Summary

Compiler Optimization of Memory-Resident Value Communication between Speculative Threads.

Kiran Yellajosula

February 10, 2005

1  To be completed before class

What are the problems solved by this paper? (50 words)

TLS has various hardware schemes for improving value communication between threads and the performance penalty of re-executing a failed thread is large, thereby, requiring that speculation to be done only in case of infrequent data dependences. The authors studied the potential improvement that can be obtained on reducing failed speculation time and are targeting to reduce it by synchronizing memory value forwarding.

What are the approaches attempted by this paper? (50 words)

The authors profile the dependences and select the frequently occurring dependences that need to be synchronized. The authors use the compiler to insert instructions to synchronize and forward the memory address and the memory value. The authors also schedule the instructions similar to scalars. The authors also introduce a signal address buffer to ensure correctness of program execution.

What are the main conclusions of this paper? (50 words)

The value forwarding and synchronizing has to be done to dependences occurring in at least 5% of the loops to achieve significant performance improvement. The performance impact of dependence distances of distance-one is significant. The compiler approach can be used along with the hardware approach to build a hybrid scheme which gives the best of the two worlds.

2  To be completed after class

Did this paper address an important issue? Explain. (100 words)

This paper points out the performance gain that can be obtained by decreasing the number of failed cycles and proposes to use memory address synchronization and memory value forwarding between threads. The authors develop a profiling infrastructure comprising of uniquely labeling each memory instruction and identifying dependences between them. The instructions accessing the same address frequently are then synchronized and scheduled to reduce the critical path. The functions where the dependences occur are also cloned to reduce the overhead of executing extra code. This compiler approach can be extended easily to other architectures.

Are the proposed approaches valid? Describe its strength and weakness. (100 words)

The strengths of the approaches are as follows:

1. Improves the performance by decreasing the failure time
2. The technique is similar to the scalar instruction scheduling, hence the code can be reused and the synchronization is introduced automatically by the compiler.
3. The approach can be used along with the hardware synchronization and value forwarding to improve the performance further.
4. The approach can be employed with other architecture models supporting threads.

The weaknesses of the approaches are as follows:
1. Access hardware introduces design complexity which wasn't part of the original architecture.
2. New instructions to resume threads and save to buffer are to be implemented.
3. Time to profile data dependences occurring in more than 5% of the iterations is long.
4. Each group may be very large, causing us to synchronize lot of operations.

**Do the results support the conclusions? Explain. (100 words)**

The results show an improvement in performance by reducing the synchronization time. The simulation results also show that the input set used to profile data doesn't affect the performance.

The simulation results show that the compiler inserted technique has lesser normalized execution time than the hardware value prediction scheme.

The number of unsynchronized violating loads and the speedup indicates that the compiler synchronization technique is good for some benchmarks (go, jpeg, gzip_decomp, gap)) while the hardware technique is good for a few others (m88ksim, mcf, twolf) but the hybrid approach beats them both and gives the best performance.

**Describe the potential future works? (100 words)**

This work can be employed in various other architectures and tested like the multiscalar processors. The compiler can further try to schedule the operations using aggressive techniques like the scalar variables. There is a considerable amount of other cycles in the execution time where no instructions graduate. The TLS processors can be profiled to find out the cause of the stall and employ other techniques like helper threads to prefetch data into the shared cache.