Sharing Speculation : A Mechanism for Low-Latency Access to Falsely Shared Data

Rajagopalan Desikan* Jaehyuk Huh Doug Burger Stephen W. Keckler
*Department of Electrical and Computer Engineering
Computer Architecture and Technology Laboratory
Department of Computer Sciences
The University of Texas at Austin
cart@cs.utexas.edu - www.cs.utexas.edu/users/cart

Department of Computer Sciences
Tech Report TR-03-05
The University of Texas at Austin

May 12, 2003

Abstract

False sharing of data is an important phenomenon affecting performance in shared memory multiprocessors. False sharing results in unnecessary coherency overhead by causing invalidation of the shared cache line, and increasing the latency of the load accessing the cache line. As microprocessors incorporate increasingly large caches with large cache lines, false sharing will become more common, and unless methods are proposed to alleviate, can reduce the performance of shared memory multiprocessors. In this report, we propose sharing speculation, a mechanism to speculatively load values from cache lines that are falsely shared to reduce the latency of these loads, and later validating the speculation when the coherence permissions are eventually granted. We give a specific example of our proposed mechanism in the context of the Grid Processor, and show how the inherent data speculation mechanisms in the Grid Processor lend themselves to the efficient implementation of sharing speculation.

1 Introduction

The shared memory paradigm is the programming model predominantly used in chip multiprocessors. Shared memory results in a simpler programming model, and hence has emerged as the most popular choice for multiprocessor systems. Chip multiprocessors are an emerging class of systems that are expected to use this programming model. The availability of a large number of transistors in these systems also facilitates fabrication of large, high bandwidth on-chip caches with large cache lines. Large cache lines usually increase spatial locality, and hence the hit ratio, thus resulting in fewer bus transactions. However, they also result in the co-location of unrelated data in the same line that might be used by different processors. This phenomenon is called false sharing, because even though the cache line is shared between processors, the individual data elements are not actually shared by the processors [7]. False sharing results in unnecessary coherence traffic and longer latency to data in the shared line.

Researchers have proposed methods to reduce false sharing in multiprocessor systems [6, 5, 2, 9]. However, false sharing cannot be completely eliminated, as different programs have different sharing granularities, that cannot always match the cache line size. In this report, we propose *sharing speculation*, a mechanism to load falsely shared data from cache speculatively while initiating the coherence mechanism in parallel, in order to reduce the latency of access to falsely shared data. Using sharing speculation, a cache can speculate on shared data and and service a read request assuming it is falsely shared. When the false sharing is resolved, the cache can validate or invalidate the speculation depending on the outcome of the coherence operation. Depending on the cost to recover from data value mis-speculation, sharing speculation can result in substantial speedup for applications that exhibit significant false sharing. Using confidence predictors, sharing speculation can also be tuned for each application.

2 False Sharing and Sharing Speculation

The granularity of data storage in caches is a cache line. Cache lines are normally much larger than the size of data accessed by processors. This mismatch in size can result in data accessed by different processors to reside in the same cache line, resulting in what is called false sharing [7]. False sharing results in processors invalidating and loading entire cache lines, even though they modify only a part of the cache line.

False sharing can degrade performance in two ways. First, false sharing results in a large number of unnecessary coherence messages. Second, loads to falsely shared data are unnecessarily stalled till the cache line is in a valid state. A number of techniques have been proposed to reduce false sharing. In this report we propose sharing speculation, a mechanism to load falsely shared data from the cache speculatively while initiating the coherence mechanism in parallel. When the coherence operation completes, the data are checked to see if there is false sharing and an appropriate message is sent to the processor either validating or invalidating the original load. Sharing speculation, when successful, will mitigate coherence latencies but not reduce the number of coherence messages in the system.

Sharing speculation is a form of data value speculation, where the cache speculates on the value of a load to an invalid line. When the cache receives a load for which there is a valid tag match but the line is in the invalid state, it sends the data back to the processor but marks it as speculative. The cache also simultaneously initiates coherence operations to bring the line to one of the valid states. When the cache line is updated, the cache checks the new value of the loaded data with the previous value. If the value is the same (as will happen in the case of false sharing), the cache sends a message to the processor validating the speculation. If the data value changes after the coherence operation, the cache sends the correct non-speculative value to the processor. The processor then initiates recovery from incorrect data value speculation. A flow chart depicting sharing speculation is shown in Figure 1.

The benefits of sharing speculation will depend upon both the cost of data value mis-speculation recovery

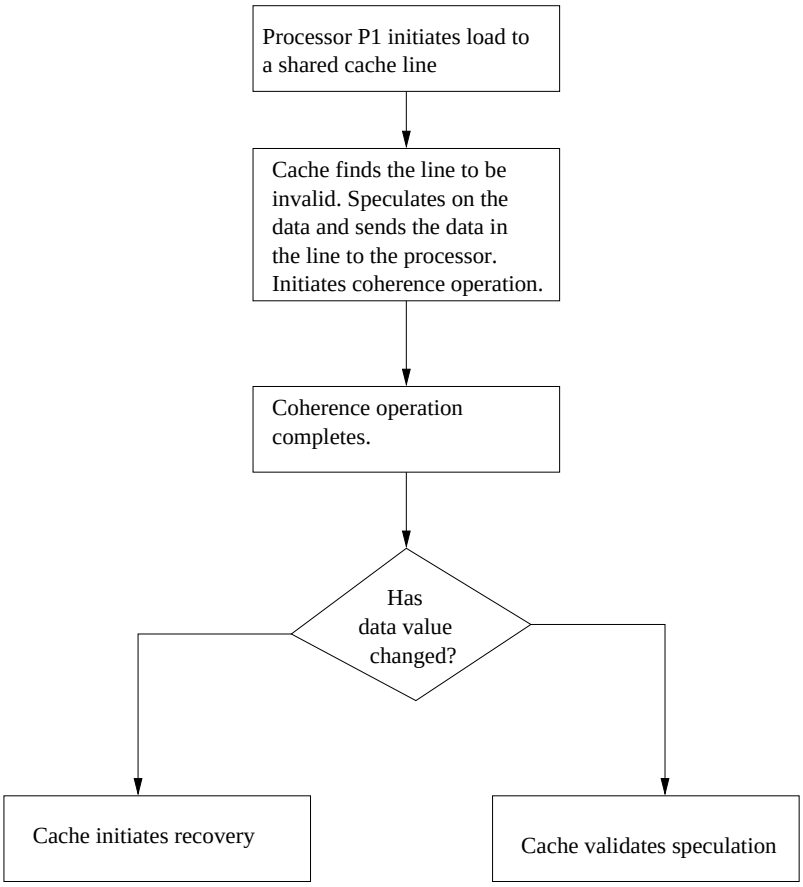

Figure 1: Flow chart depicting sharing speculation

and the amount of false sharing exhibited by an application. Using confidence predictors in each cache line, we can vary the amount of speculation during run time in order to obtain optimal performance for each application. Sharing speculation can also be greatly aided if the processor supports a light weight data speculation recovery mechanism. One such mechanism, that has been proposed for the Grid Processor Architectures, is selective re-execution. We describe this mechanism in the next section.

3 Sharing Speculation in the Grid Processor

Grid Processor Architectures (GPA) are a family of architectures that was designed to scale to high performance in future, wire dominated technologies [10]. In a GPA, instructions are statically scheduled by the compiler onto a two-dimensional array of ALUs but are dynamically issued and execute in dataflow fashion.

In a GPA, blocks of instructions are fetched as a single unit, mapped on the ALUs in the processor, and after execution, are retired in a single atomic operation. The block of instructions has a single entry point at the top, no internal loops, and possibly multiple exit points. The data within a block consists of block inputs, block temporaries, and block outputs. Within the grid processor, a mapped instruction fires when all of its input operands arrive at the node and forwards its result to the consumer(s) of the instruction. The compiler explicitly encodes the physical destinations of an instruction's result in the opcode of the instruction. Thus, operands are delivered point-to-point within the grid, rather than being broadcast to all ALUs.

The data flow execution in the GPA facilitates a low cost data value speculation mechanism called selective re-execution. The selective re-execution protocol enables simple, distributed selective re-execution, which is a light-weight mechanism for recovering from data value mis-speculation. The selective re-execution protocol in the GPA is proposed as a general mechanism for different types of data value speculation. Hence, it can be easily used for implementing sharing speculation.

The GPA uses a versioning system to support selective re-execution. The versioning system allows multiple waves of speculation to traverse the GPA concurrently. Each operand in the GPA has a version number and a commit bit associated with it. Multiple copies of an operand have different version numbers and it is guaranteed that the non-speculative value of an operand will have the highest version number. The non-speculative value of an operand also has the commit bit set. Instructions fire speculatively as soon as they receive all their input operands. If any of the input operands is speculative, then the instruction result is also speculative. When an instruction receives all the non-speculative values of its input, it forwards a non-speculative value of its result to all the consumers. The version numbers and the commit bits enable only version numbers and commit bits to be forwarded inside the grid, when the speculation is correct, instead of actual data values. Hence, to implement sharing speculation, the caches or the MSHRs [8] in the system would need logic to use the selective re-execution mechanism implemented in the GPA, to inject speculative values into the processor. It is worth noting that if mis-speculation recovery overhead is sufficiently low, as in the GPA, then it is always better to speculate, since waiting for speculation confirmation costs the same as waiting for the value.

4 Potential Benefit of Sharing Speculation

In this section, we use a multiprocessor simulator to evaluate the amount of false sharing present in six benchmarks taken from various scientific application suites. We simulated four shared-memory benchmarks, *barnes*, *mdcask*, *ocean*, and *sppm* and two MPI-based benchmarks, *lu* and *smg2k*. Our simulated chip multiprocessor is an extension to the SIMOS system [11]. The processor model in the simulator uses sim-outorder, an out-of-order simulator that is part of the simplescalar suite [1]. The system uses a MESI coherence protocol and runs AIX version 4.3.1. We used an 8 processor configuration, with 1 MB level-2 (L2) cache for each processor.

Figure 2 shows the breakdown of L2 cache misses across all the processors in the system for each benchmark. We classify the misses into two broad categories, misses due to invalid coherence state (coherence misses) and other misses, that includes cold, capacity, and conflict misses. Within the coherence misses category, we sub-classify the data into false coherence misses and true coherence misses. False coherence misses have the same data value before and after the coherence operation, and true coherence misses have different data value after the completion of the coherence operation. Note that higher fraction of false coherence misses results in a larger benefit due to sharing speculation.

We see from Figure 2 that for a number of benchmarks, a large fraction of coherence misses is false coherence misses. *barnes*, *lu*, and *smg2k* all have a significant fraction of false coherence misses and hence can have significant improvement in performance with sharing speculation. *ocean* has very few coherence misses and *mdcask* and *sppm* have a larger fraction of true coherence misses. These benchmarks may not benefit significantly from sharing speculation. However, using confidence predictors and selective re-execution, we can ensure that sharing speculation does not hurt performance for these benchmarks.