Marcos Woehrmann <marcos.woehrmann@artifex.com>

math is hard

10 messages

Marcos H. Woehrmann <marcos.woehrmann@artifex.com> **Thu, Sep 10, 2009 at 2:46 PM**
To: tech <tech@artifex.com>

I've been tracking down some the local cluster regressions in the PCL code and after much time running gdb have found a difference in the hypot() function on the MacPro and iMac vs. Linux. The following code demonstrates the difference:

```
#include <stdio.h>
#include <math.h>

int main(void) {
 double x=8216.053934050371;
 double y=982.13340514204765;

 double t=hypot(x,y);

 printf("%20.15f\n",t);

 return(0);
}
```

Compiled with 'cc t1.c -lm' and run on any Linux box I have access to it produces:

```
8274.547013143406730
```

But on Henry's MacPro and my iMac it produces:

```
8274.547013143408549
```

The difference is small, but in gs/base/gxpdash.c this difference results in the variable fraction being set to 0.099999999999999992 vs. 0.10000000000000001. The value of fraction is later used to set the variable fx which is ends up being 28 vs. 29 due to a conversion from floating to fixed point (this is on line 122, in case you want to follow along).

As you were probably aware 28 != 29 and this results in minor rendering differences which are caught as md5sum differences in the local cluster regressions.

As an experiment I rounded the output of hypot() to five decimal places and this fixes the problem, but that feels a little hackish and I assume that somone has a cleaner solution or at least wants to comment on this (where is Igor when you need him :-).

marcos

--
Marcos H. Woehrmann

marcos.woehrmann@artifex.com

Henry Stiles <henry.stiles@artifex.com> **Thu, Sep 10, 2009 at 9:02 PM**
To: "Marcos H. Woehrmann" <marcos.woehrmann@artifex.com>
Cc: tech <tech@artifex.com>

Nice find! casper (a linux box) gets your mac answer. I'd guess a
gcc change is responsible, assuming your linux boxes are at gcc 4.3,
the macs are a few versions behind and casper hasn't been updated for
a while.

Henry
[Quoted text hidden]

Ralph Giles <ralph.giles@artifex.com> **Fri, Sep 11, 2009 at 9:36 AM**
To: "Marcos H. Woehrmann" <marcos.woehrmann@artifex.com>
Cc: tech <tech@artifex.com>

On Thu, Sep 10, 2009 at 2:46 PM, Marcos H. Woehrmann
<marcos.woehrmann@artifex.com> wrote:

> Compiled with 'cc t1.c -lm' and run on any Linux box I have access to
> it produces:
>
> 8274.547013143406730
>
> But on Henry's MacPro and my iMac it produces:
>
> 8274.547013143408549

My linux box (fedora 11 x86_64, gcc 4.4.1 20090725 (Red Hat 4.4.1-2))
gives the former by default and with -O0 and the latter with -O1 and
above. We pass -O2 to the our builds, so this shouldn't be a source of
indeterminism. Or it's not a sufficient example of the phenomena, at
least.

The cluster is different still. All machines in a given cluster are
the same. (darn!) The 32 bit 'red' nodes, running a 32 bit version
return ...08549 with either -O0 or -O2. The 64 bit 'green' and
'orange' nodes also return ...08549 running the 32 bit code with
either -O0 or -O2, but return ...06730 running a 64 bit version with
either optimization setting, in contrast to what I see on my own 64
bit machine. This is using gcc version 4.1.2 20080704 (Red Hat
4.1.2-44) for both the 32 bit and 64 bit builds, although each is
natively hosted on the two different targets.

-r

P.S. I also tried -ffast-math and -fno-fast-math locally, but it made
no difference.

Ralph Giles <ralph.giles@artifex.com> **Fri, Sep 11, 2009 at 9:51 AM**
To: "Marcos H. Woehrmann" <marcos.woehrmann@artifex.com>
Cc: tech <tech@artifex.com>

I do get consistent results on my local machine with -frounding-math
(...06730). I'm not sure why that makes a difference; are we messing
with the rounding mode anywhere?

Anyway, you might try adding that to the cluster builds and seeing if
it helps with repeatability.

-r

Ralph Giles <ralph.giles@artifex.com> **Fri, Sep 11, 2009 at 10:33 AM**
To: "Marcos H. Woehrmann" <marcos.woehrmann@artifex.com>
Cc: tech <tech@artifex.com>

On Fri, Sep 11, 2009 at 9:51 AM, Ralph Giles <ralph.giles@artifex.com> wrote:

> I do get consistent results on my local machine with -frounding-math
> (...06730). I'm not sure why that makes a difference; are we messing
> with the rounding mode anywhere?

Apparently this issue is fairly well known on x86 because the fp
registers are 80 bits wide, but a double is only 64 bits wide, so one
gets different answers depending on whether an result is stored to
memory or kept in a register. The options -fround-math and
-ffloat-store counter this in different ways, but they both add extra
instructions around all fp calculations, so they may slow things down
unacceptably, even for testing purposes.

However, I'm told this is less of an issue in x86_64 because -mfpu=sse
is the default on that target (it may also be for 32 bit MacOS builds,
since all intel macs have an sse unit) and this has 64 bit registers.
Or there could be something else going on. On my x86-64 linux machine
the hypot call is getting inlined with -O2 (i.e. -lm isn't required)
and -fno-builtin gives consistent results regardless of the
optimization level, so it may be something with the intrinsic vs
function call implementations.

-r

Ralph Giles <ralph.giles@artifex.com> **Fri, Sep 11, 2009 at 11:08 AM**
To: "Marcos H. Woehrmann" <marcos.woehrmann@artifex.com>
Cc: tech <tech@artifex.com>

Looking at the assembly is more enlightening. My gcc evaluates the
hypot() at compile time with -O2 and the executable code is equivalent
to printf("%20.15f\n",8274.547013143408549); So this charming
variation is all in whether gcc's optimizer is invoked.

Marcos, is this equivalent to gs code that shows indeterminism in a
real run? Is pre-computation with different rounding rules a
reasonable explanation?

Passing the values from the command line works around the problem:

```
$ cat hypotest2.c
/* test of hypot() implementation.
  calling us with '8216.053934050371 982.13340514204765'
  gives different results on different platforms
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[]) {
 double x, y;
 double t;

 if (argc != 3) {
 printf("usage: '%s <x> <y>' prints hypot(x, y)\n", argv[0]);
 return 1;
 }

 x = atof(argv[1]);
 y = atof(argv[2]);

 t = hypot(x, y);

 printf("%20.15lf\n", t);

 return (0);
}

$ gcc -g -Wall hypotest.c -lm && ./a.out 8216.053934050371 982.13340514204765
8274.547013143406730
$ gcc -g -O0 -Wall hypotest.c -lm && ./a.out 8216.053934050371
982.13340514204765
8274.547013143406730
$ gcc -g -O2 -Wall hypotest.c -lm && ./a.out 8216.053934050371
982.13340514204765
8274.547013143408549
```

 -r

Marcos H. Woehrmann <marcos.woehrmann@artifex.com> **Sat, Sep 12, 2009 at 11:52 PM**
To: Ralph Giles <ralph.giles@artifex.com>
Cc: tech <tech@artifex.com>

Ralph,

I'm becoming more confused by this issue the more I look into it. With -O0 the use of -frounding-math and/or -ffloat-store makes no difference, my iMac and my i7 produce different results. My amd64 box matches my i7 with -O0 but with -O1 or higher the iMac and the i7 match and the amd64 doesn't, here's a summary showing the last for digits of the test results:

```
opt i7 iMac amd64
-O0 6730 8549 6730
-O1 8549 8549 6730
-O2 8549 8549 6730
```

The iMac is running Snow Leopard with the updated Xcode which is gcc 4.2.1 and generating x86_64 code by default, the i7 is Ubuntu 9.04 with gcc 4.3.3, the amd64 system runs gcc 4.1.3 and both produce x86_64 code as well.

The other part that is confusing me is that the i7 and the iMac match with -O2, which is the how the local cluster compiles the code. So at least in this example case there shouldn't be differences; I'm running with -O0 as part of the debugging, which is how I found the problem in the first place, but now I'm not sure that the problem I found is the problem the cluster is seeing.

I'll spend some more time on this and try to become less confused...

marcos
[Quoted text hidden]
--
[Quoted text hidden]

Marcos H. Woehrmann <marcos.woehrmann@artifex.com> **Thu, Oct 22, 2009 at 8:08 PM**
To: Ralph Giles <ralph.giles@artifex.com>
Cc: tech <tech@artifex.com>

Ralph,

I've finally had a chance to get back to this and have done a more complete analysis of the results of hypot() with various compiler optimization settings, compiler options (-/-frounding-math/-ffloat-store), and storing the values in the program (which allows pre-computing of the value)/reading them in from the command line.

The short answer is there appears to be no combination of options that allow a MacPro running Mac OS X and an i7 running Linux to produce the same values if the values are not stored in the program.

Macs always produce the same value, independent of cpu (G5/Core 2 Duo/Xeon), word size (32/64), gcc version (4.0.1/4.2.1), OS (10.5/10.6), compiler optimization (O0/O1/O2), other compiler options (-/-frounding-math/-ffloat-store), and whether or not the values are stored in the program or passed as command line parameters.

An i7 running Linux produces the same answer as the Mac only if the values are stored in the program and -O1 or -O2 is used and -frounding-match is not used.

Unless you or anyone else has an objection I'm going to temporarily modify the code to round the hypot() output to some number of significant digits (5?) and run with this for a while to see what it does to the regression output.

marcos
[Quoted text hidden]

Ralph Giles <ralph.giles@artifex.com> **Fri, Oct 23, 2009 at 12:39 AM**
To: "Marcos H. Woehrmann" <marcos.woehrmann@artifex.com>
Cc: tech <tech@artifex.com>

On Thu, Oct 22, 2009 at 8:08 PM, Marcos H. Woehrmann <marcos.woehrmann@artifex.com> wrote:

> Unless you or anyone else has an objection I'm going to temporarily
> modify the code to round the hypot() output to some number of
> significant digits (5?) and run with this for a while to see what it
> does to the regression output.

It's annoying to slow the code down just for the sake of the regression tests, but I don't might the experiment. I guess precomputing it as a constant isn't any better.

-r

Henry Stiles <henry.stiles@artifex.com> **Fri, Oct 23, 2009 at 8:03 AM**
To: "Marcos H. Woehrmann" <marcos.woehrmann@artifex.com>
Cc: Ralph Giles <ralph.giles@artifex.com>, tech <tech@artifex.com>

On Thu, Oct 22, 2009 at 9:08 PM, Marcos H. Woehrmann
<marcos.woehrmann@artifex.com> wrote:
> Ralph,
>
> I've finally had a chance to get back to this and have done a more
> complete analysis of the results of hypot() with various compiler
> optimization settings, compiler options
> (-/-frounding-math/-ffloat-store), and storing the values in the
> program (which allows pre-computing of the value)/reading them in from
> the command line.
>
> The short answer is there appears to be no combination of options that
> allow a MacPro running Mac OS X and an i7 running Linux to produce the
> same values if the values are not stored in the program.
>
> Macs always produce the same value, independent of cpu (G5/Core 2
> Duo/Xeon), word size (32/64), gcc version (4.0.1/4.2.1), OS
> (10.5/10.6), compiler optimization (O0/O1/O2), other compiler options
> (-/-frounding-math/-ffloat-store), and whether or not the values are
> stored in the program or passed as command line parameters.
>
> An i7 running Linux produces the same answer as the Mac only if the
> values are stored in the program and -O1 or -O2 is used and
> -frounding-match is not used.
>
> Unless you or anyone else has an objection I'm going to temporarily
> modify the code to round the hypot() output to some number of
> significant digits (5?) and run with this for a while to see what it
> does to the regression output.
>
> marcos
>

I'd rather have a #define for cluster testing that uses our own hypot
function. I don't know if you read this paper, it might give you some
ideas:

floating-point-article.pdf
484K