DIVISION OF COMPUTER SCIENCE

MODELLING SUPERSCALAR PIPELINES WITH FINITE STATE MACHINES

Gordon Steven
Lucian Vintan

Technical Report No. 246

September 1993

Modelling Superscalar Pipelines with Finite State Machines

Gordon Steven* comqgbs@herts.ac.uk and Lucian Vintan** vintan@sibiu.ro

* Division of Computer Science, University of Hertfordshire, College Lane, Herts, AL10 9AB, UK
** Department of Computer Engineering, University of Sibiu, Revolutiei No 4, Sibiu-2400, Romania

We are currently examining the use of Finite State Machines to model processor pipelines. Our aim is to discover the extent to which the complexities of modern superscalar pipelines are amenable to theoretical modelling. As a result, we hope to improve our understanding of pipeline design. In this short paper we illustrate our approach by analysing the trade-offs between separate and unified caches in a superscalar system.

Key Words Superscalar ; Pipelines; Unified Cache; Finite State Machines

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.

As an alternative, this paper investigates the use of Finite State Machines (FSM) to model pipeline performance. Our objective is to gain a theoretical insight into the operation of processor pipelines and to relate pipeline performance to quantifiable characteristics of individual benchmarks.

As an example of our approach, we develop several pipeline models which we use to evaluate the impact of replacing the more usual separate data and instruction caches with a single unified cache. Our analysis suggests that, while separate caches do in general improve processor performance, the advantage is significantly less than might be expected.

2. Basic Pipeline Model

Throughout the paper we use the well-known DLX pipeline model [Hen96]. DLX has the following five stages:

IF	Instructions Fetched from the Instruction Cache.
ID	Instructions Decoded & Operands fetched from the Register File.
EX	Inter-register instructions Executed.
	Memory addresses computed.
MEM	Data Cache accessed.
WB	Results returned to Register File.

If we assume an ideal pipeline with unit instruction latencies, 100% cache hit rates and perfect branch prediction, then DLX will execute one instruction per cycle. If, however, the separate instruction and data caches are replaced by a single unified cache, the pipeline will stall for one cycle every time a load or store instruction is executed. A benchmark will therefore require "1 + P(mem)" cycles to execute each instruction, where P(mem) is the probability of an instruction being a load or store. The Instruction Issue Rate (Instructions Per Cycle) is therefore given by:

$$\text{Issue Rate (IR)} = 1 / (1+ P(\text{mem})) \quad (1)$$

With P(mem) typically around 0.40, the introduction of a unified cache will therefore cause the IR to fall from 1 to around 0.71. The case for using separate caches therefore appears to be very strong.

However, several considerations make a unified cache more attractive than the above simple analysis suggests. Firstly, providing only one cache simplifies the processor hardware, since only one set of cache control and address decode logic is required. Secondly, the data cache stall may be masked by stalls elsewhere in the pipeline. For example, in the DLX pipeline, an instruction immediately following a load instruction must stall

for one cycle if it wishes to use the data being loaded. This stall effectively negates any benefits from a separate data cache. Similarly a long latency instruction, such as a multiply or divide which remains in the EX stage for several cycles can mask the effect of an instruction fetch stall. Finally, a unified cache will result in more efficient cache utilisation and a higher hit rate. Patterson and Hennessy, for example, report DEC 5000 simulations where, depending on the cache sizes, separate caches have miss rates 5-14% higher than a unified cache [Hen96]. In all cases, to keep the total number of cache entries constant, the separate caches are assumed to be half the size of the unified cache.

As a result, in a commercial processor, the advantages of separate caches are far less obvious. For example, the Power PC-601 [Weis 94] combines a unified cache with an instruction fetch rate of eight instructions per cycle and a maximum instruction issue rate of three. Furthermore an instruction buffer with capacity of eight instructions can continue to supply instruction to the pipeline when a read or write instruction stalls the instruction cache.

3. Superscalar Pipeline Models

We therefore developed more complex models to explore alternative cache configurations in a superscalar environment. The basic five-stage DLX model was retained with an Instruction Buffer (IB) inserted after the IF stage (Fig1). Both the instruction fetch and instruction issue rates were also no longer restricted to one. However, we continued to assume unit instruction latencies and perfect branch prediction. One of our main interests was to determine how frequently the Instruction Buffer could continue to supply instructions to the pipeline during data cache stalls.

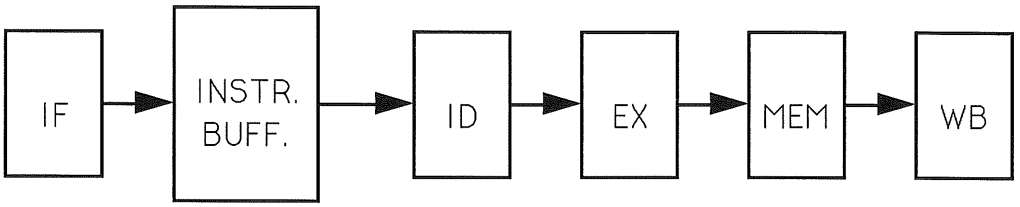

Figure 1. The Pipeline Model

Modelling was initially performed using the concept of collision vectors [Sto93]. However, this approach proved to be unnecessarily complex. Instead, the pipeline was modelled directly as a FSM (Finite State Machine). Conceptually, each state in our models consists of three orthogonal components, S(IB), S(ID) and S(EX), which taken together represent the state of the pipeline. The first component, S(IB), represents the number of instructions currently held in the Instruction Buffer. The second component, S(ID), is a binary value which records whether an instruction in the ID stage is a load or a store. S(ID) therefore indicates whether the instruction in the ID stage will result in an instruction fetch stall after two cycles. Similarly, S(EX) records whether there is a load or store instruction in the EX stage. A S(EX) value of one indicates that there will be an instruction fetch stall in the next cycle.

As an example, consider a model with a FR (Fetch Rate) of two, an IR (Issue Rate) of two and an IBS (Instruction Buffer Size) of four. This FSM therefore requires 20 distinct states. Now consider the probability of each state transition. Surprisingly, the transition probabilities can be computed as a function of only two parameters, P(2instr), the probability that two instructions can be issued from the Instruction Buffer and P(mem), the probability that an instruction is a load or store. (We assume that at least one instruction can always be issued from a non-empty buffer, so P(1instr) = 1 - P(2instr)). Using these two variables, we can assign a probability to each state transition.

Current State	Transition	Next State	Transition Probability
3,1,0	Fetch 2, Issue 1	4,0,1	P(1instr) (1-P(mem))
		4,1,1	P(1instr) P(mem)
	Fetch 2, Issue 2	3,0,1	$P(2instr)\ (1-P(mem))^2$
		3,1,1	$P(2instr)\ (1-(1-P(mem))^2)$

Figure 2. FSM Example

Figure 2 shows the state transitions from one state in our FSM example. Initially there are three instructions in the Instruction Buffer and a memory reference instruction occupies the ID stage. Four transitions are possible. Since at least one instruction will always be issued, two more instructions will always be fetched into the Instruction Buffer. Also the next value of S(EX) will always be one. However, one or two instructions may be issued depending on P(2instr), and S(ID) will be zero or one depending on whether or not a load instruction is issued.

From our state transition table we can obtain an equilibrium matrix equation relating the states of the FSM:

$$| \mathbf{p} | = | \mathbf{p} | * | \mathbf{T} | \quad (2), \text{ where:}$$

$$| \mathbf{p} | = | p_1 \; p_2 \cdots p_i \cdots \; p_N |$$

$$| \mathbf{T} | = | t_{i,j} | \text{ where } i=1, 2,..., N \text{ and } j=1, 2,..., N$$

We have used the following notation:

p_i = the probability of being in state i of the FSM
N = the total number of FSM states; N = (IBS + 1)*4
$t_{i,j}$ = the probability of transition from state i to state j

Since, at any given time, our FSM must always be in one of the states, we also have the following normalisation equation:

$$\Sigma_i \; p_i = 1 \qquad (3)$$

The above system of N homogeneous equations can be solved using well-known numerical techniques [Nob88]. The equilibrium probabilities obtained for each state can then be used to calculate the average instruction issue rate:

$$IR_{uc} = f \text{ (P(2instr), P(mem), } P_1, \ldots .P_N) \qquad (4)$$

In the case of separate instruction caches, the performance will be entirely determined by the processor's ability to issue two rather than one instruction in each cycle.

$$IR_{sc} = 1 + \text{P(2instr)} \qquad (5)$$

Using these techniques a wide range of models can be developed for a variety of pipelines.

4. Results

The DLX model (Fig.1) was investigated for a range of Instruction Fetch(IF) rates, Issues Rates (IR) and Instruction Buffer Sizes (IBS). The results for IF = 2, IR = 2, and IBS = 4 are shown in Fig 3 for different values of P(mem). Note that using a separate data and instruction cache will given the same results as setting P(mem) to zero. In this model the impact of a unified cache is minimal if only one instruction is issued per cycle but increases steadily as P(2instr) and P(mem) are increased. For example, with P(2instr) = 0.5 and P(mem) = 0.4, performance is degraded from 1.5 instructions per cycle for a dual cache model to 1.14 instructions per cycle for an unified cache model.

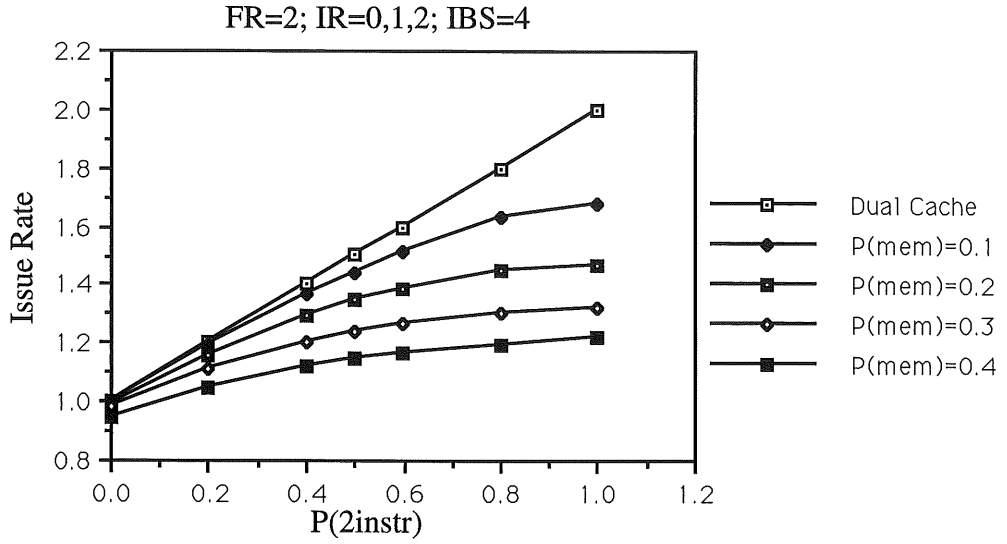

Figure 3

This performance gap can be reduced by either increasing the Instruction Buffer Size (IBS) or by increasing the Instruction Fetch rate. For example, in Fig.4, the IBS is increased to 16 while IR remains at two. Now with P(2instr) = 0.5 and P(mem) at 0.4, performance is 1.48 instructions per cycle, only 1.2% less than the equivalent dual cache model.

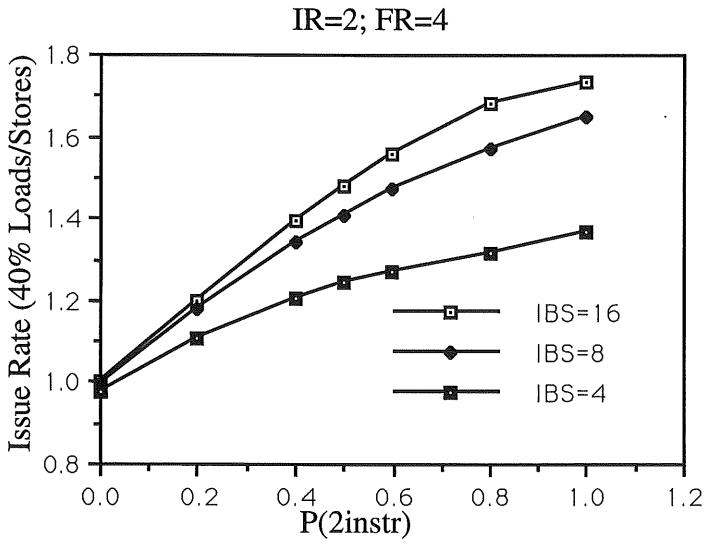

Figure 4

In Fig.5, the Instruction Fetch rate is also increased. Now with the same parameters, performance is only degraded by 0.38%.

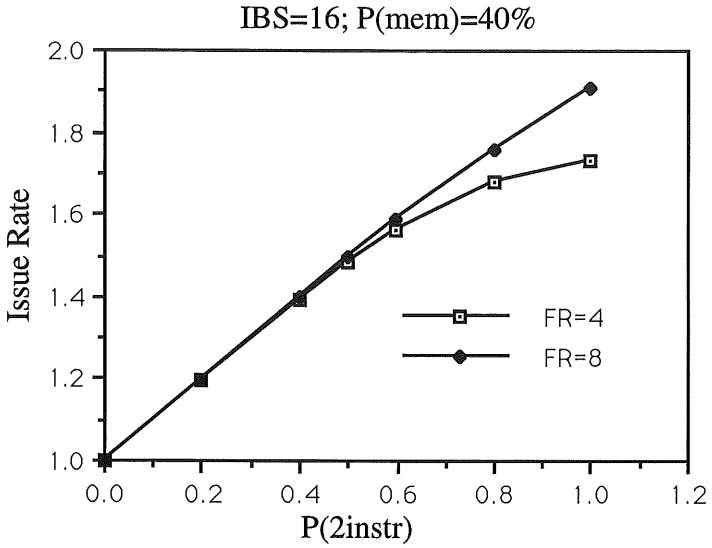

Figure 5

Figure 6 confirms that with an IBS of 16 and with a fetch rate of four or eight, the Instruction Buffer is rarely empty as long as P(2instr) is less than or equal to a half. These figures are consistent with the low degradation of performance observed earlier.

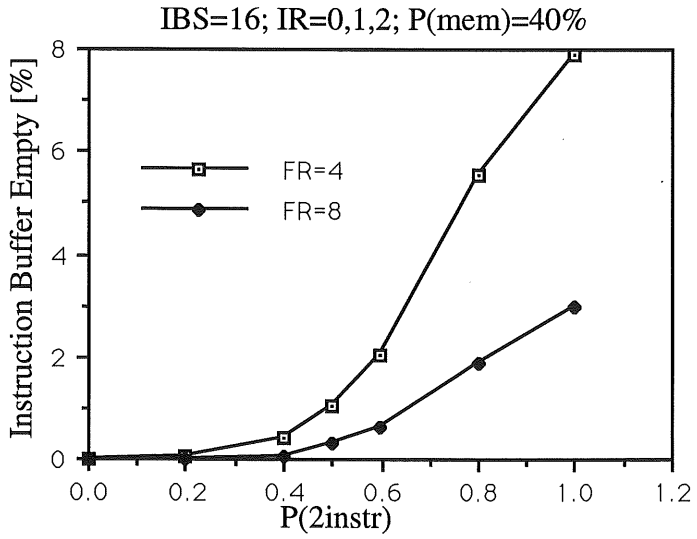

Figure 6

Finally, Figures 7, 8 and 9 present quantitative evaluations of the Issue Rates as a function of P(2instr) for different processor models.

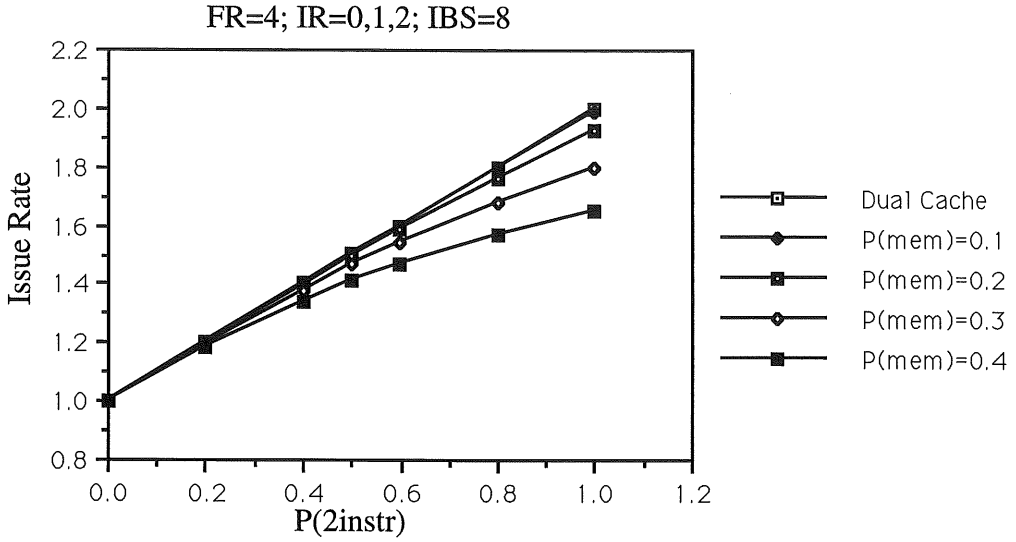

Figure 7

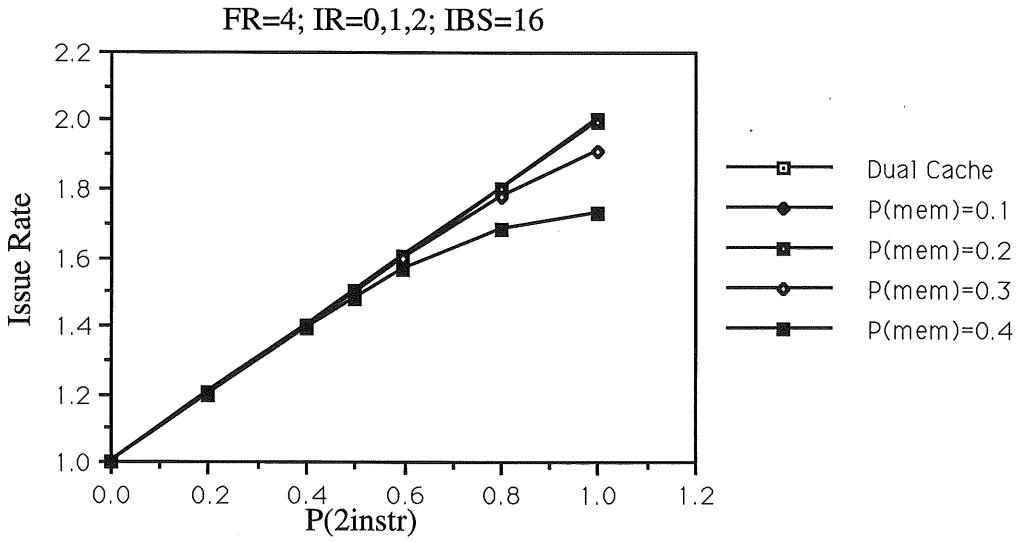

Figure 8

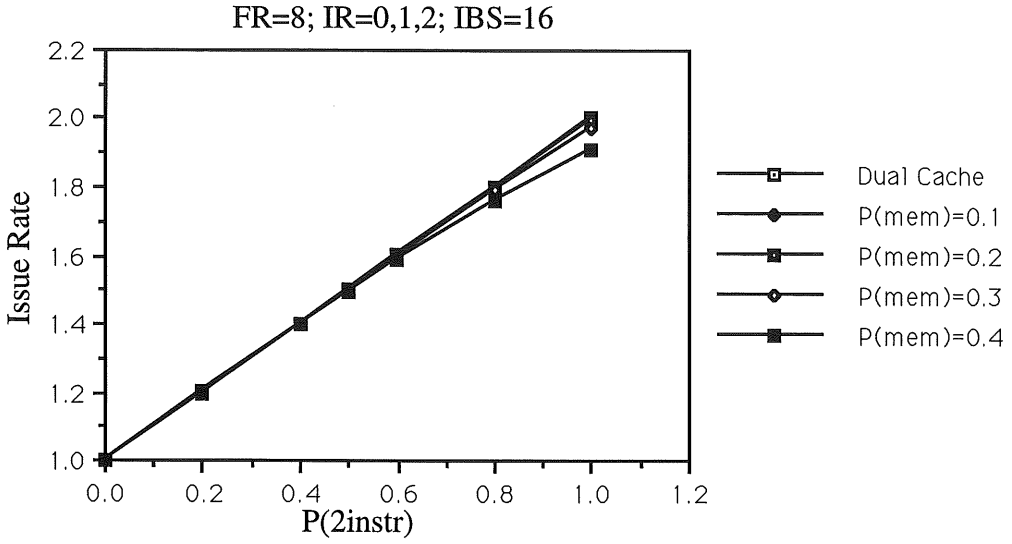

Figure 9

5. Conclusions and Discussion

We conclude that FSM modelling is a useful alternative to conventional processor simulation techniques. In particular, we have shown that the potential performance loss suffered through the use of a unified instruction and data cache can be avoided by prefetching instructions into an Instruction Buffer.

In future we plan to extend our approach to more complex pipeline models and to remove several simplifying assumptions from our existing models. For example, at present we assume that a fixed number of instructions can always be successfully pre-fetched from the instruction cache into the Instruction Buffer. Even with perfect branch prediction and an ideal cache this is incorrect, since in practice any instructions following a taken branch must be discarded from the Instruction Buffer.

Our overall goal is to explore the limitations of FSM modelling and to discover at what level of complexity our approach becomes unrealistic. Certainly the number of states cannot be increased indefinitely, since the computational complexity of solving N homogeneous linear equations increases in proportion to the cube of the number of states. Finally, we would also like to correlate the performance of our FSM models with the performance of benchmarks on more traditional simulation models.

Acknowledgements

The authors would like to thank Dr Mike Bartholomew-Biggs from the Numerical Optimisation Centre (NOC) at the University of Hertfordshire for providing a FORTRAN program to solve homogeneous linear equations systems and for patiently explaining how to use it effectively.

References

[Hen96] Hennessy J. and Patterson D. *Computer Architecture A Quantitative Approach*, Morgan Kaufmann 1996.

[Nob88] Noble B.- *Applied Linear Algebra,* 3 rd edition, 1988.

[Sto93] Stone H.- *High Performance Computer Architecture*, Addison-Wesley, 1993.

[Weis94] Weiss S. and Smith J.E. - *POWER and Power PC*, Morgan Kaufmann, 1994.