

## Re: Bad performance on alpha? (make buildworld)

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/performance/2004-02/0058.html>

---

**From:** Chuck Swiger (cswiger\_at\_mac.com)

**Date:** 02/25/04

Date: Wed, 25 Feb 2004 00:19:15 -0500

Peter Jeremy wrote:

> On 2004-Feb-24 20:17:07 -0500, Charles Swiger <cswiger@mac.com> wrote:  
[ ...compiler optimizations... ]  
>> I'm afraid you've got this backwards. :-)  
>  
> Maybe in theory, but not necessarily in practice.

It's been a few years since I'd written a compiler, but my viewpoint isn't based entirely on theory.

>> The primary attributes of RISC architectures, namely lots of registers,  
>> a relatively simple but orthogonal instruction set, and a relatively  
>> fast clock rate / CPI  $\approx$  1.0 / a short pipeline make it far easier for  
>> the compiler to generate and optimize code.  
>  
> Alpha pipelines are only short in a relative sense – the EV5 pipeline  
> is 7 (integer) or 9 (FP) stages and I suspect the EV56 pipeline is the  
> same. In theory, it is 4-way superscalar but the different execution  
> units aren't equivalent and the compiler has to understand which  
> instructions will be allocated to which execution units in order to  
> minimise stalls.

A Northwood P4 has 20 stages (or 21, or 28, depending on how you want to count instruction decode stages rather than just the integer pipeline), the P3 has 12, and even the P2 has 8. Prescott has what, 32 stages?

Any superscalar processor architecture is going to be harder to compile for than an architecture which is not, just as parallel execution with limited execution units requires more work than a truly orthogonal architecture.

>> CISC architectures make the compilers job much harder because they tend  
>> to require lots of register spills, they tend to have very long  
>> pipelines which involve hazards and require a lot of instruction  
>> reordering to avoid stalling the pipeline to often. The amount of CPU  
>> clocks it takes per instruction (CPI) often varies on CISC as is  
>> generally much larger than  $\sim$ 1.0, and sometimes varies from CPU model to  
>> CPU model making it far more difficult to determine the "fastest"  
>> instruction sequence.

frebsd-performance: Re: Bad performance on alpha? (make buildworld)

- >
- > *Recent iA32 implementations (basically anything more recent than a*
- > *P11) are RISC cores which directly execute a subset of the iA32*
- > *instruction set with the remainder handled by microcode. You get*
- > *quite respectable results by treating it as a load/store RISC*
- > *architecture and relying on the L1 cache to handle the register spills*
- > *in a timely fashion. The pipelines and super-scalar execution*
- > *abilities are all handled in hardware. Register scoreboarding allows*
- > *the implementation to have more physical registers than the programmer*
- > *view supports – allowing multiple instructions to simultaneously see*
- > *different values in the same visible register.*

Your technical description is accurate, but the points you are making here seem to support my argument, rather than contradict what I said. :-)

Basicly, you've suggested that it's easier to compile for a recent x86 than for a P2 because the hardware in a P4 goes to extravagant lengths to dynamically optimize x86 instructions (CISC) into simpler RISC instructions which can be scheduled, executed out-of-order, using a ~120 register scoreboard to multiplex 8 visible registers amongst all of the pipeline stages, etc, etc.

Why does the P4 actually execute RISC u-ops and microcode, rather than implementing the x86 opcodes directly? Because it's far easier to optimize RISC u-ops, whether in the context of a compiler or in the context of the CPU hardware itself, than to try to optimize CISC opcodes directly.

- > *The compiler has to expend a lot of effort on instruction scheduling*
- > *to get decent performance out of a typical RISC architecture. Much of*
- > *this is automatically handled by the hardware on an iA32 and you can*
- > *get equivalent results with a much simpler compiler.*

Is it easier or harder to optimize generic x86 code for the P2 or for a P4, and why?

If you don't optimize intermediate code at all, which performs better? Why?

Depending on your L1 cache to reduce the costs of spilling registers all over the place because you've only got about 6 or so freely available is a mediocre bandaid compared to having 32 or so registers: unoptimized PowerPC, MIPS, PA-RISC, or SPARC code does a heck of a lot better than running unoptimized x86 code, and that's without considering something like the SPARC register windows which do a heck of a job of passing context between caller and callee and freeing up 8 new registers for temp use at each function invocation (and without the mind-numbing complexity of HP's PA-RISC calling conventions).

--  
-Chuck

---

frebsd-performance@frebsd.org mailing list  
<http://lists.frebsd.org/mailman/listinfo/frebsd-performance>  
To unsubscribe, send any mail to "frebsd-performance-unsubscribe@frebsd.org"