ADVANCED TECHNIQUES FOR IMPROVING PROCESSOR PERFORMANCE IN A SUPERSCALAR ARCHITECTURE

Gordon B. STEVEN,
University of Hertfordshire, Department of Computer Science, UK,
E-mail: *comqgbs@herts.ac.uk*

Lucian VINTAN, Adrian FLOREA,
University of Sibiu, Department of Computer Science, 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 optimum values for some processing parameters, as well as some advanced techniques for improving processor performance, such as dependence collapsing and instruction bypassing. Our analysis is based on a trace driven simulation method. 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.

Key words: *Superscalar, Trace Driven Simulation, Cache, Write Back, Write Through*

1. INTRODUCTION

Researchers traditionally use simulation techniques, to evaluate different processor pipeline configurations. First a parameterised simulator is written for the processor model; then a suite of benchmarks is executed to evaluate the performance of different configurations. While this well tried technique has generated many useful insights into processor performance, there is always some concern that results may have be biased by the characteristics of the benchmarks used.

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 potential of some advanced techniques (like dependence collapsing, instruction bypass using DWB – data write buffer) for improving processor performance in a superscalar architecture.

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. 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. Some characteristics of the used traces are given in Table 1.

Benchmark	Total instr.	% Branches	% (Load + Store)	% ALU	Description
Puzzle	804.620	23.11	17.01	59.87	Solves a cube packing problem
Bubble	206.035	15.08	29.55	55.36	Bubble sorts an array
Matrix	231.814	8.90	28.86	62.23	Matrix multiplication
Permute	355.643	12.35	40.56	47.07	Recursive computation of permutations
Queens	206.420	9.34	33.66	56.99	Solves the eight queens problem
Sort	72.101	11.35	23.65	64.98	Quick sorts a randomised array
Towers	251.149	11.45	38.38	50.15	Solves Towers of Hanoi problem (recursive)
Tree	136.040	17.76	26.62	55.61	Performs a binary tree sort

Table 1. Characteristics of the HSA 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 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
- IBS – instruction buffer size: up to 64 instructions
- 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
- 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)

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, 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.

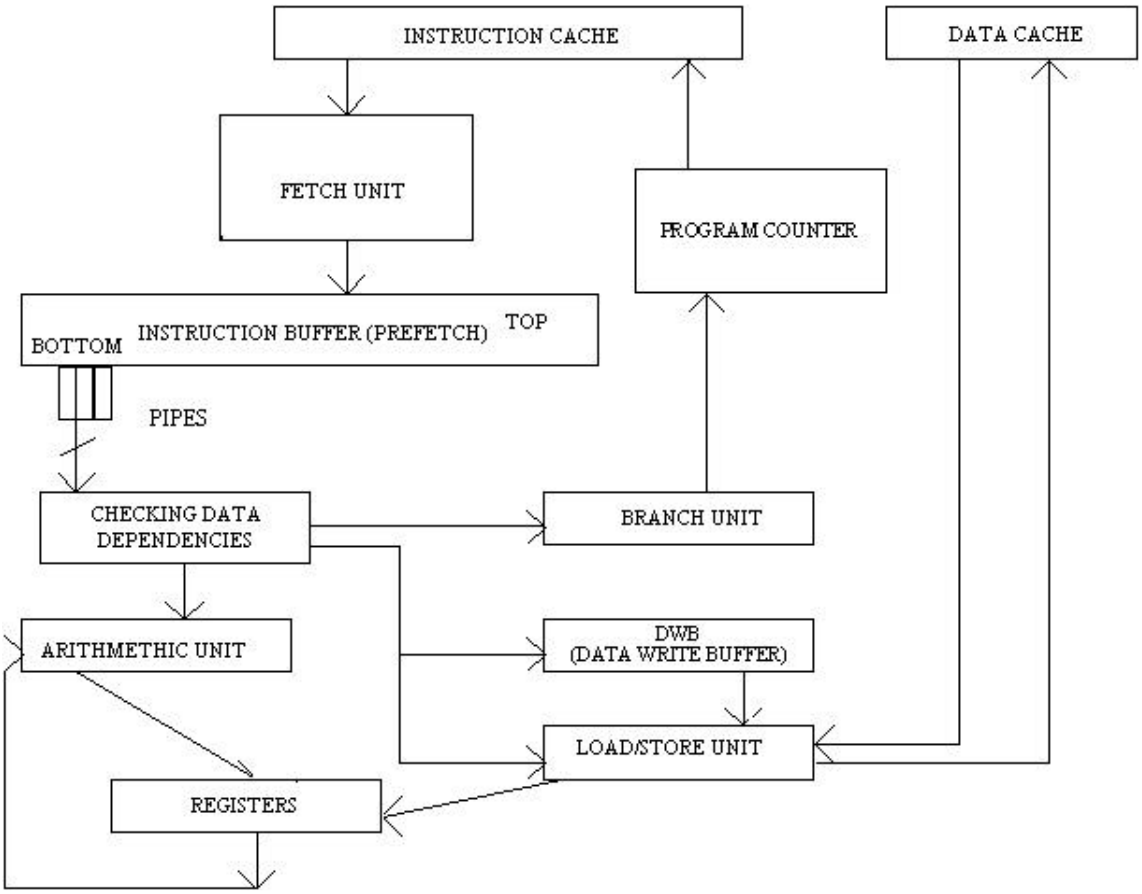

Fig. 1 The Simulated Architecture

3. SOME RESULTS

A. The potential of **dependence collapsing** for improving processor performance

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 dependences 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.

There are at least two possible implementation strategies: 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 instruction scheduler. In this work, we implement the first strategy: 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). Three dependent instructions or two groups of two dependent instructions belonging to the "*instructions window*" may be collapsed. The distance separating the collapsed instructions is nearly always less than 8.

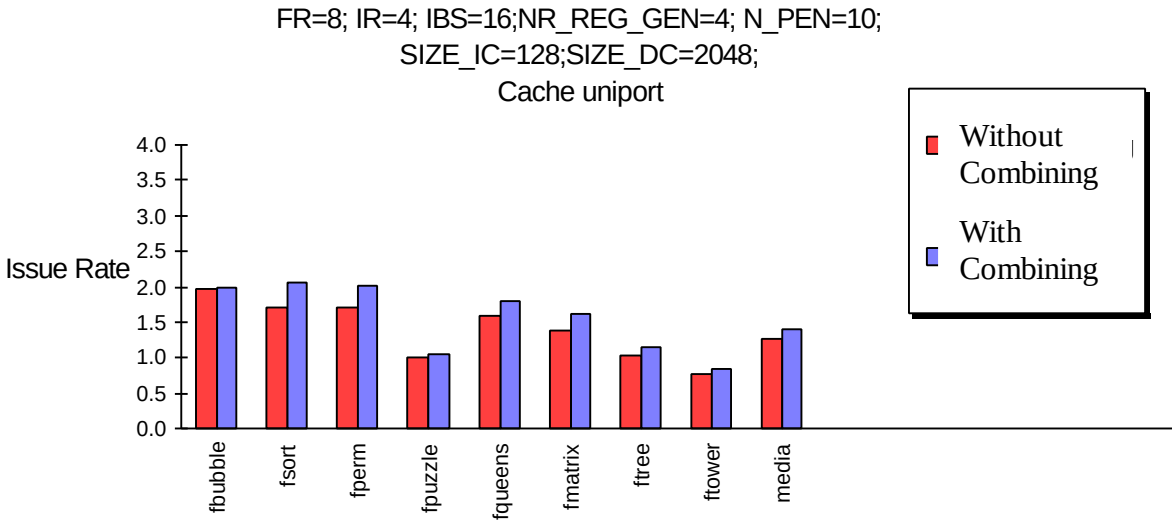

Fig. 2 Processing rate on a single-port data cache

In the single-port data cache case, the processing rate raises with 11.19% due to dependence collapsing. Also, in the two-port data cache case, this technique improves processor performance with 8.51% at average. The maximum rise is 26.34% on *queens* benchmark. (figure 3)

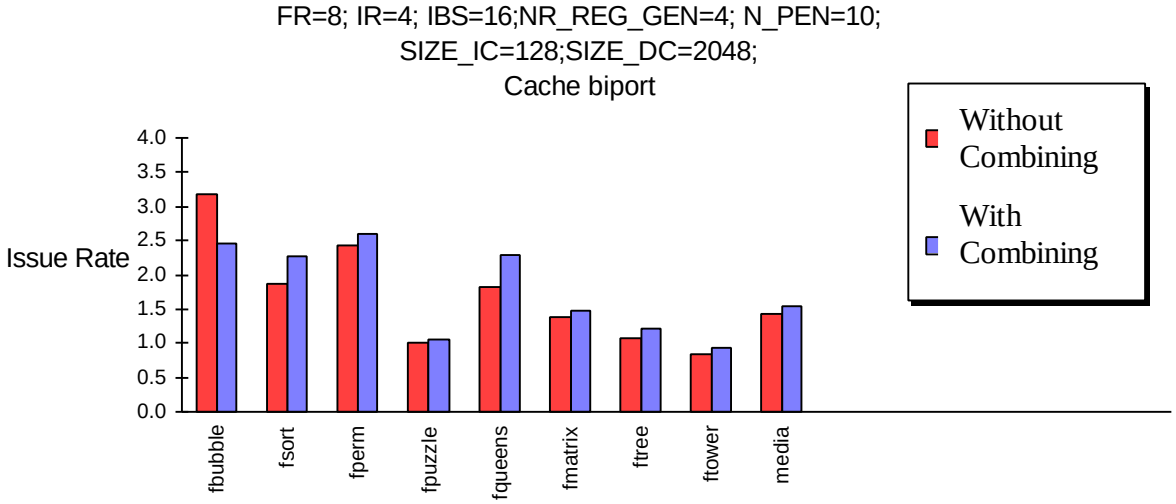

Fig. 3 Processing rate on a two-port data cache

Using dependence collapsing technique, on two-port data caches, the processing rates are with 9.2% greater than these get on single-port data caches, unlike 11.9% when we didn't use this technique.

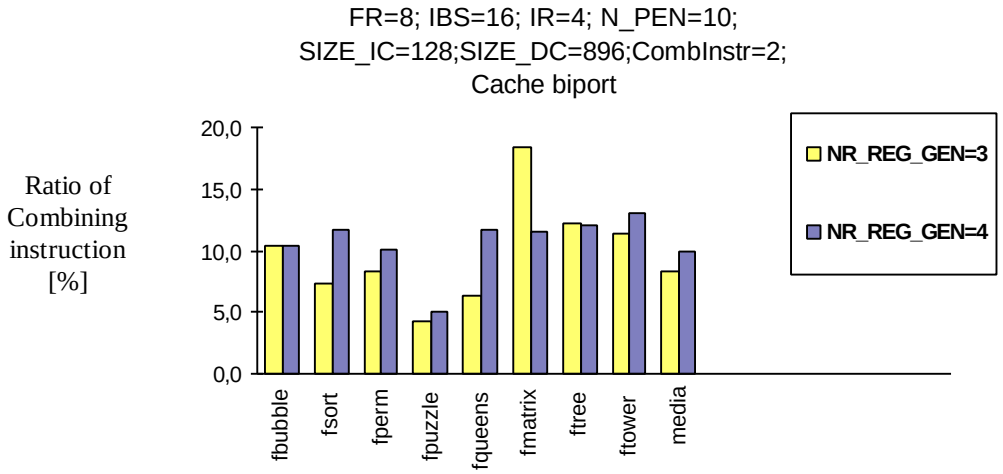

Fig. 4 Influence of number of register file on the percentage of combined instruction

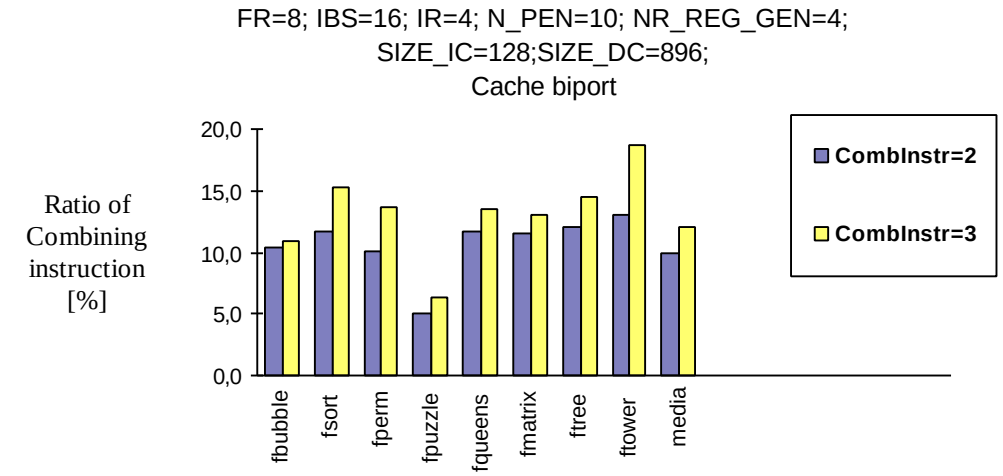

Fig. 5 Influence of parameter **CombInstr** on the percentage of combined instruction

CombInstr represents the number of combined instructions. If this value is 3 it means that three dependent instructions or two groups of two dependent instructions belonging to the "*instructions window*" may be collapsed. Variation from 2 to 3 of **CombInstr** parameter determines a rise with 22% of percentage of combined instruction (figure 5). Variation from 3 to 4 of number of register file implies a rise with 20% of percentage of combined instruction (figure 4). That means the resource limitation has a negative impact about techniques for improving processor performance.

B. 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 6).

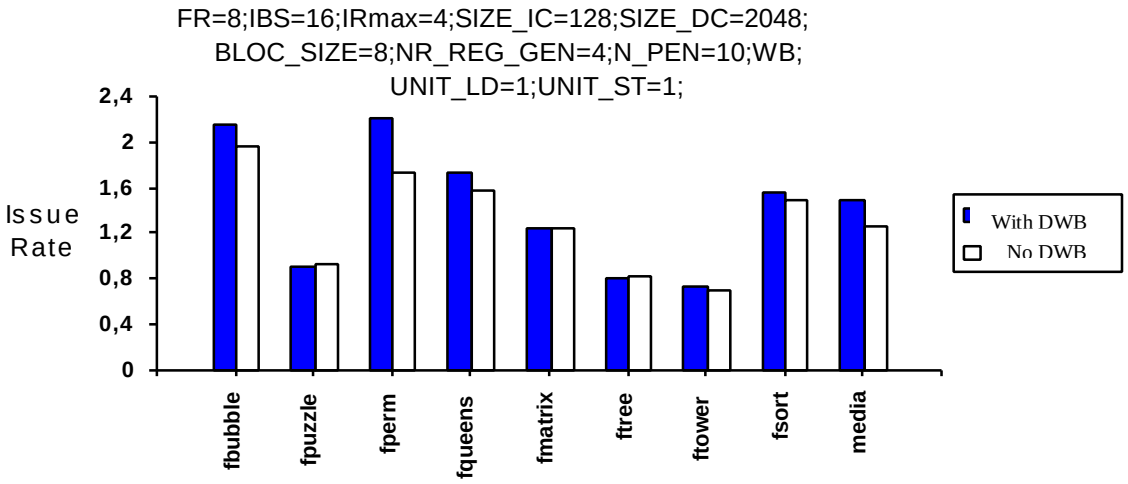

Fig. 6 Processor performance improved with bypassing technique in DWB

C. Optimising **BLOC_SIZE** (the size of the data cache block)

A way for improving the processor performance is the cache optimisation by reducing miss rate. A simple way to reduce miss rate is to increase the block size. Larger block sizes will reduce compulsory misses. At the same time, larger blocks increase the miss penalty. Since they reduce the number of blocks in the cache, larger blocks may increase conflict misses and even capacity misses if the cache is small [Hen96]. To choose an optimal value for data block size it requires an optimal trade-off between the increased miss penalty and the decreased miss rate. After simulation we got that the variation of BLOC_SIZE parameter: from 4 to 8 implies an improving of processing rate with 8.91%; from 8 to 16 with 5.61% and from 16 to 32 with just 1.91%. Thus, we recommend that optimum value for BLOC_SIZE to be 8.

D. Optimising IRmax parameter (the parallel issue capability)

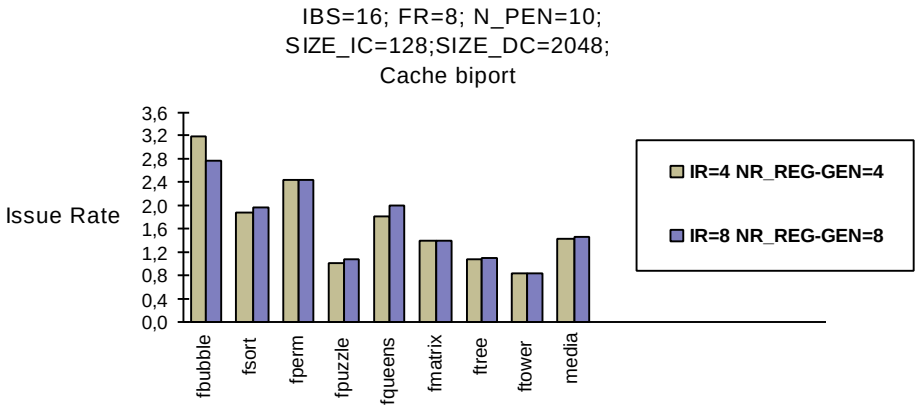

Fig. 7 Influence of IRmax parameter about processing rate

High-performance superscalar processors attempt to increase performance by issuing multiple instructions in each processor cycle. Although, our simulation results point out that the processor configuration with IRmax=8 (noted **C1**) outperforms another configuration having IRmax=4 (noted **C2**) with only about 2% (figure 7). The anomaly appeared on *bubble* benchmark is because by growing up the parallel issue capability during the trace execution it is obtaining a dynamic scheduling of concurrent running instruction, different from a case C1 to C2, resulting a glide *instruction window*. Thus, it is possible that on k-th issue step, the two gliding instruction windows (first – C1, second – C2) will be therefore completely disjunctive. Besides, in C1 case, by this dynamic scheduling, will add some data dependencies (data cache conflicts) that before (case C2), didn't exist. Anyway, the anomaly disappears in the case of a single port data cache. Finally, we conclude that if the processors don't run scheduled benchmarks, it has no sense to choose a parallel issue capability greater than 4.

4. CONCLUSIONS AND FURTHER WORK

The previous results show consistently that a substantial performance growth is possible by using dependence collapsing. We show that the raising of instruction number that may be combined has a more significant impact on processor's performance. Also, by hardware bypassing of instruction favouring by DWB the processing performance is increased at average with 17.87%. Regarding to write policy in data cache we proved that write back is more adequate. For further research we are working now to a method for reducing the cache miss penalty, called "*multiple load*". Assuming that the target address of Load/Store instructions belonging to the instruction buffer are known, it could be issued multiple Loads instructions, which are reading from same block from data cache and there are a no Store instruction, intercalate between two Loads.

ACKNOWLEDGEMENTS

This research has been partially supported by the National Romanian Agency for Research and Technology grant 4086/1998 and also through a sponsorisation obtained from "S.C. PIM S.A. SIBIU".

REFERENCES

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

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

[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][Smi96] **Smith J., Vassiliadis S.** - *The Performance Potential Of Data Dependence Speculation & Collapsing*, Paris, Proc. of the 29th IEEE Int'l Symp. on Microarchitecture, December 1996.

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

[6][Hill88] **Hill M**. – *A Case for Direct-Mapped Caches*, IEEE Computer, December, 1988.