
Subject: "better" version of Iscale functions

Posted by [mdelfede](#) on Tue, 01 Apr 2008 21:10:55 GMT

[View Forum Message](#) <> [Reply to Message](#)

In file 'mathutils.cpp' the Iscale...() functions use some floating point math, when assembly is unavailable OR when it's not compatible with intel syntax (I.E. GCC and MinGW). That makes it slow and not showing divide-by-0 errors when third argument is 0. So, here an (IMHO) better version of such functions :

```
#include "Core.h"
```

```
// iscale: computes x * y / z.
```

```
#ifdef flagGCC
```

```
#define __USE_64BIT_MATH__
```

```
#endif
```

```
namespace UPP
```

```
int iscale(int x, int y, int z)
```

```
{
```

```
#ifdef __NOASSEMBLY__
```

```
#ifndef __USE_64BIT_MATH__
```

```
    return int(x * (double)y / z);
```

```
#else
```

```
    int64_t res = x;
```

```
    res *= y;
```

```
    res /= z;
```

```
    return (int)res;
```

```
#endif
```

```
#else
```

```
    __asm
```

```
{
```

```
    mov eax, [x]
```

```
    imul [y]
```

```
    idiv [z]
```

```
}
```

```
#endif
```

```
}
```

```
// iscalefloor: computes x * y / z, rounded towards -infty.
```

```
int iscalefloor(int x, int y, int z)
```

```
{
```

```
#ifdef __NOASSEMBLY__
```

```
#ifndef __USE_64BIT_MATH__
```

```
    return (int)ffloor(x * (double)y / z);
```

```
#else
```

```

int64_t res = x;
int64_t mulres = res * y;
res = mulres / z;
if(res * z != mulres)
    res--;
return (int)res;
#endif
#else
__asm
{
    mov eax, [x]
    imul [y]
    idiv [z]
    and edx, edx
    jge __1
    dec eax
__1:
}
#endif
}

```

// iscaleceil: computes $x * y / z$, rounded towards +infty.

```

int iscaleceil(int x, int y, int z)
{
#ifdef __NOASSEMBLY__
#ifndef __USE_64BIT_MATH__
return fceil(x * (double)y / z);
#else
int64_t res = x;
int64_t mulres = res * y;
res = mulres / z;
if(res * z != mulres)
    res++;
return (int)res;
#endif
#else
__asm
{
    mov eax, [x]
    imul [y]
    idiv [z]
    and edx, edx
    jle __1
    inc eax
__1:
}
#endif
}

```

```
}
```

BTW, we could completely drop the assembly code, as-is it's not portable between compilers with greater integer width.

My version is also **not** portable on compilers with 64 bit wide integers, but can be made ok just changing function prototype :

```
int32_t iscale(int32_t x, int32_t y, int32_t z)
```

Leaving so to the compiler the integer width check and warnings.

Attached here the patched 'mathutil.cpp' (NO patched function prototype, as it'll require Core.h patch too).

Ciao

Max

File Attachments

1) [mathutil.cpp](#), downloaded 297 times

Subject: Re: "better" version of Iscale functions

Posted by [mirek](#) on Wed, 02 Apr 2008 13:11:16 GMT

[View Forum Message](#) <> [Reply to Message](#)

mdelfede wrote on Tue, 01 April 2008 17:10

BTW, we could completely drop the assembly code, as-is it's not portable between compilers with greater integer width.

Yes, but int64 does not come cheap on non-64 architecture. Maybe even that FP computation could be faster. Of course, as long as FP is performed by HW. For ARM this new iscale can be good.

Quote:

My version is also **not** portable on compilers with 64 bit wide integers, but can be made ok just changing function prototype :

```
int32_t iscale(int32_t x, int32_t y, int32_t z)
```

Leaving so to the compiler the integer width check and warnings.

IMO, that really is not that bug trouble, as any serious portable code should work with 32-bit int.

Mirek

Subject: Re: "better" version of Iscale functions
Posted by [mdelfede](#) on Wed, 02 Apr 2008 14:48:36 GMT
[View Forum Message](#) <> [Reply to Message](#)

luzr wrote on Wed, 02 April 2008 15:11mdelfede wrote on Tue, 01 April 2008 17:10
BTW, we could completely drop the assembly code, as-is it's not portable between compilers with greater integer width.

Yes, but int64 does not come cheap on non-64 architecture. Maybe even that FP computation could be faster. Of course, as long as FP is performed by HW. For ARM this new iscale can be good.

I thought you were in holydays

Back to Iscale, I don't know about modern processors that does have an hardware ftp and doesn't have 32x32->64 bit mul and 64/32 ->32 bit div core instructions... but I can be wrong.
Yet, I don't remember if intel ones works just with unsigned or signed or both integers...
BTW, I noticed that my iscale needs to work with 32 bit result; if not it'll use full 64 bit math for the multiply (in iscalefloor and iscaleceil) which can be slow.
I guess that using 32x32 multiply and 64/32 division, GCC translates it directly in DIV and MUL, but I've not checked yet.

Quote:

My version is also **not** portable on compilers with 64 bit wide integers, but can be made ok just changing function prototype :

```
int32_t iscale(int32_t x, int32_t y, int32_t z)
```

Leaving so to the compiler the integer width check and warnings.

IMO, that really is not that bug trouble, as any serious portable code should work with 32-bit int.

Mirek[/quote]

I can agree, but I think more and more that the lack of width specs in C++ is really a nasty stuff. Now it's too late, but if I'd have to write a framework from scratch, I'd use some typedef'd int8, int16, int32 and so on stuffs.

BTW, our Iscale is much better than micro\$oft's one, MulDiv, which returns -1 on 0 divisor.... so on both

```
MulDiv(1,-1,1)
```

MulDiv(a, b,0)

you get -1 as result.

Another possibility would be to use GCC built in asm, which is much different in syntax from Intel one (MS), but it's quite complicated, even if much more powerful.

Max

Subject: Re: "better" version of Iscale functions
Posted by [mirek](#) on Sun, 06 Apr 2008 02:47:32 GMT
[View Forum Message](#) <> [Reply to Message](#)

mdelfede wrote on Wed, 02 April 2008 10:48

Back to Iscale, I don't know about modern processors that does have an hardware ftp and doesn't have 32x32->64 bit mul and 64/32 ->32 bit div core instructions... but I can be wrong.

Yet, I don't remember if intel ones works just with unsigned or signed or both integers...

BTW, I noticed that my iscale needs to work with 32 bit result; if not it'll use full 64 bit math for the multiply (in iscalefloor and iscaleceil) which can be slow.

I guess that using 32x32 multiply and 64/32 division, GCC translates it directly in DIV and MUL, but I've not checked yet.

Well, this is what MSC does seem to do to divide these numbers:

```
0041A920 push edi
0041A921 push esi
0041A922 push ebx
0041A923 xor edi,edi
0041A925 mov eax,[esp+0x14]
0041A929 or eax,eax
0041A92B jnl 0x41a941
0041A92D inc edi
0041A92E mov edx,[esp+0x10]
0041A932 neg eax
0041A934 neg edx
0041A936 sbb eax,byte +0x0
0041A939 mov [esp+0x14],eax
0041A93D mov [esp+0x10],edx
0041A941 mov eax,[esp+0x1c]
0041A945 or eax,eax
0041A947 jnl 0x41a95d
0041A949 inc edi
0041A94A mov edx,[esp+0x18]
0041A94E neg eax
```

0041A950 neg edx
0041A952 sbb eax,byte +0x0
0041A955 mov [esp+0x1c],eax
0041A959 mov [esp+0x18],edx
0041A95D or eax,eax
0041A95F jnz 0x41a979
0041A961 mov ecx,[esp+0x18]
0041A965 mov eax,[esp+0x14]
0041A969 xor edx,edx
0041A96B div ecx
0041A96D mov ebx,eax
0041A96F mov eax,[esp+0x10]
0041A973 div ecx
0041A975 mov edx,ebx
0041A977 jmp short 0x41a9ba
0041A979 mov ebx,eax
0041A97B mov ecx,[esp+0x18]
0041A97F mov edx,[esp+0x14]
0041A983 mov eax,[esp+0x10]
0041A987 shr ebx,1
0041A989 rcr ecx,1
0041A98B shr edx,1
0041A98D rcr eax,1
0041A98F or ebx,ebx
0041A991 jnz 0x41a987
0041A993 div ecx
0041A995 mov esi,eax
0041A997 mul dword [esp+0x1c]
0041A99B mov ecx,eax
0041A99D mov eax,[esp+0x18]
0041A9A1 mul esi
0041A9A3 add edx,ecx
0041A9A5 jc 0x41a9b5
0041A9A7 cmp edx,[esp+0x14]
0041A9AB ja 0x41a9b5
0041A9AD jc 0x41a9b6
0041A9AF cmp eax,[esp+0x10]
0041A9B3 jna 0x41a9b6
0041A9B5 dec esi
0041A9B6 xor edx,edx
0041A9B8 mov eax,esi
0041A9BA dec edi
0041A9BB jnz 0x41a9c4
0041A9BD neg edx
0041A9BF neg eax
0041A9C1 sbb edx,byte +0x0
0041A9C4 pop ebx
0041A9C5 pop esi

0041A9C6 pop edi
0041A9C7 ret 0x10

I would say we should better learn GCC assembly syntax

Mirek

Subject: Re: "better" version of Iscale functions
Posted by [mirek](#) on Sun, 06 Apr 2008 02:51:33 GMT
[View Forum Message](#) <> [Reply to Message](#)

I can agree, but I think more and more that the lack of width specs in C++ is really a nasty stuff. Now it's too late, but if I'd have to write a framework from scratch, I'd use some typedef'd int8, int16, int32 and so on stuffs.

Well, that might not be that good either.

I see "int" as type that is at least 32-bit (not correct, but reasonable guess today) and is the most optimal for target architecture.

There might be CPU where int is 64-bit and 32-bit int is in fact less optimal. In that case, using int32 everywhere would mean less optimal code.

Mirek

Subject: Re: "better" version of Iscale functions
Posted by [mdelfede](#) on Sun, 06 Apr 2008 17:36:21 GMT
[View Forum Message](#) <> [Reply to Message](#)

luzr wrote on Sun, 06 April 2008 04:47mdelfede wrote on Wed, 02 April 2008 10:48
Back to Iscale, I don't know about modern processors that does have an hardware ftp and doesn't have 32x32->64 bit mul and 64/32 ->32 bit div core instructions... but I can be wrong.
Yet, I don't remember if intel ones works just with unsigned or signed or both integers...
BTW, I noticed that my iscale needs to work with 32 bit result; if not it'll use full 64 bit math for the multiply (in iscalefloor and iscaleceil) which can be slow.
I guess that using 32x32 multiply and 64/32 division, GCC translates it directly in DIV and MUL, but I've not checked yet.

Well, this is what MSC does seem to do to divide these numbers:

0041A920 push edi
0041A921 push esi
0041A922 push ebx
0041A923 xor edi,edi
0041A925 mov eax,[esp+0x14]
0041A929 or eax,eax
0041A92B jnl 0x41a941
0041A92D inc edi
0041A92E mov edx,[esp+0x10]
0041A932 neg eax
0041A934 neg edx
0041A936 sbb eax,byte +0x0
0041A939 mov [esp+0x14],eax
0041A93D mov [esp+0x10],edx
0041A941 mov eax,[esp+0x1c]
0041A945 or eax,eax
0041A947 jnl 0x41a95d
0041A949 inc edi
0041A94A mov edx,[esp+0x18]
0041A94E neg eax
0041A950 neg edx
0041A952 sbb eax,byte +0x0
0041A955 mov [esp+0x1c],eax
0041A959 mov [esp+0x18],edx
0041A95D or eax,eax
0041A95F jnz 0x41a979
0041A961 mov ecx,[esp+0x18]
0041A965 mov eax,[esp+0x14]
0041A969 xor edx,edx
0041A96B div ecx
0041A96D mov ebx,eax
0041A96F mov eax,[esp+0x10]
0041A973 div ecx
0041A975 mov edx,ebx
0041A977 jmp short 0x41a9ba
0041A979 mov ebx,eax
0041A97B mov ecx,[esp+0x18]
0041A97F mov edx,[esp+0x14]
0041A983 mov eax,[esp+0x10]
0041A987 shr ebx,1
0041A989 rcr ecx,1
0041A98B shr edx,1
0041A98D rcr eax,1
0041A98F or ebx,ebx
0041A991 jnz 0x41a987
0041A993 div ecx
0041A995 mov esi,eax
0041A997 mul dword [esp+0x1c]


```
0041A99B mov ecx,eax
0041A99D mov eax,[esp+0x18]
0041A9A1 mul esi
0041A9A3 add edx,ecx
0041A9A5 jc 0x41a9b5
0041A9A7 cmp edx,[esp+0x14]
0041A9AB ja 0x41a9b5
0041A9AD jc 0x41a9b6
0041A9AF cmp eax,[esp+0x10]
0041A9B3 jna 0x41a9b6
0041A9B5 dec esi
0041A9B6 xor edx,edx
0041A9B8 mov eax,esi
0041A9BA dec edi
0041A9BB jnz 0x41a9c4
0041A9BD neg edx
0041A9BF neg eax
0041A9C1 sbb edx,byte +0x0
0041A9C4 pop ebx
0041A9C5 pop esi
0041A9C6 pop edi
0041A9C7 ret 0x10
```

I would say we should better learn GCC assembly syntax

Mirek

Wow, that's what I call "optimizing compiler".....

Max

Subject: Re: "better" version of Iscale functions
Posted by [mdelfede](#) on Sun, 06 Apr 2008 18:05:38 GMT
[View Forum Message](#) <> [Reply to Message](#)

Btw, I guess GCC does some better optimizations in this case :

```
int iscale(int x, int y, int z)
{
    int64_t res = x;
    res *= y;
    res /= z;
    return (int)res;
}
```

gets translated as :

```
.globl _ZN3Upp6iscaleEiii
.type _ZN3Upp6iscaleEiii, @function
_ZN3Upp6iscaleEiii:
.LFB4039:
.file 4 "/home/massimo/sources/upp-svn/uppsrc/Core/mathutil.cpp"
.loc 4 11 0
pushq %rbp
.LCFI15:
movq %rsp, %rbp
.LCFI16:
movl %edi, -20(%rbp)
movl %esi, -24(%rbp)
movl %edx, -28(%rbp)
.LBB2:
.loc 4 17 0
movl -20(%rbp), %eax
cltq
movq %rax, -8(%rbp)
.loc 4 18 0
movl -24(%rbp), %eax
movslq %eax, %rdx
movq -8(%rbp), %rax
imulq %rdx, %rax
movq %rax, -8(%rbp)
.loc 4 19 0
movl -28(%rbp), %eax
cltq
movq -8(%rbp), %rdx
movq %rax, %rcx
movq %rdx, %rax
sarq $63, %rdx
idivq %rcx
movq %rax, -8(%rbp)
.loc 4 20 0
movq -8(%rbp), %rax
.LBE2:
.loc 4 30 0
leave
ret
```

If you drop assembly directive and/or line references and other stuffs, you can see that work is done with just one imul and one idiv... like your assembly version. The rest is just registry moving stuffs. I don't know what happens if you compile it with full optimization, but I guess mostly of them

just disappears.

And I think that's still much quicker than floating point version.

Max

Subject: Re: "better" version of Iscale functions

Posted by [mdelfede](#) on Sun, 06 Apr 2008 18:14:08 GMT

[View Forum Message](#) <> [Reply to Message](#)

... and, just because I'm curious, that's the -O3 full optimized GCC assembly code :

```
movslq %edi,%rax
movslq %esi,%rsi
movslq %edx,%rdx
imulq %rsi, %rax
movq %rdx, %rcx
movq %rax, %rdx
sarq $63, %rdx
idivq %rcx
ret
```

Well, I could say that's just 2-3 mov's ahead from by hand assembly code....

Max

Subject: Re: "better" version of Iscale functions

Posted by [mirek](#) on Thu, 10 Apr 2008 00:44:58 GMT

[View Forum Message](#) <> [Reply to Message](#)

Now, this is an argument

Applied. (I have deleted float version altogether - either it is MSC on Win32 which is unable to do it right using int64, or it is GCC or MSC on ARM and ARM does not have FP).

Mirek

Subject: Re: "better" version of Iscale functions

Posted by [mirek](#) on Thu, 10 Apr 2008 00:46:44 GMT

[View Forum Message](#) <> [Reply to Message](#)

Oh, and changed to "int64" too...

Mirek

Subject: Re: "better" version of Iscale functions
Posted by [mdelfede](#) on Thu, 10 Apr 2008 13:07:51 GMT
[View Forum Message](#) <> [Reply to Message](#)

luzr wrote on Thu, 10 April 2008 02:44Now, this is an argument

Applied. (I have deleted float version altogether - either it is MSC on Win32 which is unable to do it right using int64, or it is GCC or MSC on ARM and ARM does not have FP).

Mirek

Well, even if usually MSC optimizes better than GCC, that's not always true
BTW, I'm surprised on how bad does it MSC, having hardware instruction on x86 processors for 32x32->64 bit multiply (signed AND unsigned) and 64/32->32 bit divide. Maybe they're still anchored to good old 8088 assembly code...

Max

Subject: Re: "better" version of Iscale functions
Posted by [mirek](#) on Thu, 10 Apr 2008 16:01:32 GMT
[View Forum Message](#) <> [Reply to Message](#)

mdelfede wrote on Thu, 10 April 2008 09:07luzr wrote on Thu, 10 April 2008 02:44Now, this is an argument

Applied. (I have deleted float version altogether - either it is MSC on Win32 which is unable to do it right using int64, or it is GCC or MSC on ARM and ARM does not have FP).

Mirek

Well, even if usually MSC optimizes better than GCC

Actually, this is no longer true. I am not sure when it happened (IMO sometime around 4.1 version , but current GCC produces faster code on average.

Quote:

BTW, I'm surprised on how bad does it MSC, having hardware instruction on x86 processors for 32x32->64 bit multiply (signed AND unsigned) and 64/32->32 bit divide. Maybe they're still anchored to good old 8088 assembly code...

I guess they just do not detect that it is 32x32... and just use normal 64x64 routine.

Mirek

Subject: Re: "better" version of Iscale functions
Posted by [mdelfede](#) on Fri, 11 Apr 2008 07:26:43 GMT
[View Forum Message](#) <> [Reply to Message](#)

luzr wrote on Thu, 10 April 2008 18:01mdelfede wrote on Thu, 10 April 2008 09:07luzr wrote on Thu, 10 April 2008 02:44Now, this is an argument

Applied. (I have deleted float version altogether - either it is MSC on Win32 which is unable to do it right using int64, or it is GCC or MSC on ARM and ARM does not have FP).

Mirek

Well, even if usually MSC optimizes better than GCC

Actually, this is no longer true. I am not sure when it happened (IMO sometime around 4.1 version , but current GCC produces faster code on average.

Well, I think that when they "killed" the only true competitor in c++ compilers (Borland) they've lost interest on improving their compiler...

Quote:

I guess they just do not detect that it is 32x32... and just use normal 64x64 routine.

Indeed. But that's a really bad optimizer if it can't detect it.

Max
