Summary Part 1
Compiler Optimization of Memory-Resident Value Communication Between Speculative Threads
Mrinal Nath (ID: 3307043)
February 7, 2005

What are the problems solved by this paper?
Multithreaded processors are used to improve the throughput by running independent threads simultaneously. But in a single ‘normal’ program, there isn’t much explicit TLP, and so the multithreaded processor is under-utilized. Such ‘normal’ programs are also very difficult to parallelize (especially integer programs).

So far, hardware mechanisms have been used to track data dependences through memory resident values. If an epoch violates any dependence, then the hardware squashes and re-starts the epoch, which can be a large overhead, especially if the dependences that are being violated occur frequently.

This paper builds on previous work related to compiler optimization for scalar value communication between speculatively threads. The compiler is used to locate frequently occurring memory-resident data dependences and insert synchronization instructions. This can reduce the squash-restart overhead significantly. There is also the possibility that hardware complexity can be reduced by shifting some of the complexity to the compiler.

What are the approaches attempted by this paper?
First, profile information is used to help the compiler in identifying the frequently occurring memory-resident data dependences. These are the probable instructions that have data dependences. Then \texttt{wait} and \texttt{signal} instructions are used to explicit synchronize epochs. (‘Cloning’ of procedures is used to minimize the synchronization overhead.) The compiler applies the optimizations only to the loops that are very likely to improve the performance after optimization.

Some additional hardware support (some additional instructions to be supported and some flags and buffers) is needed to verify the execution, and to recover if a wrong value was forwarded by an earlier epoch to a later epoch.

What are the main conclusions of this paper?
It was seen that in almost all benchmarks, the input set used for profiling did \textit{not} matter. Also, the hardware used to avoid misspeculation by using hardware synchronization could not be eliminated completely by using the compiler techniques. This is because the compiler and the hardware synchronize different sets of memory-resident values. So, the hardware and the compiler can work in tandem to maximize the performance improvement.

To achieve sufficient speed up in all benchmarks, even dependences that occur only in around 5% of the epochs should be synchronized. Inter-epoch data dependency distance was found to be almost always 1, which can simplify the forwarding mechanism.