

Some Thoughts on GCC-on-Itanium

January 16, 2005

Here I present a few project ideas for improving GCC on IA64 which I think are realistic given the relative scarcity of open-source IA64 compiler developers with the time to do the sorts of things I'm suggesting. Each of these projects would be worth roughly a few percentage points on something like SPECcpu2000. If all of these were implemented, pretty much everything in a typical Linux environment would get something like a 10% speed gain, on average. (Perhaps more for FP-intensive codes.) Later, I mention some longer-term projects that might be appropriate, but are somewhat more demanding.

- **Relatively Easy Jobs**

- 1) **Implement proper bundle selection.**

The last time I checked, GCC chose how to pack IA64 instructions into bundles using a very poor, heuristic method. The basic problem is that the chip state is not tracked as instruction bundling is done, which results in many empty or near-empty bundles that waste cycles and instruction cache, both of which are scarce on IA64 chips (i.e. low clock speed and small L1 caches.) The new L2I of the Montecito chip will alleviate this problem to *some* extent but significant gains will be seen both on Itanium 2 and future systems. The “right way” to do this is to integrate scheduling and bundling using a system like that described in *Efficient Modeling of Itanium Architecture during Instruction Scheduling using Extended Finite State Automata*, Chen, Liu et al., Journal of Instruction-Level Parallelism vol. 6 (2004) pp1-26.

- 2) **Support data speculation.**

Loading data from memory takes time, and many common codes spend most of their time doing it. For 99.99% of machines these days it's not as big a problem as it might be, because the CPU you use has special hardware to automatically execute other instructions while waiting for the load to complete. Almost every CPU manufactured today has this capability (dynamic out-of-order execution) but Itanium chips don't, and that was a design decision. (One I happen to agree with). Currently, when Itanium code generated by GCC emits a load instruction, the CPU typically sits there doing almost nothing for a while because GCC couldn't “hoist” the load up high enough. Itanium provides special “advanced” (early) load instructions 'ld.a' and 'chk.a' which allow loads to be performed early enough (so early that such loads would be considered dangerous on other architectures) that a meaningful amount of other useful work can be found and performed while waiting for the load to complete. At this point, it's worth noting the following e-mail from Jim Wilson, April 2004:

```
Subject: Re: user-guided speculative precomputation? (my wacky ia64 idea)
From: Jim Wilson <wilson@specifixinc.com>
To: Duraid Madina <duraid@octopus.com.au>
Cc: gcc@gcc.gnu.org
Date: 07 Apr 2004 17:39:15 -0700
Message-Id: <1081384756.1050.79.camel@leaf.tuliptree.org>
```

On Wed, 2004-04-07 at 17:09, Duraïd Madina wrote:

> On that note, does GCC *ever* emit ld8.a / chk.a pairs? This would be a
> very nice addition for yucky pointer code, and isn't the can of worms
> for the compiler that ld.s/chk.s is.

No.

Cygnus did try to implement some advanced/speculative load support about 4-5 years ago, but we couldn't get a reliable performance increase. I think part of the problem was that the alias info in the RTL wasn't quite good enough, and as a result we ended up generating too many advanced loads which hurt performance. It would be worthwhile trying again some day, as there have been infrastructure improvements since then. Or maybe we didn't find the right set of heuristics to use. Unfortunately, there isn't anyone doing serious IA-64 gcc work at this time.

--

Jim Wilson, GNU Tools Support, <http://www.SpecifixInc.com>

..or in other words, they tried this a long time ago but failed due to problems in other areas of GCC that have since been addressed.

3) **Support predicated execution in a quick-and-dirty-hack software pipeliner**

IA64 supports predicated execution but it is well known that the performance gains due to if-conversion are not generally very large without *substantial* compiler and run-time (profile) support which GCC is hardly in a position to provide. However, predicates greatly enhance the efficiency with which software pipelines can operate. As GCC currently knows basically nothing about predicated execution, if one wanted to support predication properly, he would have to go through many GCC passes (many optimizations, register allocation etc) and make them predicate-aware. This is difficult, time consuming-work and probably isn't worth it. However, and in the conspicuous absence of decent GCC support for cross-platform modulo scheduling, it *may* be worth getting an IA64 software pipeliner up and running in a slash-and-burn sort of manner, and having *that* be the *only* part of GCC which uses (rotating) predicate registers. This work would be a big win for those unfortunate souls doing FP-intensive work with GCC on Itanium (they're out there!).

• More Difficult Work

1) **Polyhedral Loop Transformations**

There is a French group that has extended Open64 (SGI's open Itanium compiler) to perform polyhedral loop transformations. This is basically loop nest optimization (fusion/fission, unrolling, cache tiling/strip-mining, interchange etc.) on steroids, and is a big win for every architecture but a *huge* win on IA64 due to its big caches and unmatched support for software pipelines (rotating registers, predicates, counted loop support).

It would be work to get this into GCC (and even more work to do so 'nicely'), but it could be done. Only worth doing if decent software pipeline support is present.

See e.g. http://www.lri.fr/~girbal/site_wrapit/publications/rr-lcpc.ps.gz

For more info, the homepage is at: http://www.lri.fr/~girbal/site_wrapit/

2) **Proper support for predicated execution**

The title says it all; one of the most obvious features of IA64 is the support for predicated execution, but this is completely ignored by GCC. However, since the early IA64 days, it has become clear that the performance gains from simple optimizations taking advantage of predicates are not always significant, and often depend on substantial compiler infrastruc-

ture being present, such as interprocedural analysis (deferred optimization.) The lack of such infrastructure is one of GCC's most prominent deficiencies these days, but it is getting some attention. It *may* be worthwhile adding proper support for predicated execution to GCC, in the hope of taking advantage of later developments such as interprocedural analysis.

Importantly, however, this work would be as timeconsuming as, and made almost completely redundant by an Itanium backend to LLVM (see 2) below) if one considers LLVM as 'GCC without a backend', so one would want to embark on this project with a little caution.

· Long-Term Projects

1) **Create a WHIRL/GCC mutant**

Work on getting a WHIRL "back-end" *integrated* into GCC. This would not be an incredible amount of work, and would allow the GCC optimizers/codegen to be completely replaced by alternative compiler back-ends such as IPF-ORC.

The advantage of doing this would be that developers of Itanium compilers such as IPF-ORC would no longer have to track GCC development: so long as the WHIRL backend was maintained, optimizing Itanium back-ends could be developed more or less independently.

2) **Create an LLVM/GCC mutant**

LLVM is a unique virtual machine: it supports almost any imaginable language, features the current version of GCC's C/C++ front-end with which it can apply a number of optimizations GCC can't (including link- and run-time optimizations), and as a static compiler often outperforms not only GCC but vendor compilers such as the Intel x86 C/C++ compiler and the IBM XL C/C++ on PowerPC. LLVM really is that great. More than any other architecture, LLVM offers Itanium some *very* important optimization opportunities: one main example being the chance to dynamically form meaningful hyperblocks and determine branch predictability, so that appropriate if-conversion tradeoffs can be made and predicated execution can be leveraged as originally hoped, but not yet achieved..

Unlike WHIRL, however, there are no Itanium back-ends for LLVM. I have a desire to see this happen and have submitted a grant to the Gelato SGP for assistance in getting this done; however there are *many* things to be done and it will take years to do them all: the more help the better, and to that end there are some opportunities for collaboration: The ORC guys could consider porting their back-end to LLVM, or writing an LLVM to WHIRL translator. The OpenIMPACT guys have great ideas but very messy code (it's a shame, but it's true) and I sincerely believe that it would *help* them to write an OpenIMPACT v2.0 from scratch, using the wealth of experience and ideas they have gathered and LLVM as a base.

3) **Study interaction between future Itanium processors and GCC as it exists today**

It's hard to imagine that any of the work mentioned above *wouldn't* be of benefit to future Itanium processors, but the relative importance of each project will depend, perhaps greatly, on the microarchitecture of future Itanium processors. For example, if future Itanium processors will feature speculative precomputation (see e.g. *Speculative Precomputation: Long-range Prefetching of Delinquent Loads* by Collins et al., Proc. 28th Intl. Symp. Comp. Arch., July 2001) then one should spend more time on 'proper bundle selection', and (much) less time on 'support data speculation.' Similarly, if future Itanium processors will feature enlarged L1 caches, then one shouldn't worry quite as much about proper bundle selection, etc.

Basically, detailed knowledge of *future* Itanium microarchitectures should be given to open-

source compiler developers so that they can make appropriate choices on where to spend their **very** limited development time. It is my honest opinion that the single worst thing Intel, Gelato, or anyone could do to steer open-source developers *away* from Itanium is to keep quiet (or stick to vague marketing-speak) about future Itanium processors, for two reasons: Firstly, without a roadmap, it's difficult to be motivated to do development work (the current attitude in the GCC world is "why should we kill ourselves developing software to make a chip we can't even afford simply *match* the performance of AMD's stuff that we *can* afford and that greatly outperforms it *right now*?"). Secondly, developers should have enough information to make appropriate choices about development direction, otherwise you run the very, very great risk of alienating developers by having them work on particular optimizations for a year or two, sacrificing their own personal time to do so, only to have their work made completely redundant by Intel developments that they could have been informed of. Get people to sign NDAs if they must, but it's not critical product details or performance estimates that they need to have, only microarchitectural and product-level directions: GCC developers are generally smart enough; they can fill in the blanks themselves!

• End Notes

An IBM compiler developer in the U.S. recently commented to me that "Itanium should have stayed in research for a few more years." While this is probably true, I wonder if the real problem is that people in the Itanium project didn't fully engage with the research and open-source communities a few years earlier than they did, while Itanium 1 was still in development. Intel don't need to be taught how to design revolutionary microprocessors, but they could have learned some valuable lessons on how (not) to sell them.

As a final, general comment I'd like to say that HP discontinuing the zx2000 and zx6000 workstations seems to be the worst thing to have happened to Itanium adoption in the open-source community recently, since it has been interpreted by many as proof that "Itanium is more or less cancelled." For better or for worse, open-source development is easiest on workstations, or at least servers that are cool, quiet and sit under developers' desks, not hundreds or thousands of miles away in corporate machine rooms. If you want open-source developers to work on Itanium, you have to convince them that it will be *relevant* to them in the future, if not today. **To a software developer, workstations are relevant!** The good news is that a workstation is easy to make. You take a server, stick a small, reliable video card in it, and install quiet fans. This is neither expensive nor difficult; *someone* should make it happen. I have sent a few notes to Al Stone at HP about some simple experiments I did in this direction.

If anyone has any questions or comments, I'd be very happy to receive them! My e-mail address is duraid@kinoko.c.u-tokyo.ac.jp.

Hope this helps,

Duraid