CMPSCI 630 Systems **Spring 2017**

Lecture 7

Lecturer: Emery Berger *Scribe: Abram Handler,Daniel Sam Pete Thiyagu*

RISC v. CISC (continued)

One of the main arguments for RISC was that RISC is easy for compiler writers and easy for computer architect designers. However, there are very few compiler writers and designers and many billions of users. We can alleviate the problem of difficult code writing for compiler writers or difficult chip design by simply paying more money for skilled workers. Ultimately, the needs of users are much, much more important.

Moreover, the assumptions behind the RISC vs. CISC debate have changed. Currently we have two architectures - INTEL and ARM. Prior to that we had a proliferation of architectures like Motorola chips, MIPS and many more. So some of the worry about supporting varying, heterogeneous CISC is less relevant today.

Some assorted notes:

- Intel embedded a compiler onto their chip, which translates CISC to RISC. So recent Intel processors are "CISC outside, RISC inside".
- The x86 architecture is compatible with a many applications.
- Itanium (sometimes called "itanic") was a failure at Intel which utilized RISC. It had no speculation and access to cache was unpredictable. It used Explicitly Parallel Instruction Computing (EPIC) and had giant instructions of same size. These giant instructions are called bundle. A bundle consists of 3 instructions. All code in same bundle can be run independently in parallel. It can also be fetched together in a batch. This is very similar to VLIW, which was also developed to alleviate the problem of non-parallelism, but which also did not turn out to be a success.
- For increased parallelism, chips try to check for dependencies and, if none are found, they run instructions in parallel. Tomasulo's algorithm is an algorithm that helps detects dependencies. Modern chips also employ speculation (next section).

Branch prediction

Branch prediction tries to decide what branch should in a code will be taken so things can be computed ahead of time. However, chip designers need to keep track of speculative operations with a speculative bit, because some of the speculative calculations could be wrong or not necessary.

Speculation is expensive (i.e. power or clock cycles), so incorrect speculation or misses should be really low. Current branch predictors now have an accuracy of around 99%. Daniel Jimenez found a way to approximate the classic perceptron algorithm in one clock cycle. Any algorithm that does prediction needs to be very fast.

Static Branch Predictors Static predictors decide what to predict without any dynamic, run time information from the code. Examples might be code that always accepts or always declines to take a conditional branch.

Dynamic Branch Predictors Another option is to use dynamic prediction, such as keeping a count of how many times a particular branch was taken. The number of bits stored for the count should be low. Sometimes the bit counts are "saturating counters".

Moore's law, power and multi-core

Moore's law was in effect until about 2005. But around this time, chip makers started having power problems. Adding more transistors was not the problem, but power was. Power and heat grows quadratically as gates are added to a chip. But gates grow linearly in log space. So if we keep adding chips at the current rate, at a certain point our computers will be as hot as the sun, which is a problem for mobile devices.

The distance between gates is "feature size." People have speculated about limits to feature size, but then these limits are broken. As density of gates increases, the distance between them decreases and they can send quicker signals to each other. Dennard scaling concerns the performance per unit of power as density increases.

Chips need to be cool. So one idea for multicore chips: if there are hotspots, move the processing to another core or another part of the chip. One problem with this is, what happends when processing uses all cores?

Multicore machines work better in some settings (webserver) than others (mobile devices for personal use).

7.1 Notes about parallel processors

Vector Processor: executes a single operation on lots of data (vectors). This is called "SIMD", Single Instruction Multiple Data. Vectorizing operations makes them much faster than executing the same operation many times, component by component. If you do multiple instructions in parallel on lots of data it is called "MIMD", Multiple Instruction Multiple Data.

Other notes: How do you expose parallelism? Dynamically discover dependencies and use compiler independent optimizations.

Mesa Monitors

A monitor is a procedure which handles concurrent access to shared data. The Mesa language went away. But Mesa monitors were ported into Java: in Java, each object has a monitor with a recursive lock which can be acquired multiple times.

Other notes:

1. Java objects each have a counter, a thread id, and a condition variable.
2. Sometimes, the object header is often bigger than the object itself
3. You don't necessarily need all of this information for each object. So David Bacon introduced "thin locks," which are basically just bits. Each time you use the object, you get the bit (lock). This optimization, called a "Bacon bit", saves space.