Saving Power through Explicit Mechanisms

Dave Maze, Edwin Olson
6.893, Fall 2000

Mainstream computer architectures, including the DEC Alpha family and the Intel IA32 and IA64 architectures, have historically targeted high performance. Performance has been enhanced through mechanisms such as out-of-order execution and multiple execution units.

These advancements, however, have come at a significant cost of power. For many applications, especially those in the embedded arena, have stringent power requirements. Unfortunately, they can also have very high peak performance requirements. Embedded processors can be decompressing a complex MPEG4 video stream one moment, and sitting idle the next. While typical desktop microprocessors consume vastly different amounts of power while busy and idle, they can not approach the low levels of power dissipation required of embedded processors when idle.

The ideal embedded microprocessor can operate at a large number of performance levels, consuming as little power as possible. Various techniques exist to throttle cycle time and operating voltage, as well as putting functional units to sleep. We would like to extend these ideas to further reduce power. For example, we could turn off several ways of a multi-way associative cache when the working set is small, or disable out-of-order execution logic (and the issue queues, extra registers, etc, that usually accompany it). Other ideas include skipping TLB/cache accesses when the compiler can statically determine whether a hit would occur.

We also propose to do as much of this as possible at compile time; the last thing that a low-power microprocessor needs is additional circuitry to perform run-time profiling. It is not possible to perform all such optimizations at compile time. For example, it would be very difficult to determine the working set for code that can operate on variously sized buffers. A carefully chosen set of performance counters could allow the compiler to emit code to conditionally change the operating configuration based on the values of the performance counters.

We will begin our work by narrowing down our list of candidate approaches to the most promising and novel by continuing to search for relevant literature and profiling code to determine the amount of improvement that could be realized. Preliminary estimates should be available by our first checkpoint, October 19.

A process of refinement begins after the first checkpoint, as we extend our ideas, build automated tools for compiling code (perhaps a SUIF optimization pass). We anticipate high-quality results by our second checkpoint, November 9th.

Our tools will include a MIPS ISA simulator for profiling code and analyzing different optimzation methods. SyCHOSys might be used to help measure energy numbers. We may use the SUIF compiler suite to optimize code for low-power.