The Decoupled-Style Prefetch Architecture

Kevin D. Rich and Matthew K. Farrens

University of California at Davis

Abstract. Decoupled processing seeks to dynamically schedule memory accesses in order to tolerate memory latency by prefetching operands. Since decoupled processors can not speculatively issue memory operations, control flow operations can significantly impact their ability to prefetch data. The prefetching architecture proposed here seeks to leverage the dynamic scheduling benefits of decoupled processing while allowing memory operations to be speculatively invoked. The prefetching mechanism is evaluated using the SPEC95 suite of benchmarks and significant reductions in cache miss rate are achieved, resulting in speed-ups of over 40% of peak for most of the inputs.

1 Introduction

Decoupled access/execute (DAE) architectures [6, 8, 7] rely on an intelligent compiler to separate the memory access and execution components of a program into independent instruction streams so they can be executed on separate processors. These processors are loosely coupled through architectural queues, which provide an asynchronous message passing channel. Using the queues to maintain the logical order of instructions, the two instruction streams are able to "slip" with respect to one another, allowing the access processor to lead the execute processor. This effectively migrates data fetching to a earlier time and produces what amounts to *dynamic* loop unrolling and pipelining.

Unfortunately, the strict FIFO nature of the queues limits the performance of decoupled architectures, since they cannot easily exploit branch prediction and speculative execution. In this paper an approach is proposed which makes use of the access instruction stream identification technology developed for the decoupled compiler to create an access stream which will prefetch data into the cache in order to reduce the occurrence of first-reference cache misses (a category of misses not generally addressed by current prefetching techniques.)

In the proposed architecture (D-SPA), the main processor (MP) is augmented with a *prefetch* processor (PFP), whose job is essentially that of decoupling's access processor - issue memory requests prior to the data being needed by the MP. However, the requirement that the data accesses be correct is removed since the prefetches only bring data into the data cache and do not effect MP state.

2 Background

Memory latency is not a new problem and many prefetching schemes have been explored to address it. Most of these proposals augment a uniprocessor with a

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 989–993, 2000.

simple, parameterizable prefetch-engine. The compiler is responsible for analyzing the source code and identifying regular access patterns that can be described with parameters of the form `<start address>`, `<stride>`, and `<stop address>`. Examples of this include work by Chiueh [3], Chen [2], VanderWiel and Lilja [9].

The prefetch processor of D-SPA differs in that the hardware is considerably more powerful. Instead of having the prefetch engine execute a small set of parameterizable routines, the prefetcher is a fully functional integer-only processor — essentially the access processor from the DAE architecture. Like the access processor, it is loaded with its own executable. The prefetcher relies on the main processor only for flow control information and, when absolutely necessary, some data (function parameters and return values). Unlike the access processor, the prefetch processor only issues non-binding prefetches. The prefetched data is loaded into a cache or prefetch buffer, not into the processor's register file or a queue. Whether or not the data is ever used is solely a performance issue — it has no impact on program correctness.

3 The Decoupled-Style Prefetch Architecture

D-SPA was designed to take advantage of the compilation techniques developed for decoupled processing, while eliminating the restrictions on the execution model which prevent speculative execution. This paper presents only an overview of the architecture - for a more detailed description of the proposal and it's effectiveness please see [4].

Conceptually, D-SPA can be seen as a merging of a standard uniprocessor and the DAE architecture. It consists of a uniprocessor (MP) augmented with a prefetching processor (PFP). The PFP employs a subset of the same base RISC-ISA as the MP. The MP performs loads and stores in the usual manner, oblivious to the existence of the PFP. Branches are implemented using the <external branch>/<branch from queue> technique employed to co-ordinate branch decisions in decoupling, except that the MP hardware sends the results of every conditional branch to the branch queue, and all conditional branch operations on the PFP are `bfq` instructions. The PFP prefetch instructions are similar to typical load operations except that the data is brought into the cache or a prefetch buffer.

In addition to issuing memory requests purely for the purpose of prefetching, the PFP needs to access memory in order to reference variables that it requires to perform address calculations. In order to ensure that the PFP makes no references which could impact MP correctness, it must be restricted from writing to shared memory locations.

When considering how to enforce this restriction, it is important to recall that D-SPA is intended to be an augmentation of an existing RISC processor, not an entirely new processing model. As a result, modifying the memory system to provide exclusive access or other capabilities is not an option. Instead, the execution model must be modified to ensure that the PFP does not write shared/global variables. This can be accomplished by making the compiler responsible for ensuring that the target location of any PFP store is on the PFP's

stack; if this can not be guaranteed at compile time, the store is not included in the PFP executable. (To reduce the limitations on slip, each processor maintains a private run-time stack.) This may result in the PFP producing incorrect memory requests, but since all of the PFP's prefetches are non-binding these errors will not affect program correctness.

The ability to run uniprocessor binaries on D-SPA is achieved via a few minor modifications to the hardware necessary to address the external branch implementation and the addition of the copy queue. In order to run D-SPA binaries on an un-augmented MP the existence of copy operations in the D-SPA binary must be addressed. For a detailed description see [4].

By specifying the architectural and execution models to closely model those of decoupled processing, it is possible to leverage the compilation techniques developed for DAE processing. The most significant task is the identification of those instructions necessary to compute addresses — this must be done in order to generate the PFP instruction stream.

Function calls present perhaps the biggest challenge for the compiler. In order to accurately prefetch within a function which accepts parameters, it may be necessary for the PFP to have the parameter values which are calculated by the MP and must be communicated to the PFP via the copy queue. The PFP may also require a copy of the return value of a function in order to generate prefetch addresses (this is identified during def-use analysis). Therefore, the compiler must insert the copy operations necessary to communicate function parameters and return values from the MP to the PFP.

4 Results

The experiments performed use a combination of trace-driven and execution-driven simulation techniques, because it provides a reasonable first-order approximation without requiring extensive compiler modifications or the creation of a special multiprocessor simulator. The cycle-level *sim-outorder* SimpleScalar simulator was used to simulate the execution of the benchmarks [1].

Sixteen of the eighteen SPEC95 benchmarks were used in order perform the preliminary analyses of D-SPA (two of the eight integer benchmarks were omitted because of incompatibilities with the simulation environment.)

Due to disk space limitations, full runs of these benchmarks was not feasible; instead, a "window" of several million instructions was selected on which to perform the simulation, these windows were selected based on work performed by Skadron [5].

Two different cache configurations were employed. The first consisted of an 8K (256 blocks by 8 words/block) direct-mapped, single cycle, L1 cache backed by a 256K (1024 blocks by 16 words/blocks) 4-way set-associative, six cycle, L2 cache. The second consisted of a 16K (512 blocks by 8 words/block) direct-mapped, single cycle, L1 cache backed by a 256K (1024 blocks by 16 words/blocks) 4-way set-associative, six cycle, L2 cache.

The small cache sizes were used because the SPEC95 benchmarks generally exhibit very low data cache miss rates. Since the goal of this research is to

determine if number of data cache misses can be reduced, there needs to be some data cache misses. Future studies are planned with different applications and more representative cache configurations.

The memory latency used was 18 cycles for the first cache-to-main memory bus transfer unit and 2 cycles for each other unit in a single cache-block request. An additional consideration is the number of memory ports available. The default SimpleScalar configuration provides two memory ports; however, preliminary studies in which the two processors shared two ports indicated that performance suffered. The results presented in this chapter use a configuration with three memory ports, with two dedicated to the MP and one dedicated to the PFP.

Four different prefetching schemes were investigated. All four issue prefetches for loads of global data. The four possible combinations are derived by including or excluding prefetches for references to stack-based data, and including or excluding prefetches for store operations. The simulations were done using both the 8K and 16K cache configurations described previously. Full details of these simulations are available in [4] - in this paper there is only room for an overview of the results.

Ideally the PFP will be running (will "slip") far enough ahead of the MP to effectively prefetch, but not so far ahead that it adversely impacts cache performance. With unbounded slip, the PFP reduces cache misses (and as a result, achieves a large percentage of the peak speed-up) for many of the benchmarks. However, prefetching results in an *increase* in the cache miss rate for several others.

In order to determine if the behavior of D-SPA on the poor performing benchmarks is in fact attributable to excessive slip, a set of experiments with metered slip was performed. The slip metering approach employed here sets a threshold on the number of outstanding prefetches permitted. When this quantity is hit, the PFP stalls until it drops below the threshold. For these experiments, the slip was metered at 10, 25, 50, and 100 outstanding prefetches. With slip metered at 50, D-SPA achieves over 40% of peak speed-up on ten of the sixteen benchmarks, and between 25% and 40% of peak speed-up on three others.

The overall impact of metering slip was substantial. In the case of tomcatv for instance, the negative impact of prefetching can be directly attributed to excessive slip. With unmetered slip, prefetching results in increased execution time. When slip is metered at 50 outstanding prefetches, 40% of peak speed-up is achieved. For applu and hyrdro2d the impact of metering is more dramatic. Metering slip results in over 60% of peak speed-up being achieved for applu, while for hydro2d over 75% of peak speed-up is achieved. Metering has a lesser impact on apsi, fpppp, mgrid, and swim, but in all cases it significantly improves the performance over the unmetered case.

5 Conclusion

In order to exploit the capabilities of decoupled processing, while removing some of the performance constraining restrictions of its execution model, the

decoupled-style prefetch architecture (D-SPA) is proposed. D-SPA shares several characteristics with the decoupled processing model, including the use of branch and copy queues for communication between the processors. Unlike the decoupled model, it does not employ queues in its interface with memory. The elimination of the memory queues, and the use of non-binding prefetches, allows for the use of speculative execution in the PFP, reducing the slip limiting effect of conditional branch operations. By specifying D-SPA's architectural and execution model to closely resemble that of decoupled processing, the compilation techniques developed for decoupled processing can be employed in a D-SPA compiler. Despite the existence of the branch and copy queues, binary compatibility with an existing RISC instruction set architecture is accomplished with a modicum of effort.

D-SPA reduces the restrictions on slip by employing speculative execution and issuing non-binding prefetches. As a result, it may be possible for the PFP to slip too far ahead of the MP, adversely impacting performance. In order to counter this, techniques to constrain, or meter, slip were developed.

The experiments performed here have shown that D-SPA has the potential to significantly reduce the frequency of data cache misses. While these experiments are only a first order approximation of the performance of D-SPA, they indicate that further evaluation is warranted.

References

[1] Doug Burger and Todd M. Austin. The SimpleScalar tool set, version 2.0. Technical Report 1342, University of Wisconsin-Madison, 1997.

[2] Tien-Fu Chen. An effective programmable prefetch engine for on-chip caches. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, 1995.

[3] T.-C. Chiueh. Sunder: A programmable hardware prefetch architecture for numerical loops. In IEEE, editor, *Proceedings, Supercomputing '94: Washington, DC, November 14–18, 1994*, Supercomputing, pages 488–497, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1994. IEEE Computer Society Press.

[4] Kevin D. Rich. *Compiler Techniques for Evaluating and Extending Decoupled Architectures*. PhD thesis, University of California at Davis, 2000.

[5] Kevin Skadron. *Characterizing and Removing Branch Mispredictions*. PhD thesis, Princeton University, June 1999.

[6] James E. Smith. Dynamic instruction scheduling and the Astronautics ZS-1. *IEEE Computer*, 22(7):21–35, July 1989.

[7] N.P. Topham and K. McDougall. Performance of the decoupled ACRI-1 architecture: the Perfect Club. In *Proceedings of High Performance Computing - Europe*, 1995.

[8] Gary S. Tyson. *Evaluation of a Scalable Decoupled Microprocessor Design*. PhD thesis, University of California at Davis, 1997.

[9] Steven P. VanderWiel and David J. Lilja. A compiler-assisted data prefetch controller. Technical Report ARCTiC 99-05, University of Minnesota, May 1999.