A Comparison of Data Prefetching on an Access Decoupled and Superscalar Machine*

G. P. Jones

Dept. of Computer Science
Edinburgh University
Edinburgh, Scotland, U.K

N. P. Topham

Dept. of Computer Science
Edinburgh University
Edinburgh, Scotland, UK

Abstract

In this paper we investigate the behavior of data prefetching on an access decoupled machine and a superscalar machine. We assess if there are benefits to using the decoupling paradigm given that an out-of-order (o-o-o) superscalar architecture could in principle prefetch to the same degree as an access decoupled machine.

We have found that for large issue width *the access decoupled machine can hide memory latency more effectively than a single instruction window o-o-o superscalar architecture. Our findings also demonstrate that an access decoupled machine offers the benefit of reducing the complexity of window issue logic.*

1 Introduction

The future of high performance microprocessor design is to provide improved performance by extracting higher degrees of instruction level parallelism. In superscalar architectures parallelism is exploited by reordering instructions within an instruction window and issuing multiple independent instructions per cycle. However as processor speeds increase and issue widths get larger the cost of a main memory access is becoming relatively more expensive. One solution is to hide memory latency by *data prefetching*.

Data prefetching is a technique that hides memory latency by overlapping access and data operations. Data prefetching can be implemented in either hardware [6] and software [3] or a hybrid [4] of both schemes. However as memory latencies become relatively more expensive the number of independent overlapped instructions required to hide the access times increases. Larger instruction windows are therefore required to detect independent instructions that can execute in parallel with memory access operations.

The pressure to increase window sizes is also driven by the goal of providing ever larger issue widths. However large window and issue width sizes introduces greater complexity in window issue logic. Palacharla *et al* have shown that delays in the issue logic vary quadratically with window and issue width size [11]. Since delays in issue logic will be critical to processor clock there is a need to consider architectures that simplify issue window logic.

To solve the window complexity problem some architectures use separate microclusters. Microclusters may share or have a dedicated instruction window, but each has its own register file and function units. This simplifies window logic by flagging instructions for execution on particular microclusters, and reduces the size of the instruction window, but can limit the number of instructions issued per cycle.

Access decoupling is a latency hiding technique that partitions a programs - statically or dynamically - into two separate instruction streams in order prefetch data aggressively [1, 10, 12]. The instruction streams are loosely coupled. One stream, executed on an *address unit* (AU), prefetches data for the second stream, executed on a *data unit* (DU). Memory accesses can then be pipelined to tolerate large memory latencies provided the two streams can *decoupled* sufficiently.

In principle the same level of prefetching in an access decoupled machine could be achieved with an out-of-order (o-o-o) superscalar architecture. The question is then "why should designers consider using the decoupling paradigm?"

Memory latencies are typically 20-50 cycles whereas arithmetic function latencies are 2-5 cycles (excluding divide and intrinsics). A system could easily tolerate a small degree of o-o-o execution amongst arithmetic operations provided loads could *slip* by a large amount with respect to arithmetic operations. This slippage between arithmetic and load operation is exactly what occurs in a decoupled machine. In effect, we can im-

*This research was supported by EPSRC grant K19723

plement a small instruction window for arithmetic and access operations provided the latter can slip by a large amount with respect to the former.

In answer to our question, we believe that an access decoupled machine can be viewed as a variant of a microcluster architecture with two separate instruction windows. The asynchronously executing units, through code partition and dynamic slippage, combine the benefits of reducing window logic complexity with data prefetching.

In this paper we compare the relationship between window size and memory latency for an access *decoupled machine* (DM) and a *single window* o-o-o *superscalar machine* (SWSM). We also evaluate the size of window required by the SWSM to achieve the same performance as the DM.

The thesis of this paper is developed in the following way. In section 2 we outline the DM and SWSM. In section 4 we describe our simulation technique. In section 3 we discuss the notion of the *effective single window* (ESW) to help explain some of our findings. In section 5 we present the results of our work. Finally in section 6 we draw together our findings and suggest avenues for future work in this area.

2 The Architectural Models

The access **decoupled machine** (DM) modelled in our experiments is shown in Figure 1. The machine consists of two separate out-of-order (o-o-o) superscalar processors, the address unit (AU) and the data unit (DU), responsible for executing the access and data operations. Each unit has is own separate instruction window, functional units and register files. The units can share results by moving data between register files. The number of instructions issued per cycle is determined by the issue width.

The **decoupled memory** lies between the two superscalar pipelines and the rest of the memory system. The decoupled memory receives addresses from the AU and sends them to the memory system. When a referenced value is returned the decoupled memory buffers the value until it is requested by the DU. Requests from the decoupled memory take a single cycle. AU self loads are executed in a similar way. Previously the decoupled memory has been implemented through the use of queues [1, 10].

The **single window superscalar machine** (SWSM) is shown in Figure 2. The architecture is an o-o-o machine with a single instruction window for reordering operations. In each cycle independent operations which are ready to execute are issued to the function units. Unlike the DM the full issue width is available for issuing instructions every cycle. This

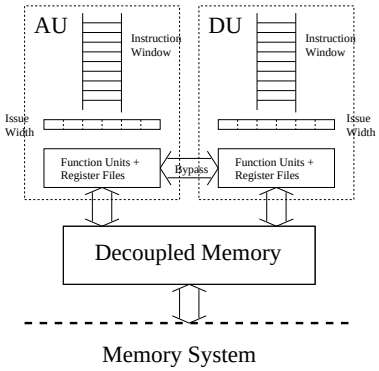

Figure 1: DM

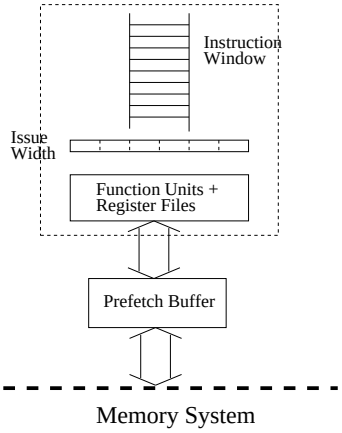

Figure 2: SWSM

means that if the SWSM is able to guarantee 100% utilisation of the full issue width it could outperform the DM.

There are different types of hardware, software and hybrid schemes for data prefetching. For SWSM we use a hybrid scheme. Every memory operation comprises two instructions, a prefetch and an access operation. The prefetch instruction pre-loads data into the prefetch buffer ahead of the access operation. Prefetch operations, unlike software schemes, are allowed to begin execution as soon as runtime resources allow. Using this scheme we gain the benefits of exact address computation with dynamic execution. The prefetch buffer is a fully associative buffer responsible for storing prefetched data. Requests from the prefetch buffer take a 1 cycle.

The **memory system** consists of the main memory but may also be composed of first or second level caches. We are not concerned with a detailed simulation of the memory system; instead we model its execution by considering every access to have a fixed

cost. The fixed cost we refer to as the *memory differential* (MD). The memory differential is the difference in time between a register and memory system access. The purpose of all latency hiding techniques is to eliminate any perceived memory differential.

3 The Effective Single Window (ESW)

An advantage of the DM is that the dynamic slippage between the window of instructions on the AU and DU means that the effective single window size can be greater than the sum of the individual units' window sizes. Figure 2 illustrates the idea of the ESW. The diagram shows the streams for the AU, DU and a single instruction stream. In the single instruction stream the instructions are shown in program order (with later instructions appearing further down the page) and labelled with the units on which they execute in the DM. The diagram shows that, due to the dynamic slippage between the units, the AU is executing instruction further into the instruction stream than the DU. The ESW is the minimum size of window required to buffer all instructions from the oldest DU instruction to the youngest AU instruction.

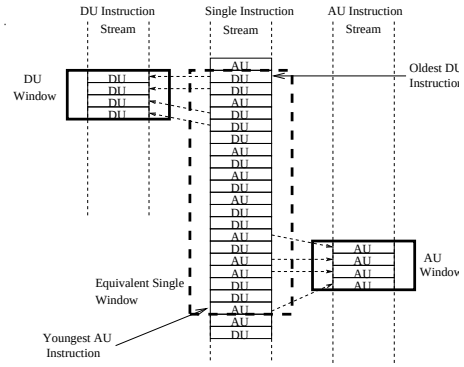

Figure 3: Effective Single Window

4 Simulation Technique

In our experiments we simulated the execution of seven programs from the PERFECT club suite [5] (for a full discussion of the simulation technique see [9]).

Load and store operations on the DM are executed as one instruction on each of the units. On the SWSM loads and stores generate a prefetch and an access operation. Integer and address computations have a 1 cycle cost. Floating point operations take 5 cycles to complete.

There is no speculative execution but we assume loop closing branches have been removed by optimisations like loop unrolling and branch prediction. Data dependency analysis is perfect and false dependencies are removed by renaming. The purpose of examining such an ideal case is to provide the best opportunity for prefetching data, to have high instruction level parallelism (ILP) and to place the greatest pressure on the latency hiding mechanism.

The issue width used for the AU and DU were 4 and 5 respectively. These widths were found to be an optimal configuration in [8]. An issue width of 9 instructions was used for the SWSM.

5 Experimental Results

In this section we present the major findings of the paper. For the purposes of this paper we have selected three representative programs that exhibit the range of observed behavior. The three selected programs were FLO52Q, MDG and TRACK. Table 1 shows the *latency hiding effectiveness* of all seven programs when the window size is unlimited and the memory differential is 60 cycles [1] The latency hiding effectiveness (LHE) is defined as $LHE = T_{perfect}/T_{actual}$ where T_{actual} is the execution time for the DM and $T_{perfect}$ is the execution time for a machine with perfect latency hiding in which each memory access perceives a single cycle latency. It can be seen there are three bands in which the programs are highly (80-100%), moderately (40-60%) and poorly ($< 40\%$) effective at hiding latency. It can be seen that the three programs fall within each of the bands.

Prog.	DM Window Size					
	∞	1	20	40	50	100
TRFD	1.00	0.90	0.53	0.41	0.42	0.45
ADM	1.00	0.72	0.43	0.39	0.38	0.39
FLO52Q	0.97	0.82	0.74	0.70	0.70	0.73
DYFESM	0.87	0.81	0.53	0.48	0.49	0.49
QCD2	0.55	0.76	0.46	0.43	0.43	0.44
MDG	0.48	0.56	0.33	0.32	0.34	0.40
TRACK	0.22	0.45	0.23	0.23	0.22	0.22

Table 1: Latency Hiding Effectiveness for MD=60 cycles

Figures 4, 5 and 6 show the variation in speedup with window size for the access decoupled and superscalar architecture when the memory differential is 0

[1] An MD of 60 was chosen because it is comparable to the cost of a second level cache miss (the pentium Pro has 50 cycle L2 miss latency[2]) and it assumes a weak memory system capable of capturing no locality. In practice for a high performance architecture the memory system will be able to reduce the average access time by using first and second level caches.

and 60 cycles. When MD is 0 we see that for small window sizes the DM performs better than the SWSM with same window size. This is due to the DM having two windows for reordering operations compared to one for the SWSM. This means there are fewer resource conflicts for window slots and greater scope for reordering operations. It will also be noticed that the graphs show the law of diminishing returns for increasing window size; once window sizes are above 10 instructions, doubling the size does not double the speedup. All the programs reach a cut-off point for window sizes between 40 and 80 instructions when the SWSM performs more effectively. This is due the benefit of the larger instruction issue width available to the SWSM. This benefit is only realised once the instruction window is large enough to utilise the available issue width.

In Figures 4, 5 and 6 we see that once MD reaches 60 cycles there is no cut-off point when the SWSM performs better than the DM. This results applies even for very large windows of 100 instruction slots. The difference between the performance of the two machines must be solely due to the more effective data prefetching of the DM. Operations on the SWSM which on DM would have been executed on the DU, are causing address computations to execute later, reducing the pipelining of memory accesses and decreasing the effectiveness of the data prefetching. The difference in performance between the two machines is also dependent on the type of program. For FLO52Q which is highly parallel the gap between the DM and SWSM is large. However, for TRACK which has little parallelism there is little difference between the two architectures.

We can state therefore that for all the programs we have simulated the DM is more effective at hiding large memory latencies than the SWSM. The difference in performance is dependent on the parallelism and decoupling in the program. Programs that decouple well show the largest improvement in performance for the DM.

Figures 7, 8 and 9 show, for a range of memory differentials, the ratio of the SWSM and DM window sizes that yield equivalent performance. We will refer to this ratio as the *equivalent window ratio*. The ratio was derived by projecting from the DM graph to SWSM graph in Figures 4,5 and 6. The graphs show the way in which the ratio varies as a function of the memory latency. It can be seen that as latencies approach 60 cycles the ratio gets larger. This is due solely to the more effective data prefetching of the access decoupled machine. As the memory latency increases, the DU waits longer for data to arrive and the slippage between the two units grows. This means conceptually that the effective single window size (see Figure 3) for DM gets larger. In order for the SWSM to achieve equivalent performance it requires a correspondingly larger window.

The graphs in Figures 7, 8 and 9 also show that as the DM window size is increased the equivalent window ratio reduces. This is due to the SWSM architecture being able to reorder operations to a similar degree as the DM, and also the benefits of the larger issue width.

Significantly it can be observed that for a realistic DM window size of 30 instructions and a memory latency of 60 cycles, the required increase in window size for equivalent SWSM performance is dependent on the program, but lies between 2.5 to 5. Experiments with the other benchmark programs have also been found to fall within this range. Larger windows introduce extra hardware complexity and longer window logic delays [2]. We can state therefore that the DM requires smaller instruction windows and hence simpler window logic.

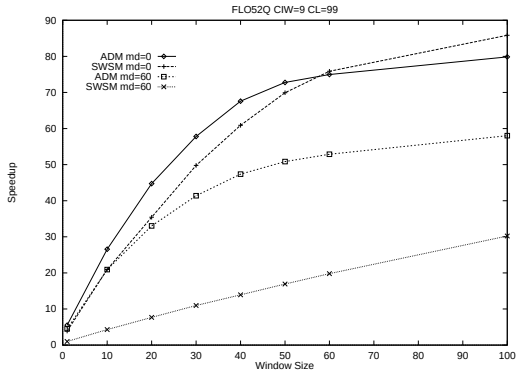

Figure 4: FLO52Q

Having shown that the DM performs consistently better than the SWSM we now compare the latency hiding effectiveness of the DM against a perfect latency hiding technique (one in which all the memory differential is hidden). Table 1 shows the measured LHE for different window sizes when the memory differential is 60 cycles.

The results show that when window sizes are small increasing the window size causes a reduction in the LHE. This is due to the extra parallelism on the

[2] In [11] is was shown that delays vary quadratically with window size.

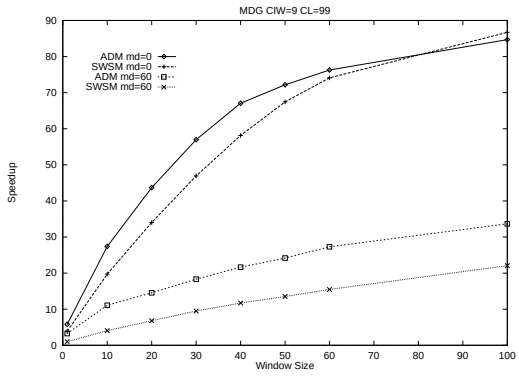

Figure 5: MDG

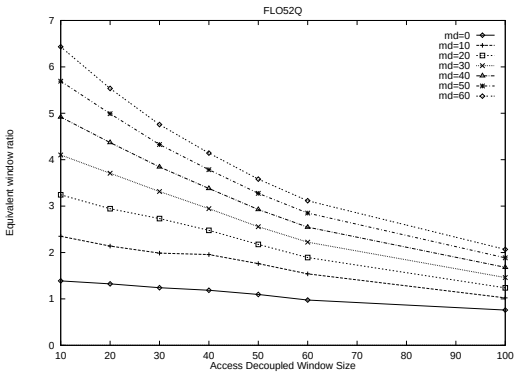

Figure 7: FLO52Q

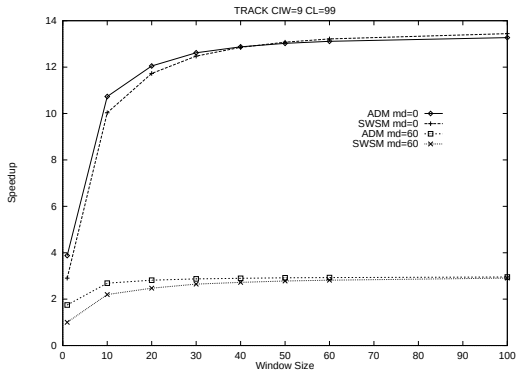

Figure 6: TRACK

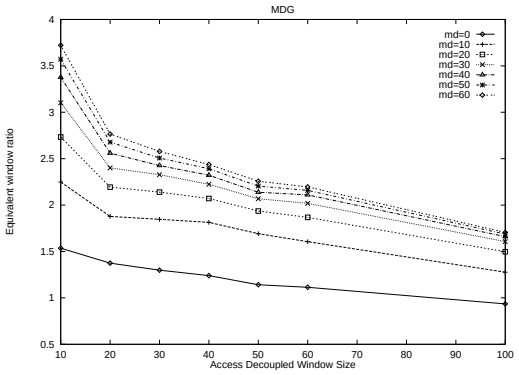

Figure 8: MDG

DU placing greater pressure on the memory system. The AU window is not yet large enough allow the AU to pipeline accesses sufficiently to hide the latency. However there eventually comes a point when the larger window size allows more operations to execute in parallel and the LHE starts to improve. For six of the programs this point is between 40 to 50 instructions. This result suggests that for realistic window sizes (1 to 30 instructions), increasing the window size will result in the latency hiding mechanism of the DM performing less effectively. Table 1 also shows that even with large window sizes we do not approach the LHE of an DM with unlimited resources.

Our findings show that for realistic window sizes the DM can hide latencies better than SWSM but that as the window size increases its effectiveness at hiding latency deteriorates. This illustrates the tensions that exist between having greater parallelism and the access decoupling mechanism. As the window size get larger, the instruction level parallelism increases and the execution times fall. However the extra parallelism places greater pressure on the decoupling mechanism resulting in a decrease in LHE. The result is that more of the critical path time is now composed of the memory differential. There comes a point however, when the AU window is large enough to compensate for the extra parallelism on the DU, and more address operations can be pipelined to hide the latency.

In the short to medium term high performance architectures will have window sizes in the range that shows a reduction in the LHE. In future work we will investigate mechanisms to improve the latency hiding of the DM. One possibility is a bypass mechanism which captures the temporal locality exposed by decoupling [7].

6 Conclusion and Future Work

This paper has focused on two objectives in the design space of future microprocessors; the need to

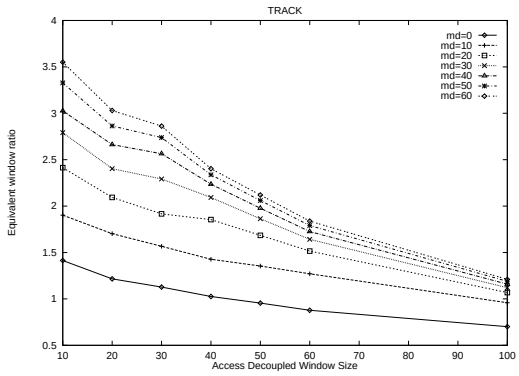

Figure 9: TRACK

hide large memory latencies and the need to reduce the complexity of window issue logic. We have investigated the use of data prefetching on an access decoupled machine and a single window o-o-o superscalar architecture.

In this paper we have examined the relationship between memory latency, window size and speedup for the two architectures. In order to remove the impact of other architectural issues we have assumed an idealistic environment. This environment provides good conditions for data prefetching, high levels of ILP and places the greatest pressure on the latency hiding mechanism.

We have found that the DM is more effective at hiding memory latency than the SWSM. For large memory differentials (60 cycles) we have found that even for large window sizes of 100 instructions, the DM consistently performs better than the SWSM. Our results have also shown that to achieve the same speedup as an DM the SWSM needs a window size between 2.5 to 5 larger. The increase in window size required to achieve equivalent performance on the SWSM was also found to increase with larger latencies.

To explain some of our findings we have introduced the concept of the effective single window. The ESW conceptually illustrates how the DM is able to perform better than an architecture with twice the size of instruction window.

Our results have also shown how the latency hiding effectiveness of the DM decreases as the window size increases to 40 instructions. Though the speedup did increase with larger window size the DM was not found to be as effective at hiding latency. However when windows were greater than 50 instructions the LHE was found to improve. This behavior illustrates the tensions that exist between higher ILP and the access decoupling mechanism.

This paper has shown that access decoupling can combine the benefits of latency hiding with simplifying the window logic complexity. We conclude therefore that there is a need for further work in the use of access decoupling. In future work we will examine the effects of code expansion on the DM and SWSM. We will also compare the difference in performance between a static and dynamic partition of the code on the DM.

References

[1] A. Berrached, L.D. Coraor, and P.T. Hulina. A Decoupled Access/Execute Architecture for Efficient Access of Structured Data. In *Proc. of the 26th Hawai Int. Conf. on System Sciences*, volume 1, pp. 438–47, Jan. 1993.

[2] D. Bhandarkar and J. Ding. Performance Characterisation of the Pentium Pro Processor. In *Proc. of the 3rd Int. Symp. on High Performance Computer Architecture*, Feb. 1997.

[3] D. Callahan, K. Kennedy, and A. Porterfield. Software Prefetching. In *4th Ann. Symp. on Parallel Languages and Operating Systems*, pp. 40–52, Apr. 1991.

[4] Tzicker Chiueh. Sunder : A Programmable Hardware Prefetch Architecture for Numerical Loops. In *Proc. Supercomputing '94*, pp. 488–497, Nov. 1994.

[5] M. Berry *et al.* The Perfect Club Benchmarks, Effective Performance Evaluation of Supercomputers. Tech. report 827, CSRD, University of Illinois, Urbana-Champaign, Urbana, Illinois., May 1989.

[6] J.W.C. FU and J.H. Patel. Data Prefetching Strategies for Vector Cache Memories. In *Proceedings. The Fifth Int. Parallel Processing Symp.*, pp. 555–560, Apr. 1991.

[7] G.P. Jones and N.P. Topham. A Limitation Study into Access Decoupling. *Euro-Par'97*, Springer Verlag, Vol. LNCS 1300, pp. 1102–1111, Aug. 1997.

[8] G.P. Jones and N.P. Topham. The Effect of Restricted Instruction Issue on an Access Decoupled Machine. *ParCo97*, Germany, Sept. 1997.

[9] G.P. Jones and N.P. Topham. Simplifying Hardware for Out of Order Execution using the Decoupling Paradigm. Tech. report CSG-32-97, Edinburgh University, Sept. 1997.

[10] M.K.Farrens and A.R.Pleszkun. Implementation of the PIPE Processor. *IEEE Computer*, pp. 65–70, Jan. 1991.

[11] S. Palacharla, N.P. Jouppi, and J.E. Smith. Complexity-Effective Superscalar Processors. In *24th Ann. Int. Symp. on Computer Architecture*, 1997.

[12] Wm A. Wulf. An Evaluation of the WM Architecture. In *Proc. Int. Symp. on Computer Architecture*, May 1992.