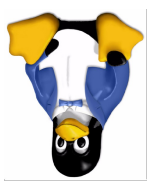
Tricks of a Spec Master



Michael Meissner
meissner@linux.vnet.ibm.com

Tricks of a Spec Ma

- Spec Benchmarks
  - Reciprocal estimate
  - Pow (x, 0.75)
  - Vectorized libraries
  - Fused floating point multiply and add instructions
  - Benchmarks that are vectorized by GCC on Powerpc
  - Benchmarks with hotspot functions
  - Running in 32-bit vs. 64-bit modes

Spec benchmarks

- Spec is an industry consortium that provides various benchmarks to compare computers. In terms of GCC, I tend to only look at Spec 2006 CPU which has two sub-categories:
  - SpecINT: Integer benchmarks
  - SpecFP: Floating point benchmarks
- There are strict rules about reporting results. From a compiler point of view, I tend to focus on using the benchmark as one are to optimize GCC, and not on official results.
- When I came to IBM from AMD, I joked that I was working for a new company, with a new processor, but I was still working on the same old benchmarks.

SpecINT benchmark

400.perlbench	401.bzip	403.gcc
429.mcf	445.gobmk	456.hmmer
458.sjeng	462.libquantum	464.h264ref
471.omnetpp	473.astar	483.xalancbmk

SpecFP benchmarks

410.bwaves	416.gamess	433.milc
434.zeusmp	435.gromacs	436.cactusADM
437.leslie3d	444. namd	447.dealII
450.soplex	453.povray	454.calculix
459.GemsFDTD	465.tonto	470.lbm
481.wrf	482.sphinx3	

Base and Peak runs

- Base runs are meant to run the entire benchmark suite with a common set of options. Profiling is not allowed for base runs.
- Peak runs are no holds barred runs. Profiling and inter-module runs are allow. You can specify different options for each benchmark.
- For tuning GCC, I tend towards base runs, with some changes:
  - I use -fno-strict-aliases on 401.perlbench and 433.milc.
  - I do use -ffast-math for runs
  - For GCC 4.3 I don't use -ftree-loop-linear on 464.h264ref due to a compiler bug.

Current PowerPC o

- -O3
- -fpeel-loops
- -funroll-loops
- -ftree-loop-linear
- -fvect-cost-model
- -ffast-math
- -falign-functions=16
- -falign-loops=32
- -mveclibabi=mass
- -mcpu=power7
- -mrecip=rsqrt

Reciprocal estimate

- Divide and square root instructions tend to be costly operations
- Some machines (x86, powerpc) provide estimate instructions:
  - 1/x estimate (FRE, FRES on powerpc)
  - 1/sqrt(x) estimate (FRSQRTE, FRSQRTES on powerpc)
- After the estimate, you use a few rounds of Newton-Raphson steps to improve the accuracy (2 steps on power6/power7 to get within 2 bits of the answer)
- 435.gromacs has 9 1/sqrt(x) scalar operations in the inner loop, and using FRSQRTE improves performance by 25%
- 437.leslie3d improves slightly for FRE, but 450.soplex slows down, so I only use -mrecip=rsqrt.

Pow (x, 0.75)

- Looking at the oprofile output, I noticed 410.bwaves was spending 50% of the time in the pow function. In looking at the benchmark, I saw it only used x**2 (which the compiler optimized) and x**0.75 (which was not optimized)
- I added machine independent code to GCC to optimize x**0.75 to sqrt(sqrt(x)) * sqrt(x) under -ffast-math.
- I also added x**0.25 to be sqrt(sqrt(x)) and x**(1./6.) to be sqrt(cbrt(x)). GCC was already optimizing x**2 and x**(1./3.).
- 410.bwaves sped up by 48%.
- It would be useful to move this operand to gimple before vectorization rather than in rtl expansion.

Vectorized libraries

- Starting with GCC 4.3, GCC got the ability to vectorize calls to builtin functions.
- X86 port supports two optimized math libraries:
  - -mveclibabi=acml (AMD's optimized math library)
  - -mveclibabi=svml (Intel's optimized math library)
- In GCC 4.6, I added support for the IBM MASS library with -mveclibabi=mass.
- 459.GemsFDTD sped up by 8%
- 465.tonto sped up by 75% (2% due to vectorizing calls)
- 481.wrf sped up by 72% (6% due to vectorizing calls)
- It would be nice to have a pass that can optimize the whole vector instead of just V4SF, V2DF, etc.

Fused floating point

- PowerPC has had fused multiply/add (FMA) instructions for sometime
- Other machines now have FMA's either in current machines or future machines
- Most machines do not do a rounding step between the multiply and add (ARM has both).
- Until last week GCC could de-optimize (a*b)+c and (a*b)-c
- A lot of benchmarks generate FMA's, but the one that really is dependent on FMA is 454.calculix, which speeds up by 16% with FMAs.

Benchmarks that ar

- On PowerPC power7 most of the 29 spec benchmarks generate some vectorized code.
- However, most of the vectorized code does not occur in the hot functions, so it doesn't contribute to performance improvements.
- When I measured it some time ago, there were only 3 benchmarks affected by vectorization:
  - 410.bwaves, -10% loss (may be code alignment)
  - 436.cactusADM, 60% improvement
  - 437.leslie3d, 28% improvement

Code alignment issues

- Code alignment is a factor in a few benchmarks that have most of the code in one function.
- Minor changes to the compiler or library may change where a loop is located in memory and cause a benchmark to be much slower or faster that has nothing to do with the changes you are making.
- Just blindly bumping up loop alignments can slow down other benchmarks due to the i-cache being less effective.
  - 410.bwaves has -/+ 20% variability
  - 450.soplex has -/+ 6% variability
  - 456.hmmer has -/+ 12% variability

Shrinkwrapping

- Shrinkwrapping is a function that has an early exit test that often fails
- The rest of the function is complex and needs to create a stack frame to spill variables to and save caller saved registers
- Void foo(bool test, /*...*/) { if (test) return; /*...*/ }
- 453.povray is the benchmark we noticed this on in the pov::Ray_In_Bound function, which accounts for 6% of the total time, and if we hand code the test before doing the frame save/restore, it saves 3% on the benchmark time.

Benchmarks with hot

- Some benchmarks are fairly flat in that execution is spread out over lots of functions (like 403.gcc for instance)
- Others spend a lot of time in a few functions, which are easier to optimize

470.lbm	99% (1 function)	436.cactusADM	99% (1 function)
456.hmmer	96% (1 function)	462.libquantum	94% (3 functions)
437.leslie3d	93% (7 functions)	459.GemsFDTD	90% (5 functions)
410.bwaves	89% (2 functions)	401.bzip2	86% (5 functions)
473.astar	85% (3 functions)	429.mcf	85% (3 functions)
444.namd	73% (7 functions)	435.gromacs	71% (2 functions)

32-bit vs. 64-bit ben

- On PowerPC, I see roughly 10% drop in SpecINT for running in 64-bit mode, and 1% drop for SpecFP
- However, some benchmarks slow down a lot and some speed up.
- 32-bit code has smaller stack frames and pointer sizes, which means the data cache is much more effective
- 32-bit code has a different ABI than 64-bit that has pluses and minues
- On x86, 64-bit instructions tend to be larger due to the instruction encoding
- Some benchmarks are faster due to use of long long in C or Fortran integers

Benchmarks that ar

429.mcf	31%	471.omnetpp	23%	464.h264ref	18%
483.xalancbmk	12%	473.astar	10%	445.gobmk	10%
401.bzip2	9%	416.gamess	9%	450.soplex	9%
458.sjeng	9%	482.sphinx3	6%	444.namd	5%
447.dealII	5%	403.gcc	5%	400.perlbench	4%
465.tonto	4%				

Benchmarks that ar

462.libquantum	4%	410.bwaves	14%
481.wrf	13%	436.cactusADM	9%
453.povray	4%		

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM.
- IBM is a registered trademark of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Other company, product, and service names may be trademarks or service marks of others.