Re: alpha slower than pentium II

Linus Torvalds (*Linus.Torvalds@cs.helsinki.fi*)
Mon, 18 Dec 1995 16:57:38 +0200

- **Messages sorted by:** [date][thread][subject][author]
- **Next message:** Morita: "Re: g77"
- **Previous message:** David Mosberger-Tang: "Re: alpha slower than pentium II"

[sorry for the long inclusions, but David had a lot of good points]

David Mosberger-Tang: "Re: alpha slower than pentium II" (Dec 14, 20:43):
>
> *Yan-Song> I am looking for a small machine with good FP performance,*
> *Yan-Song> I don't care the software stuff, since the only software I*
> *Yan-Song> use is gcc. I think the intel is better for integer*
> *Yan-Song> performance and x86 compability which means more*
> *Yan-Song> software. But finally I find that I have to go with intel*
> *Yan-Song> only for floating point performance. Seems the alpha does*
> *Yan-Song> not have any advantage if your budget is less than 20k.*
>
> *OK, I ran your code on a number of different machines in different*
> *configurations. Here is the summary:*
>
> *with*
> *current using*
> *Linux OSF/1*
> *libm libm OS compiler*
> *[secs] [secs]*
>
> *Alpha 21064A 275MHz (Cabriolet): 222.1 43.2 Linux 1.3.45 gcc 2.7.1*
> *Alpha 21064A 233MHz (AS 200): 51.2 OSF/1 v3.x cc -migrate -O4*
> *Alpha 21064A 233MHz (AS 200): 48.8 OSF/1 v3.x gcc 2.6.3*
> *Pentium 90/100 (Intel system): 109.5 Linux 1.3.x gcc 2.6.3*
> *Alpha 21066 166MHz (Noname): 706.6 214.5 Linux 1.3.45 gcc 2.7.1*
>
> *All machines had at least 16MB of memory and memory demand of*
> *the test program is small.*
>
> *First off, why are things so slow on the Alpha when using libm that*
> *comes with libc-linux-0.39? Well, libm hasn't been optimized at all*
> *and it turns out that the test program spends almost all of its time*
> *in sqrt() which turns out to be about as badly implemented as it could*
> *be (it computes one bit at a time!). Just to verify this point, I*
> *replaced it with a quick & dirty sqrt algorithm that's a little more*

> *sane (but doesn't care to get all those inexact and lsb things right).*
> *Just by replacing sqrt(), time on my Cabriolet dropped down from 222.1*
> *to 79.04 seconds. How is that for an improvement? :) I don't have the*
> *mathematical background (or the desire, for that matter), to optimize*
> *libm, but it sure looks like there is a lot to be gained by spending a*
> *little time on it (hint, hint... :)*
>
> *Second, the Noname is doing really quite bad on this testprogram.*
> *It's almost 5 times slower than the Cabriolet that I have. The clock*
> *rate accounts only for a factor of 1.65. So there still is a factor 3*
> *slow that needs to be explained. I'm not certain what it is but the*
> *Cabriolet has both bigger primary caches (16KB vs. 8KB) and the*
> *secondary cache is also a lot bigger (2MB vs 256KB). So my *guess* is*
> *that for some reason the CPU simply doesn't get to execute because it*
> *spends all the time waiting for memory accesses to complete. For*
> *example, it may be that the program's access pattern is such that with*
> *the given cache sizes there are a lot of conflict misses. Therefore,*
> *it would be interesting to see how the Noname would do on a machine*
> *with a 1MB cache and/or with a 233MHz CPU (that CPU also has the 16KB*
> *primary caches). Any takers?*

Ok, I have this narrowed down pretty well, I think.

The reason that the NoName is so much slower than the Cabrio is probably not any cache issues as much as the fact that the NoName has the EV4 chip, which is worse at division than the EV45 in the Cabrio. And the reason that the OSF/1 math libraries do so well is that they use another algorithm altogether, which doesn't need division..

Anyway, thanks to David for forwarding to me a copy of a drafted paper by Prof W. Kahan and K.C. Ng, written in May, 1986 that has a few different ways of computing the square root of a IEEE double efficiently. I decided I might as well look into it, and try to come up with a better sqrt() routine for Linux/alpha.

Here is my current try at a sqrt() that does the correct thing (including rounding and special case handling). It's not exactly pretty, and it's still a bit slower than the OSF/1 libs, but it's a _lot_ better than it used to be.

No guarantees about working: I _have_ tested this (including special cases), but I have by no means made any exhaustive regression tests on the thing. The math _should_ be ok (assuming the paper David sent me was accurate, and all my limited tests do indicate that it is), and I think I handled all the special cases correctly too. But note the I _think_ part ;-)

Linus

```
----------
/*
* Copyright 1995 Linus Torvalds
*/
#include <errno.h>

static int T2[64]= {
0x1500, 0x2ef8, 0x4d67, 0x6b02, 0x87be, 0xa395, 0xbe7a, 0xd866,
0xf14a, 0x1091b,0x11fcd,0x13552,0x14999,0x15c98,0x16e34,0x17e5f,
0x18d03,0x19a01,0x1a545,0x1ae8a,0x1b5c4,0x1bb01,0x1bfde,0x1c28d,
0x1c2de,0x1c0db,0x1ba73,0x1b11c,0x1a4b5,0x1953d,0x18266,0x16be0,
0x1683e,0x179d8,0x18a4d,0x19992,0x1a789,0x1b445,0x1bf61,0x1c989,
0x1d16d,0x1d77b,0x1dddf,0x1e2ad,0x1e5bf,0x1e6e8,0x1e654,0x1e3cd,
0x1df2a,0x1d635,0x1cb16,0x1be2c,0x1ae4e,0x19bde,0x1868e,0x16e2e,
0x1527f,0x1334a,0x11051,0xe951, 0xbe01, 0x8e0d, 0x5924, 0x1edd
};

#define lobits(x) (((unsigned int *)&x)[0])
#define hibits(x) (((unsigned int *)&x)[1])

static inline double initial_guess(double x, unsigned int k)
{
double ret = 0.0;

k = 0x5fe80000 - (k >> 1);
k = k - T2[63&(k>>14)];
hibits(ret) = k;
return ret;
}

#define two_to_minus_5 (0.5*0.5*0.5*0.5*0.5)
#define two_to_minus_10 (two_to_minus_5*two_to_minus_5)
#define two_to_minus_30 (two_to_minus_10*two_to_minus_10*two_to_minus_10)

/* up = nextafter(1,+Inf), dn = nextafter(1,-Inf) */
static unsigned long __up = 0x3ff0000000000001;
static unsigned long __dn = 0x3fefffffffffffff;

#define dn (*(double *)&__dn)
#define up (*(double *)&__up)

/* Multiply with chopping rounding.. */
static inline double choppedmul(double a, double b)
{
__asm__("multc %1,%2,%0"
:"=f" (a)
:"f" (a), "f" (b));
return a;
```

```
}

double sqrt(double x)
{
unsigned long k;
double y, z, zp, zn;

*(double *)&k = x;

/* Negative or NaN or Inf */
if ((k >> 52) >= 0x7ff)
goto special;
y = initial_guess(x, k >> 32);
y = y*(1.5 - 0.5*x*y*y);
y = y*((1.5 - two_to_minus_30) - 0.5*x*y*y);
z = x*y;
z = z + 0.5*z*(1-z*y);
zp = choppedmul(z,dn);
zn = choppedmul(z,up);
y = choppedmul(z, zp) - x;
x = choppedmul(z, zn) - x;
/* I can't get gcc to use fcmov's.. */
__asm__("fcmovge %2,%3,%0"
:"=f" (z)
:"0" (z), "f" (y), "f" (zp));
__asm__("fcmovlt %2,%3,%0"
:"=f" (z)
:"0" (z), "f" (x), "f" (zn));
return z;
special:
/* throw away sign bit */
k <<= 1;
/* -0 */
if (!k)
return x;
/* special? */
if ((k >> 53) == 0x7ff) {
/* NaN? */
if (k << 11)
return x;
/* sqrt(+Inf) = +Inf */
if (x > 0)
return x;
}
/* -ve or -Inf */
errno = EDOM;
k = ~0UL;
```

```
return *(double *)&k;
}
```

- **Next message:** Morita: "Re: g77"
- **Previous message:** David Mosberger-Tang: "Re: alpha slower than pentium II"