Summary Part 1

Helper Threads via Virtual Multithreading on an Experimental Itanium 2 Processor-based Platform

Mrinal Nath (ID: 3307043)
February 7, 2005

What are the problems solved by this paper?
Uniprocessors without SMT capability can stall for a few hundred cycles if there are long-latency microarchitectural events, like a memory reference going all the way to the main memory. These stall cycles can be used to do some useful work. We can not do a context switch in a few hundred cycles, so other threads or ‘normal’ helper threads (that can be used in a SMT processor) can’t be used in this stall time. This is because a context switch (requiring OS intervention) can take several thousand cycles.

This paper proposes a solution by which the stall cycles of the main thread can be used to do useful work by using a helper thread. The approach is targeted at uniprocessors which don’t have SMT capability.

What are the approaches attempted by this paper?
This paper introduces the VMT mechanism, by which a uniprocessor can take advantage of ‘helper’ threads when the main thread is stalled for a long time. A helper thread can be started and can do some useful work (like prefetching or branch prediction) for the main (stalled) thread. Unlike context switches, the VMT mechanism can suspend the stalled thread and start the subordinate thread within a few cycles (fly weight switching). This is possible by using compiler optimizations and judiciously partitioning the register file between the main and the helper thread, so that the ‘context switch’ can occur with minimal state saving – only the PC of the main thread has to be saved. The subordinate thread is totally transparent to the OS, and incurs no overhead for the main thread. Since the VMT helper threads are used only in stall cycles, the main and subordinate threads do not run simultaneously, and there are no resource conflicts among the threads.

The opcodes for the special VMT instructions are annotated NOP opcodes in the Itanium ISA. This is important as it allows backward-compatibility if VMT capability is not available on a particular processor platform. The PAL supported by Itanium is used to emulate the special instructions. This uses the extensive hardware debug and performance monitoring capabilities of the Itanium.

What are the main conclusions of this paper?
Workloads that are known to suffer from long-latency cache misses used to evaluate the performance of the VMT approach. Significant performance improvement was achieved. Compiler optimizations for register partitioning and synchronization between the main and the helper threads is important to minimize the thread switching overhead. Even a few fetches are done using a couple of helper threads, the performance benefit can be quite noticeable. If the co-routine execution mode is used, then the helper thread can stay ahead of the main thread and help the main thread, increasing the performance. Importantly, even though VMT was mainly targeted at improving the performance (not throughput) of a single threaded application, it can improve the throughput of highly threaded workloads, like database applications.