Summary
A scalable approach to thread level speculation.

Venkatesan Packirisamy
February 2, 2005

1 To be completed before class

What are the problems solved by this paper? (50 words)
Chip manufacturers are building commercial multi-core processors. But they are primarily used to increase throughput. One reason we could not use the multi-core processors to increase performance is our inability to parallelize applications. So there should be a mechanism to parallelize seemingly serial code. Also the mechanism has to be scalable. The paper tries to develop a mechanism that can automatically parallelize programs and that can be applicable to any machine which can support multiple threads.

What are the approaches attempted by this paper? (50 words)
The paper proposes to use thread-level speculation (TLS) to parallelize programs that have possible (but infrequent) dependencies between threads. The mechanism proposed is built over the invalidation based cache-coherence protocol, which is used in multiprocessor architectures with shared memory. So this mechanism can be used in any architectures which has shared memory. The compiler partitions the programs into threads which are run on the different processing elements. The sequential semantics is maintained with the help of 'epoch' numbers assigned to the different threads.

What are the main conclusions of this paper? (50 words)
The proposed mechanism achieves 8 – 46% speedup over single processor execution. The overhead due to the hardware and software mechanisms are reasonably small. This shows that speedup can be achieved with very small amount of hardware by using TLS. Applications like buk and equake are almost insensitive to communication overhead showing that TLS can be very effective in multiprocessor configurations with very large communication overhead. Thus using the scalable approach proposed TLS can be effective on a large range of processor configurations.

2 To be completed after class

Did this paper address an important issue? Explain. (100 words)
Inspite of having lots of commercial processors which support multiple cores within the same processor, we could not use them to extract parallelism in a single program. At present they are mainly used to get throughput. The major issue is that we could not parallelize a sequential integer program. This paper addresses this main issue by proposing to use speculative threads to parallelize difficult to parallelize loops. Another issue in the most of current CMP designs proposed is that they rely on centralized hardware and typically use very complex hardware. Instead this paper aims to have a scalable design which will apply not only to multiple core processors but all processors that support multiple threads. Parallelization using speculative thread (TLS - thread level speculation) and scalable design are the two issues the paper tries to address.
Are the proposed approaches valid? Describe its strength and weakness. (100 words)
The compiler partitions the threads and if necessary it inserts explicit synchronizations. The threads are executed in parallel in the processing elements. The speculative store value buffering and hardware detection of dependence violation is done by overloading the invalidation based cache coherence protocol. Additional states are added to the possible cache states which indicate speculative state. Also special bits are added to each cache line to identify speculative load and store. New messages are also created which does dependence checking.

Strength:
The amount of hardware needed to implement this method is very less. Apart from adding few bits to the cache line and augmenting the functionality of the existing cache coherene protocol, there is no other hardware support needed.

Weakness:
The method is not absolutely scalable. Since invalidation based cache coherence protocol is limited, this method is also limited. Also for smaller number of processing elements (say 2), the cache coherence protocol is a huge overhead.

Do the results support the conclusions? Explain. (100 words)
The performance results for 4 benchmarks were presented. All the benchmarks show good improvement. Though in jpeg the results are bad because of conflict between cache accesses which was causing unnecessary squashing. On the whole the results show the potential for thread level speculation. The performance of the benchmarks is studied by increasing the communication latency. Some of the benchmarks show very less sensitivity to increased latency. Also these benchmarks also show very good results when studied under multiprocessor like configurations. This shows that the method is applicable to various configurations and inspite of huge communication latency, some benchmarks can show performance. So the design is scalable. Though, the results were studied only till 8 processing elements and the benchmarks are seemingly optimized for 4 thread performance.

Describe the potential future works? (100 words)
Only 4 benchmarks are studied. And the benchmarks themselves are not really scalable. To really understand the potential of a scalable design, we have to study under some really scalable benchmarks. The benchmarks which have a lot of potential for parallelism and at the same time they are not parallelisable using conventional compiler techniques and thus requiring thread level speculation. For example, we could take some good parallel programs and try to further parallelize them, say by augmenting openmp directives with some new directives which are specific to TLS. The technique could benifit from compiler optimizations - code scheduling and placing the synchronization points. Also we can try to design a method which is not limited to just shared memory systems.