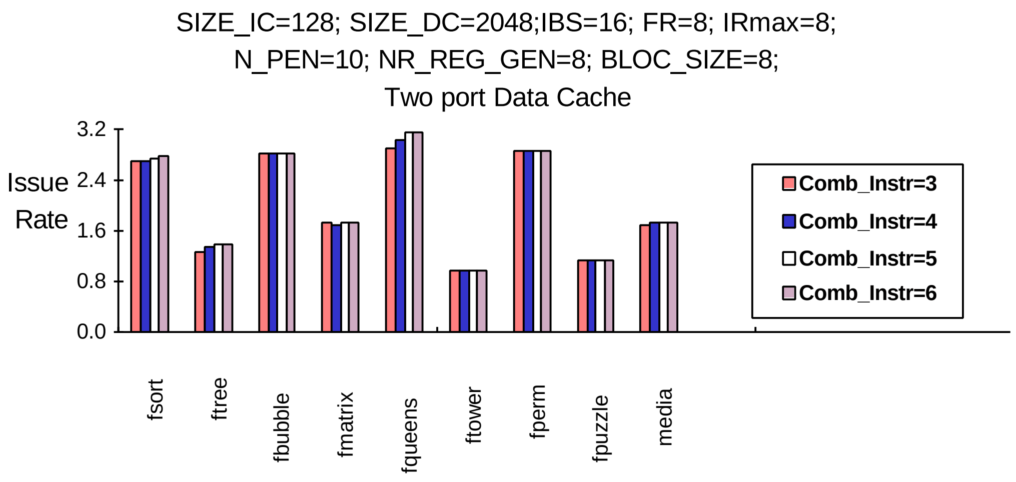

Figure 6 *Impact of the* ***Comb_Instr parameter*** *raising about processor performance*

D. Instruction bypass using DWB – data write buffer

DWB is a small write buffer, which contains the virtual address and data that must be written in Data Cache. Assuming that DWB has enough ports for supporting the worst situation (a lot of Store instructions, independent and concurrent stored in *instruction window*), offering then multiple virtual writing ports [Tat98] although, Data Cache has one or maximum two reading ports and just a single writing port. We set the writing latency in DWB to 1 cycle and the number of cycles needed for writing data from DWB to Data Cache to 2 or 3 cycles (variable). Using DWB we eliminate the need of serialising Store instructions with afferent penalties and besides, through bypassing we can eliminate many “Load after Store” hazards.

Variation from 2 to 3 cycles of DWB-latency implies a diminution with 11.63% of processor performance. That means the data writing process from DWB in Data Cache must be hurried for improving processor performance. Also, if the reading ports number raises from 1 to 2 the processing rate is improved with 5%. The simulation results show that bypassing technique at average favouring by DWB improves processing performance with 17.87% (Figure 7).

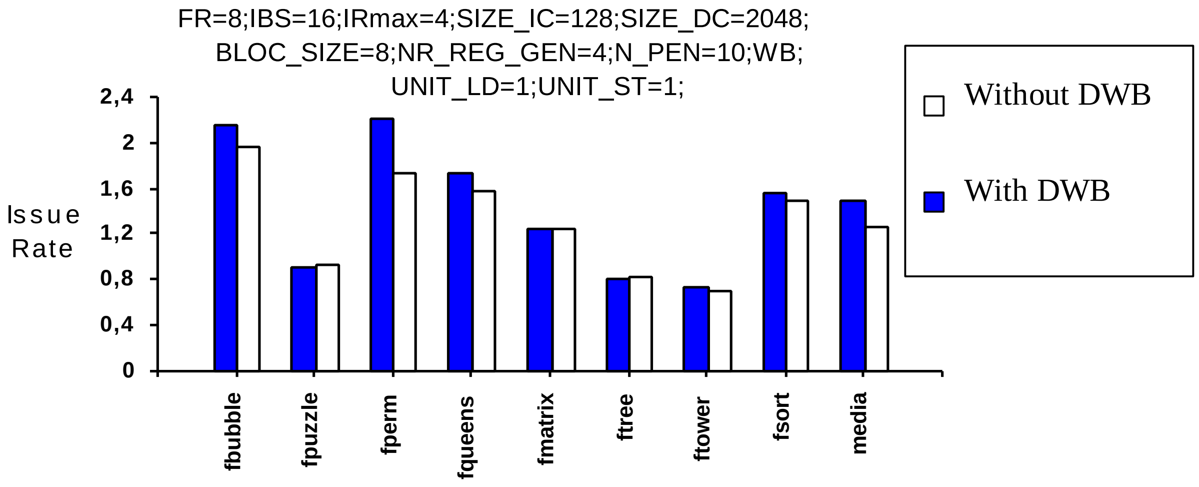

Figure 7 Processor performance improved with bypassing technique in DWB

4. CONCLUSIONS AND FURTHER WORK

The previous results point out consistently that a substantial performance growth is possible by using dependence collapsing. We show that the raising of instruction number that may be combined (Comb_Instr) from 2 to 3, has a more significant impact on processor's performance. Although, varying Comb_Instr parameter over 3, doesn't improve significantly the obtained average issue rate. Regarding to write policy in data cache we proved that write back is more adequate. The new method for reducing the cache miss penalty, "*multiple loads*" improves the processor's performance but this growth depends on execution pattern. Also, by hardware bypassing of instruction favouring by DWB the processing performance is increased at average with 17.87%.

In whole our simulations work we considered a perfect branch prediction (ideal). For further research we are concerned now to the necessity of an efficient hardware branch predictor. Very high prediction accuracy are necessary, because taking into account the multiple-instruction-issue processors characteristics as pipeline depth or issue rates, even a prediction miss rate of a few percent involves a substantial performance loss.

ACKNOWLEDGMENTS

This work was supported in part by the Romanian Ministry of Research and Technology grant MCT No. 4086/1998 and respectively by the Romanian National Council of Academic Research grants CNCSU No. 391/1998 and No. 489/1999. Also we like to thank Dr. Gordon Steven from University of Hertfordshire, UK, for providing the HSA Stanford traces.

REFERENCES

[1][Vin99] **Vintan L., Florea A., Steven G.** – *Advanced Techniques For Improving Processor Performance In A Superscalar Architecture,* CSCS-12 Conference, Bucharest, May 1999.

[2][Hen96] **Hennesy J., Patterson D.** – *Computer Architecture, A Quantitative Approach,* Morgan Kaufmann Publishers, Second Edition, 1996.

[3][Flo98] **Florea A.** – *Optimizarea proceselor de scriere într-o arhitectura RISC superscalara de tip Harvard*, Teza de Masterat, Sibiu, 1998 (co-ordinator L. Vintan).

[4][Ste96] **Steven G. B. et al.** – *A Superscalar Architecture to Exploit Instruction Level Parallelism*, Proceedings of the Euromicro Conference, 2-5 September, Prague, 1996.

[5][Vin99b] **Vintan L.** – *Architectura procesoarelor cu paralelism la nivelul instructiunilor.O abordare constructiva,* Editura Academiei Române, Bucuresti, 1999.

[6][Vas93] **Vassiliadis S., Phillips J., Blaner B.** – *Interlock Collapsing ALUs,* IEEE Transaction on Computers, Vol. 42, No. 7, 1993, pp. 825 – 839.

[7][Tat98] **Tate D., Steven G. -** *Adding a Cache Simulator to the Hatfield Superscalar Project,* University of Hertfordshire, Technical Report, 1998.