Data Speculative Multithreaded Architecture

Pedro Marcuello and Antonio González

Universitat Politècnica de Catalunya
Departament d'Arquitectura de Computadors
c/ Jordi Girona 1-3, Mòdul D6
08034 Barcelona (Spain)

Email: {pmarcue,antonio}@ac.upc.es

Abstract

In this paper we present a novel processor microarchitecture that relieves three of the most important bottlenecks of superscalar processors: the serialization imposed by true dependences, the relatively small window size and the instruction fetch bandwidth. The new architecture executes simultaneously multiple threads of control obtained from a single program by means of control speculation techniques that do not require any compiler/user support neither any special feature in the instruction set architecture. The multiple simultaneous threads execute different iterations of the same loop, which require the same fetch bandwidth as a single thread since they share the same code. Inter-thread dependences as well as the values that flow through them are speculated by means of data prediction techniques. The preliminary evaluation results show a significant speed-up when compared with a superscalar processor. In fact, the new processor architecture can achieve an IPC (instructions per cycle) rate even larger than the peak fetch bandwidth.

1. Introduction

Several studies on the limits of the instruction-level parallelism (ILP) that current superscalar organizations can attain show that it is rather limited when a realistic configuration is considered. Three of the most important bottlenecks that cause this limitation are: the serialization imposed by data dependences, the instruction window size and the fetch bandwidth. In this paper, we propose a novel microarchitecture, called *Data Speculative Multithreaded Architecture (DaSM)*, that relieves the three bottlenecks mentioned above without any modification in the instruction set architecture (ordinary programs compiled for a superscalar processor implementation can run in this).

The amount of ILP that a superscalar processor can exploit is highly dependent on the size of the instruction window. Nevertheless, the approach used by current superscalar processors to build bigger instruction windows is not scalable and finds to main problems: the branch prediction accuracy and the growth in the complexity and the delay of the issue logic. Our processor implements an effective large instruction window that is made up of several non-adjacent smaller windows. Each small window is built using the conventional control speculation approach whereas the creation of a new window is based on speculating on highly predictable branches (e.g., branches that close loops). Each small window corresponds to a different thread of control of the same program. These threads, which are not necessarily independent, are created from a single sequential program completely by hardware without compiler intervention, unlike previous proposals.

In the matter of the execution ordering constraints imposed by data dependences (true dependences), a lot of data speculation techniques have recently emerged as a new family of techniques that can provide a significant boost in ILP (see [4][5][6][10] among others). The proposed architecture speculates on inter-thread data dependences and the data that flows through them. That is, for each new speculative thread it predicts which register and memory dependences it has with previous threads in the control flow and which values are going to flow through such dependences. The thread is then executed obeying the predicted dependences and using the predicted values to avoid waiting for the actual data.

Finally, since the multiple threads of control are obtained by speculating on loop-closing branches, all simultaneously active threads of control share the same code (the loop body), and thus, a simple fetch engine can feed all the threads with the same fetch bandwidth as that required by a single thread.

The rest of this paper is organized as follows. The DaSM architecture is presented in section 2. Some performance figures of the DaSM architecture are discussed in section 3. The related work is reviewed in section 4. Finally, section 5 summarizes the main contributions of this paper.

1089-6503/98 $10.00

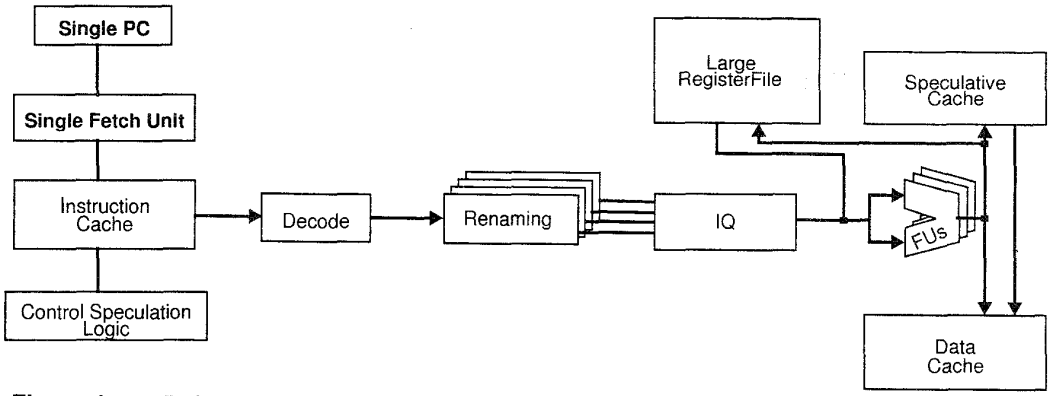

Figure 1: The DaSM microarchitecture

2. The data speculative multithreaded architecture

The DaSM hardware architecture (see figure 1) is based on a simultaneous multithreaded processor [14] with some extensions to create dynamically and to speculate on multiple threads of control (threads for short) obtained from a single sequential program, and to speculate on inter-thread data dependences/values through registers and memory. Thus, the distinguishing features of the novel architecture are the extensive use of speculation on control and data dependences, as it is reflected in its name. Both types of speculation are completely performed by the hardware without requiring any compiler support. Thus, the object code can be specified in any conventional instruction set architecture (ISA), without any additional extension. Each thread executes out-of-order.

Since all the threads share the same code, a single fetch engine can fetch the instructions of the loop just once and replicate them as many times as number of active threads. Then, each instruction is renamed using a different register map table and dispatched to the instruction queue.

The large instruction window consists of several non-contiguous small windows, each one corresponding to a different threads of control which are built at run time through control speculation. A key issue to build a large effective instruction window is to speculate only on highly predictable branches (e.g. branches that close loops) The mechanism used to dynamically extract multiple threads of control from a sequential program is largely described in [13].

The different threads simultaneously executed do not have to be independent, so data dependences through registers and memory can exist. The data dependences through memory are predicted by taking advantage of the highly predictability of memory addresses (see [5]). Predicting dependences through registers is based on the observation that the architected registers that are live at the beginning of every iteration is usually the same for a given loop. Prediction of data values is based of the observation that a significant part of executions of a static instruction produce the same value as the previous execution of the same instruction or that value plus a stride. Data dependences and data values are predicted by means of a table that is called iteration table which is indexed by the loop identifier (the target address of a backward branch).

The processor has a large file of physical registers that are shared by all the threads in the same way (as the simultaneous multithreading). Each thread has its own register map table (Rmap for short). A Rmap entry may contain a special value, NIL, that indicates that this logical register is not currently mapped to any physical register. The Rmap for the speculative threads is initialized with the same value that the non speculative one (if the register will not be used by that thread), with NIL (if the thread will perform a write before a read over that register) and a new physical register and initialized with the predicted value (if is a live-in register). When the non-speculative thread reaches the beginning of the next thread, all the non-shared registers are freed.

The processor provides support to store multiple states of memory locations by means of a special first level speculative data cache. This level consists of several private caches associated to each context, and a shared cache, which is shared by all the contexts.

Private caches are initialized with those memory values that are live and predictable, and are also used to keep all memory updates performed by speculative threads until they are committed.

Dependences among memory references with a non-zero stride are managed by t he shared cache, which undertakes the serialization of the memory accesses in the right order.

If a misspeculation occurs, all the speculative threads starting on the one that produces it are squashed.

3. Performance evaluation

In this section we present the results of a preliminary evaluation of the DaSM architecture. The objective is to demonstrate the potential of the new architecture to exploit more ILP than superscalar processors.

The DaSM architecture has been evaluated and compared to a superscalar architecture through trace-driven simulation of the SPEC95 benchmark suite. The programs have been compiled for a DEC AphaStation 600 5/266 with a 21164 processor with full optimization and instrumented by means of the Atom tool. A cycle-by-cycle simulation is performed in order to obtain accurate timing results. Because of the detail at which simulation is carried out the simulator is slow, so we have simulated 100 million of instructions for each benchmark after skipping the initial part that corresponds to initialization of data structures.

For the DaSM architecture we have assumed that it has 4 contexts, an issue bandwidth of 4 instructions per cycle for each context, 4 entries in the iteration table, a shared cache with 128 entries and 4 private caches with 64 entries each. Every context has a local reorder buffer with 64 entries. The superscalar processor can issue 4 instruction per cycle. The remaining parameters of the both architectures are the same: (fetch bandwidth: up to 4 instructions, 256 physical registers, perfect branch prediction, ideal L1 cache for the superscalar and L2 for DaSM and the following functional units (latency in brackets): 8 simple integer (1), 4 integer multiplication (2), 5 memory (2), 6 simple FP (1), 3 FP multiplication (4) and 2 FP division (17)))

3.1. Performance figures

The evaluation of the DaSM and superscalar architectures is summarized in Table 1. The first column of this table shows the average number of committed instructions per cycle in the DaSM architecture. It can be seen that for the FP programs the IPC is significantly higher than for integer programs. For most of the FP programs, the DaSM architecture achieves an IPC even higher than the fetch bandwidth. This confirms the potential benefits of the fetch mechanism in terms of the reduction in fetch bandwidth requirements.

The second column shows the average number of active threads per cycle (TPC) that are correctly speculated. The average TPC for FP programs is 3, out of a maximum of 4, whereas for the integer programs it is much lower (1.4).

The third column shows the percentage of correctly speculated threads. This percentage is in general quite high, even for integer programs, which confirms that the speculation mechanism is quite accurate in identifying predictable threads (in terms of dependences and values).

The fourth column is the IPC for the conventional superscalar processor. The relatively high IPC is explained by the perfect branch prediction, ideal cache memory and the very large number of functional units.

The last column shows the ratio between the IPC of the DaSM and that of the superscalar architecture. It can be seen that the DaSM processor is in average 80% faster than the superscalar processor for FP programs and 18% faster for the integer programs.

		DaSM			Superscalar	Speedup of DaSM
		IPC	TPC	pred. hit ratio	IPC	
FP	tomcatv	7.9	3.6	0.98	3.6	2.2
	swim	5.6	2.7	0.80	3.6	1.6
	hydro2d	5.8	3.4	0.96	3.4	1.7
	mgrid	8.2	3.6	0.97	3.7	2.3
	applu	3.8	1.7	0.86	3.1	1.2
INT	m88ksim	3.9	2.3	0.92	2.8	1.4
	vortex	3.5	1.2	0.78	3.4	1.1
	go	3.3	1.0	0.30	3.3	1.0
	ijpeg	4.1	1.2	0.72	3.5	1.2

Table 1: Performance statistics

The main conclusion of these results is that the DaSM architecture implements a novel execution model that has significant potential to boost the ILP that can be exploited by dynamic techniques. Data speculation allows the processor to dynamically extract several threads of control from a single program. In addition, the reuse of code among threads allows the processor to go beyond the barrier imposed by the fetch bandwidth in conventional organizations. This preliminary evaluation also suggests that data speculation is more effective for FP programs. However, it remains to be researched which are the main reasons for the low TPC rate in integer programs and potential alternative solutions to improve it.

4. Related work

There are few proposals in the literature dealing with the dynamic management of a large window that consists of several threads of control not necessary independent among them obtained from a sequential program. Pioneer work on this area was the Expandable Split Window paradigm [3] and the follow-up work on Multiscalar processors [11]. Other proposals are the he SPSM architecture [2]; the Superthreaded architecture [12]; the Multithreaded Decoupled architecture [1]; Trace processors [9] ; the Dependence Speculative Multithreaded Architecture [7] and the Speculative Multithreaded Architecture[8]. There are important

differences between the DaSM microarchitecture and those previous proposals, like it does not require some addition/ extension of the ISA and the extensive use of speculation techniques.

Also, the DaSM architecture reduces the fetch bandwidth requirements by taking advantage of the fact that simultaneously active threads process the same code with different data. A similar feature is exploited by the CONDEL architecture [15] and the dynamic vectorization approach proposed in [16]. However, those approaches are more restrictive than the one used by DaSM processors since our processor is not limited by the loop size

5. Conclusions

We have presented a novel architecture, which is called Data Speculative Architecture (DaSM). A novel feature of such architecture is its ability to dynamically extract and execute multiple threads of control from a single sequential program written in a conventional ISA and without requiring any compiler support. Multiple concurrent threads execute different iterations of the same loop. These threads are not necessarily independent (usually they are dependent) but inter-thread data dependences are resolved by speculation techniques: dependences and values that flow through them are predicted by means of a history table. In this way, loops that are not parallelizable by the compiler can be executed in parallel if data dependences and data values are correctly predicted. The second main feature of the architecture is that the additional ILP due to inter-thread parallelism hardly increases the fetch bandwidth requirements since multiple threads share the same code.

In a preliminary evaluation of the DaSM processor, a 4-context configuration has shown to provide significant benefits when compared with a superscalar processor with the same number of functional units and fetch bandwidth. It has been observed that the processor can exploit in average about 3 correct threads for FP applications and 1.4 for integer applications. It has also been shown that the DaSM processor can achieve an IPC higher than the fetch bandwidth.

We can conclude that the combination of data speculation and multiple threads of control is a promising alternative to relieve the most critical bottlenecks of current superscalar microprocessors: data dependences, a relatively small instruction window and a limited fetch bandwidth.

6. Acknowledgments

This work has been supported by the Spanish Ministry of Education under grants CICYT TIC 429/95 and AP96-52274600. The research described in this paper has been developed using the resources of the European Center for Parallelism of Barcelona (CEPBA).

7. References

[1] M.N. Dorojevets and V.G. Oklobdzija, "Multithreaded Decoupled Architecture", *Int. J. of High Speed Computing*, 7(3), pp. 465-480, 1995.

[2] P.K. Dubey, K. O'Brien, K.M. O'Brien and C. Barton, "Single-Program Speculative Multithreading (SPSM) Architecture: Compiler-Assisted Fine-Grained Multithreading", in *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques*, pp. 109-121, 1995.

[3] M. Franklin and G.S. Sohi, "The Expandable Split Window Paradigm for Exploiting Fine Grain Parallelism", in *Proc. of Int. Symp. on Computer Architecture*, pp. 58-67, 1992.

[4] J. González and A. González, "Memory Address Prediction for Data Speculation", in *Proc. of EURO-PAR 97 Workshop on ILP*, 1997.

[5] J. González and A. González, "Speculative Execution via Address Prediction and Data Prefetching", in *Proc of 11th. ACM Int. Conf. on Supercomputing*, 1997.

[6] M.H. Lipasti, C.B. Wilkerson and J.P. Shen, "Value Locality and Load Value Prediction", in *Proc. of the 7th. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 138-147, Oct. 1996.

[7] P. Marcuello and A. González, "Control and Data Dependence Speculation in Multithreaded Processors", in *Proc. of the Workshop on Mulithreaded Execution Architecture and Compilation* held in conjunction with HPCA-4, available as tech. report CS-98-102 from www.cs.colostate.edu, 1998

[8] P. Marcuello, A. González and J. Tubella, "Speculative Multithreaded Processors", in *Proc. of the 12th Int. Conf. on Supercomputing*, Jul. 1998.

[9] E. Rotenberg, S. Bennett and J.E. Smith, "Trace Processors", in *Proc. of the 30th Int. Symp. on Microarchitecture*, 1997.

[10] Y. Sazeides, S. Vassiliadis and J.E. Smith, "The Performance Potential of Data Dependence Speculation & Collapsing", in *Proc. of Int. Symp. on Microarchitecture*, pp. 238-247, Dec. 1996.

[11] G.S. Sohi, S.E. Breach and T.N. Vijaykumar,"Multiscalar Processors", in *Proc. of the Int. Symp. on Computer Architecture*, pp. 414-425, 1995.

[12] J-Y. Tsai and P-C. Yew, "The Superthreaded Architecture: Thread Pipelining with Run-Time Data Dependence Checking and Control Speculation", in *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques*, pp. 35-46, 1996.

[13] J. Tubella and A. González, "Control Speculation in Multithreaded Processors through Dynamic Loop Detection", in *Proc. of the 4th Int. Symp. on High-Performance Computer Architecture (HPCA-4)*, pp. 14-23, 1998.

[14] D.M. Tullsen, S.J. Eggers and H.M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism", in *Proc. of the Int. Symp. on Computer Architecture*, pp. 392-403, 1995.

[15] A. K. Uht, "Concurrency Extraction via Hardware Methods Executing the Static Instruction Stream", *IEEE Trans. on Computers*, vol 41, July 1992.

[16] S. Vajapeyam and T. Mitra, "Improving Superscalar Instruction Dispatch and Issue by Exploiting Dynamic Code Sequences", in *Proc. of the Int. Symp. on Computer Architecture*, pp. 1-12, 1997.