Compilers Love Messing With Benchmarks

02 May 2014

If you want accurate and trustworthy benchmarks, you need to perform active benchmarking, as everything, **including compilers**, can mess with your benchmark. In this post I'll describe a common example.

There seems to be a trend, especially in cloud computing, for evaluating server instances using kitchen-sink benchmarks. These drag together old and new benchmark tools and automates their execution. UnixBench is a common inclusion – the original kitchen-sink micro-benchmark from 1984 – which is nowadays trivial to run:

```
wget http://byte-unixbench.googlecode.com/files/UnixBench5.1.3.tgz
tar xvf UnixBench5.1.3.tar.gz
cd UnixBench5.1.3
./Run
```

The "./Run" command will compile and launch UnixBench using its default Makefile, which includes:

```
## Very generic
#OPTON = -O

## For Linux 486/Pentium, GCC 2.7.x and 2.8.x
#OPTON = -O2 -fomit-frame-pointer -fforce-addr -fforce-mem -ffast-math \
#    -m486 -malign-loops=2 -malign-jumps=2 -malign-functions=2

## For Linux, GCC previous to 2.7.0
#OPTON = -O2 -fomit-frame-pointer -fforce-addr -fforce-mem -ffast-math -m486

#OPTON = -O2 -fomit-frame-pointer -fforce-addr -fforce-mem -ffast-math \
#    -m386 -malign-loops=1 -malign-jumps=1 -malign-functions=1

## For Solaris 2, or general-purpose GCC 2.7.x
OPTON = -O2 -fomit-frame-pointer -fforce-addr -ffast-math -Wall

## For Digital Unix v4.x, with DEC cc v5.x
#OPTON = -O4
#CFLAGS = -DTIME -std1 -verbose -w0
```

I didn't edit this. That's what you get in version 5.1.3, the version everyone is using. Take a closer look.

Makefiles

The UnixBench Makefile sets OPTON: a list of compiler options. (I'm happier if I assume that OPTON is short for Option-ON, and isn't an egregious typo.)

The default OPTON is for Solaris 2 (the last release of which was 1997). How many users don't realize this and let UnixBench use these defaults, vs, how many choose "Very generic" or one of the Linux options?

The problem is that people may end up using different compiler options, which can massively change the benchmark results. If you try and compare your UnixBench results with others found online, without realizing that these were compiled differently, you may be seriously misled. That is, if this even makes a difference...

Optimizations

Consider the following two Linux servers, and the first UnixBench result, Dhrystone 2 (more is better):

> server A: 21775273
> server B: 35773059

Server B is 64% faster, I bet you'd rather buy that! ... Except that these are the *same server*: B is compiled with the default OPTON (Solaris 2) and A is "Very generic" (a lower optimization level).

Fortunately this Dhrystone 2 result was the largest difference from the 24 benchmarks from UnixBench, and the difference to the final UnixBench "Index Score" summary was less than 5% (at least, in my case, this could be higher for you). Unfortunately, Dhrystone 2 is listed first in the UnixBench report, so it can be eye-catching.

What's happened is that UnixBench has benchmarked GCC (specifically, compiler options), as well as the target of the study. Mistakes like this are very common.

Perhaps, for UnixBench, it's not so bad. Perhaps everyone uses "./Run" and compiles with the Solaris 2 OPTON, so that even the Dhrystone 2 results can be compared accurately.

Versions

Now consider the following two servers, this time compiled with the same default OPTON (Solaris 2):

> server A: 24932516
> server B: 35773059

So I bet you'd like to buy... ok, yes, they are the same server again. The difference this time is the GCC version: Server A is gcc 4.1.2, and B is gcc 4.6.3. I suppose I owe a GCC developer a beer or some other gesture of

thanks, at least for when they aren't obfuscating my assembly.

Wouldn't everyone generally be running the same GCC version? Absolutely not. Right now I happen to be testing the performance of different server instances on AWS EC2, and a quick check shows I have:

```
gcc (GCC) 4.8.2 20131212 (Red Hat 4.8.2-7)
gcc (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3
gcc (GCC) 4.1.2 20080704 (Red Hat 4.1.2-54)
```

So if I didn't pay attention to the compiler, any results comparing systems would be bogus and misleading. Short of running the exact same binary, you need the compilers to match, or, you need to treat the compiler as part of the tested target.

> To compare system benchmarks, you need the same compiler options and compiler version.

A workaround for this is to ship and test binaries (although this approach may miss out on new processor instructions). For example, as was done with these dhrystone results. Interestingly, they also share non-optimized Dhrystone results, to better understand compiler differences.

Index Score

The final report from UnixBench has a "System Benchmarks Index Score": a single number to summarize the system's performance. The USAGE file describes it as:

```
The BYTE Index
==============

The purpose of this test is to provide a basic indicator of the performance
of a Unix-like system; hence, multiple tests are used to test various
aspects of the system's performance.  These test results are then compared
to the scores from a baseline system to produce an index value, which is
generally easier to handle than the raw sores.  The entire set of index
values is then combined to make an overall index for the system.

Since 1995, the baseline system has been "George", a SPARCstation 20-61
with 128 MB RAM, a SPARC Storage Array, and Solaris 2.3, whose ratings
were set at 10.0.
```

Yes, it's old, but that's not a problem (eg, VAX MIPS). A problem, again, is that the compiler options and version are part of the score, and yet, this score is used to compare servers – not compilers.

For example (and there are many I could pick from), ServerBear have a VPS UnixBench Leaderboard for these Index Scores, stating:

> If you're in the market for a VPS with a high system performance then the UnixBench score is one of the metrics you should be paying attention to.

They are nice enough to share their source code that gathers these UnixBench scores (much more trustworthy than vendors who don't!). It looks like UnixBench is executed via "./Run", which compiles using whatever gcc version the system has already or is added from its default package repository. Which for the systems I'm testing right now, means three different versions of gcc.

Improving UnixBench

So while this is a neat example of compiler differences, how do we actually fix UnixBench? Well, I'm not sure it's really a problem with UnixBench. The project website does warn you about this:

> The results will depend not only on your hardware, but on your operating system, libraries, and even compiler.

As does the USAGE file under the "Interpreting the Results" section, which even suggests:

> So you may want to make sure that all your test systems are running the same version of the OS; or at least publish the OS and compuiler versions with your results.

Good advice, but I frequently see UnixBench results used without this information.

I think the biggest problem is how people are using UnixBench and other benchmarks: fire-and-forget, hoping for a quick result. And doing so without reading the documentation, or performing active benchmarking to confirm what was measured.

UnixBench could help by making `-O0` the default OPTON, reducing compiler differences. This does not – as is commonly believed – turn off all optimizations (zero optimizations). It uses fewer of them. The optimization I described in my let me obfuscate that for you post was still enabled at -O0, for example. Even so, -O0 may make the differences negligible.

Avoiding Trouble

If you want to compare different servers using benchmarks that you compile, you need the compilers to match, or you need to take that into consideration. This should be something you unearth by following an active benchmarking approach, where you study and understand what the benchmark really does.

As for UnixBench: if it does encourage you to switch to a new system because it has a higher UnixBench Index Score, you might want to try upgrading your compiler first!

(I never discussed problems with the UnixBench micro-benchmarks themselves – maybe next time.)

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).