Subject: String::Cat optimization
Posted by mirek on Wed, 30 Nov 2011 16:58:00 GMT
View Forum Message <> Reply to Message

By replacing memcpy with

```
inline void svo_memcpy(char *t, const char *s, int len)
{
 switch(len) {
 case 15: t[14] = s[14];
 case 14: t[13] = s[13];
 case 13: t[12] = s[12];
 case 12: t[11] = s[11];
 case 11: t[10] = s[10];
 case 10: t[9] = s[9];
 case  9: t[8] = s[8];
 case  8: t[7] = s[7];
 case  7: t[6] = s[6];
 case  6: t[5] = s[5];
 case  5: t[4] = s[4];
 case  4: t[3] = s[3];
 case  3: t[2] = s[2];
 case  2: t[1] = s[1];
 case  1: t[0] = s[0];
  return;
 }
 memcpy(t, s, len);
}
```

I have recieved about 10% improvemenced in the String::Cat for small values.

Subject: Re: String::Cat optimization
Posted by koldo on Wed, 30 Nov 2011 17:11:36 GMT
View Forum Message <> Reply to Message

Amazing . Is it the same in different environments?

Subject: Re: String::Cat optimization
Posted by dolik.rce on Wed, 30 Nov 2011 17:17:41 GMT
View Forum Message <> Reply to Message

Huh... and I always thought memcpy is the fastest way to copy things

Any ideas how does the compiler optimization magic works in this case? I'd like to understand, it might be useful in other situations as well.

Honza

Subject: Re: String::Cat optimization
Posted by mirek on Wed, 30 Nov 2011 20:15:05 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Wed, 30 November 2011 12:17Huh... and I always thought memcpy is the fastest way to copy things

Any ideas how does the compiler optimization magic works in this case? I'd like to understand, it might be useful in other situations as well.

Honza

Seriously, I am really ambiguos about this optimization, it is really border case. Plus I have only tested with MSC.

Anyway, looking at assembly code, memcpy is really optimized pretty well, but spends a lot of time detecting heavy-lifting scenario (like target and source both aligned etc...), whereas String is all about adding small pieces of data.

The switch leads to simple jump to 'multiplied' position and then 'linear' code up to end. It is a very little bit faster..

All in all, perhaps more data are needed. It should be easy to #ifdef svo_memcpy to regular memcpy...

My benchmarking code was something like this:

```
 String str;
 for(int i = 0; i < 10000000; i++) {
  str.Clear();
  RTIMING("Cat 18");
  str.Cat("Hello", 5);
  str.Cat("Hello", 5);
  str.Cat("Hello", 5);
 }
 for(int i = 0; i < 10000000; i++) {
  str.Clear();
  RTIMING("Cat 40");
  str.Cat("Hello", 5);
  str.Cat("Hello", 5);
  str.Cat("Hello", 5);
```

```
  str.Cat("Hello", 5);
  str.Cat("Hello", 5);
  str.Cat("Hello", 5);
  str.Cat("Hello", 5);
  str.Cat("Hello", 5);
 }
```

before optimization

TIMING Cat 40 : 1.98 s - 198.46 ns (2.17 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 591.60 ms - 59.16 ns (772.00 ms / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

after

TIMING Cat 40 : 1.48 s - 148.37 ns (1.68 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 482.71 ms - 48.27 ns (676.00 ms / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

Subject: Re: String::Cat optimization
Posted by koldo on Thu, 01 Dec 2011 05:18:49 GMT
View Forum Message <> Reply to Message

More cases

There is a slight improvement in MSC but a big one in MinGW.

MSC10 Speed
- Standard
TIMING Cat 40 : 2.34 s - 233.71 ns (2.56 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 839.12 ms - 83.91 ns (1.06 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
- Experimental
TIMING Cat 40 : 2.24 s - 223.96 ns (2.44 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 788.63 ms - 78.86 ns (987.00 ms / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

MSC10 Optimal
- Standard
TIMING Cat 40 : 2.37 s - 237.30 ns (2.57 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 857.95 ms - 85.80 ns (1.05 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
- Experimental
TIMING Cat 40 : 2.27 s - 226.80 ns (2.48 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 893.96 ms - 89.40 ns (1.10 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

MSC9 Speed
- Standard
TIMING Cat 40 : 2.38 s - 238.20 ns (2.61 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 884.96 ms - 88.50 ns (1.12 s / 10000000), min: 0.00 ns, max: 5.00 ms, nesting: 1 - 10000000
- Experimental
TIMING Cat 40 : 2.13 s - 212.92 ns (2.34 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 866.17 ms - 86.62 ns (1.07 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

MSC9 Optimal
- Standard
TIMING Cat 40 : 3.04 s - 304.05 ns (3.24 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 1.04 s - 103.95 ns (1.24 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
- Experimental
TIMING Cat 40 : 2.48 s - 248.49 ns (2.67 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 915.92 ms - 91.59 ns (1.10 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

MINGW 4.5.2 Speed
- Standard
TIMING Cat 40 : 5.59 s - 558.74 ns (5.85 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 1.89 s - 189.04 ns (2.15 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
- Experimental
TIMING Cat 40 : 2.91 s - 290.74 ns (3.18 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 853.43 ms - 85.34 ns (1.13 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

MINGW 4.5.2 Optimal
- Standard
TIMING Cat 40 : 5.54 s - 554.21 ns (5.84 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 1.81 s - 180.81 ns (2.11 s / 10000000), min: 0.00 ns, max: 2.00 ms, nesting: 1 - 10000000

- Experimental
TIMING Cat 40 : 2.84 s - 284.44 ns (3.14 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 824.40 ms - 82.44 ns (1.12 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

Subject: Re: String::Cat optimization
Posted by mirek on Thu, 01 Dec 2011 07:04:44 GMT
View Forum Message <> Reply to Message

koldo wrote on Thu, 01 December 2011 00:18More cases

There is a slight improvement in MSC but a big one in MinGW.

MSC10 Speed
- Standard
TIMING Cat 40 : 2.34 s - 233.71 ns (2.56 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 839.12 ms - 83.91 ns (1.06 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
- Experimental
TIMING Cat 40 : 2.24 s - 223.96 ns (2.44 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 788.63 ms - 78.86 ns (987.00 ms / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

MSC10 Optimal
- Standard
TIMING Cat 40 : 2.37 s - 237.30 ns (2.57 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 857.95 ms - 85.80 ns (1.05 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
- Experimental
TIMING Cat 40 : 2.27 s - 226.80 ns (2.48 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 893.96 ms - 89.40 ns (1.10 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

MSC9 Speed

- Standard
TIMING Cat 40 : 2.38 s - 238.20 ns (2.61 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 884.96 ms - 88.50 ns (1.12 s / 10000000), min: 0.00 ns, max: 5.00 ms, nesting: 1 - 10000000
- Experimental
TIMING Cat 40 : 2.13 s - 212.92 ns (2.34 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 866.17 ms - 86.62 ns (1.07 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

MSC9 Optimal
- Standard
TIMING Cat 40 : 3.04 s - 304.05 ns (3.24 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 1.04 s - 103.95 ns (1.24 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
- Experimental
TIMING Cat 40 : 2.48 s - 248.49 ns (2.67 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 915.92 ms - 91.59 ns (1.10 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

MINGW 4.5.2 Speed
- Standard
TIMING Cat 40 : 5.59 s - 558.74 ns (5.85 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 1.89 s - 189.04 ns (2.15 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
- Experimental
TIMING Cat 40 : 2.91 s - 290.74 ns (3.18 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 853.43 ms - 85.34 ns (1.13 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

MINGW 4.5.2 Optimal
- Standard
TIMING Cat 40 : 5.54 s - 554.21 ns (5.84 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 1.81 s - 180.81 ns (2.11 s / 10000000), min: 0.00 ns, max: 2.00 ms, nesting: 1 - 10000000

- Experimental
TIMING Cat 40 : 2.84 s - 284.44 ns (3.14 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 824.40 ms - 82.44 ns (1.12 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

Well, I guess this sort of vindicates the optimization...

Mirek

Subject: Re: String::Cat optimization
Posted by koldo on Thu, 01 Dec 2011 07:39:41 GMT
View Forum Message <> Reply to Message

Please could somebody try it in Linux?

Subject: Re: String::Cat optimization
Posted by dolik.rce on Thu, 01 Dec 2011 09:21:18 GMT
View Forum Message <> Reply to Message

koldo wrote on Thu, 01 December 2011 08:39Please could somebody try it in Linux?
Here it is:GCC-4.6.2 Optimal with svo_memcpy
TIMING Cat 40 : 2.65 s - 265.34 ns (11.06 s / 10000000), min: 0.00 ns, max: 5.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 0.00 ns - 0.00 ns (7.88 s / 10000000), min: 0.00 ns, max: 3.00 ms, nesting: 1 - 10000000

GCC-4.6.2 Optimal with memcpy
TIMING Cat 40 : 2.94 s - 293.57 ns (11.13 s / 10000000), min: 0.00 ns, max: 4.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 0.00 ns - 0.00 ns (8.11 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

GCC-4.6.2 Speed with svo_memcpy
TIMING Cat 40 : 246.75 ms - 24.68 ns (11.14 s / 10000000), min: 0.00 ns, max: 4.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 0.00 ns - 0.00 ns (7.93 s / 10000000), min: 0.00 ns, max: 4.00 ms, nesting: 1 - 10000000

GCC-4.6.2 Speed with memcpy
TIMING Cat 40 : 2.79 s - 279.30 ns (11.45 s / 10000000), min: 0.00 ns, max: 6.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 2.03 s - 202.80 ns (10.68 s / 10000000), min: 0.00 ns, max: 4.00 ms, nesting: 1 - 10000000

CLANG-2.9 Optimal with svo_memcpy
TIMING Cat 40 : 1.65 s - 165.35 ns (11.07 s / 10000000), min: 0.00 ns, max: 1.00 ms,

nesting: 1 - 10000000
TIMING Cat 18 : 0.00 ns - 0.00 ns (8.09 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

CLANG-2.9 Optimal with memcpy
TIMING Cat 40 : 4.20 s - 420.01 ns (11.33 s / 10000000), min: 0.00 ns, max: 2.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 1.20 s - 119.81 ns (8.32 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

CLANG-2.9 Speed with svo_memcpy
TIMING Cat 40 : 4.28 s - 428.33 ns (11.21 s / 10000000), min: 0.00 ns, max: 2.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 797.30 ms - 79.73 ns (7.72 s / 10000000), min: 0.00 ns, max: 5.00 ms, nesting: 1 - 10000000

CLANG-2.9 Speed with memcpy
TIMING Cat 40 : 5.87 ms - 0.59 ns (11.09 s / 10000000), min: 0.00 ns, max: 2.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 0.00 ns - 0.00 ns (8.36 s / 10000000), min: 0.00 ns, max: 5.00 ms, nesting: 1 - 10000000

Honza

Subject: Re: String::Cat optimization
Posted by mirek on Thu, 01 Dec 2011 10:37:50 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Thu, 01 December 2011 04:21koldo wrote on Thu, 01 December 2011 08:39Please could somebody try it in Linux?
Here it is:GCC-4.6.2 Optimal with svo_memcpy
TIMING Cat 40 : 2.65 s - 265.34 ns (11.06 s / 10000000), min: 0.00 ns, max: 5.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 0.00 ns - 0.00 ns (7.88 s / 10000000), min: 0.00 ns, max: 3.00 ms, nesting: 1 - 10000000

GCC-4.6.2 Optimal with memcpy
TIMING Cat 40 : 2.94 s - 293.57 ns (11.13 s / 10000000), min: 0.00 ns, max: 4.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 0.00 ns - 0.00 ns (8.11 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

GCC-4.6.2 Speed with svo_memcpy
TIMING Cat 40 : 246.75 ms - 24.68 ns (11.14 s / 10000000), min: 0.00 ns, max: 4.00 ms, nesting: 1 - 10000000

TIMING Cat 18 : 0.00 ns - 0.00 ns (7.93 s / 10000000), min: 0.00 ns, max: 4.00 ms, nesting: 1 - 10000000

GCC-4.6.2 Speed with memcpy
TIMING Cat 40 : 2.79 s - 279.30 ns (11.45 s / 10000000), min: 0.00 ns, max: 6.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 2.03 s - 202.80 ns (10.68 s / 10000000), min: 0.00 ns, max: 4.00 ms, nesting: 1 - 10000000

CLANG-2.9 Optimal with svo_memcpy
TIMING Cat 40 : 1.65 s - 165.35 ns (11.07 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 0.00 ns - 0.00 ns (8.09 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

CLANG-2.9 Optimal with memcpy
TIMING Cat 40 : 4.20 s - 420.01 ns (11.33 s / 10000000), min: 0.00 ns, max: 2.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 1.20 s - 119.81 ns (8.32 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

CLANG-2.9 Speed with svo_memcpy
TIMING Cat 40 : 4.28 s - 428.33 ns (11.21 s / 10000000), min: 0.00 ns, max: 2.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 797.30 ms - 79.73 ns (7.72 s / 10000000), min: 0.00 ns, max: 5.00 ms, nesting: 1 - 10000000

CLANG-2.9 Speed with memcpy
TIMING Cat 40 : 5.87 ms - 0.59 ns (11.09 s / 10000000), min: 0.00 ns, max: 2.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 0.00 ns - 0.00 ns (8.36 s / 10000000), min: 0.00 ns, max: 5.00 ms, nesting: 1 - 10000000

Honza

Looking at it, it seems like there is something fishy about times... maybe something with TIMING is now broken? Or perhaps it does bot behave well now in linux?

Subject: Re: String::Cat optimization
Posted by dolik.rce on Thu, 01 Dec 2011 11:16:17 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 01 December 2011 11:37Looking at it, it seems like there is something fishy

about times... maybe something with TIMING is now broken? Or perhaps it does bot behave well now in linux? I think there is something wrong with the TIMING macros on Linux. I get weird numbers from time to time, like 0.0ns calls above...

Honza

Subject: Re: String::Cat optimization
Posted by unodgs on Thu, 01 Dec 2011 13:38:06 GMT
View Forum Message <> Reply to Message

Quote:
Looking at it, it seems like there is something fishy about times... maybe something with TIMING is now broken? Or perhaps it does bot behave well now in linux?

Not only in linux. In Windows GetTickCount is used which resolution is in range 10 - 16 ms only.

Subject: Re: String::Cat optimization
Posted by Tom1 on Thu, 01 Dec 2011 13:50:29 GMT
View Forum Message <> Reply to Message

Hi,

I played around with Mirek's idea awhile and according to my simple '::GetTickCount()' benchmarking on MSC9/Win7x64 I managed to squeeze yet more performance out of it. The test covered all transfer lengths from 1 to 16 bytes.

The svo_memcpy() suffers a performance penalty at len==16, where secondary function call to memcpy steps in. The following macro approach helps dramatically to reduce that penalty. I also discovered that the memcpy() performance might not be reached systematically at transfer lengths above 11 bytes, so limiting the switch to <= 11 bytes should improve overall performance.

```
 inline void memcpy11i(char *t, const char *s, int len){
  switch(len) {
   case 11: t[10] = s[10];
   case 10: t[9] = s[9];
   case  9: t[8] = s[8];
   case  8: t[7] = s[7];
   case  7: t[6] = s[6];
   case  6: t[5] = s[5];
   case  5: t[4] = s[4];
   case  4: t[3] = s[3];
   case  3: t[2] = s[2];
   case  2: t[1] = s[1];
```

```
  case  1: t[0] = s[0];
 }
}
```

```
#define memcpy11(t, s, len) (len)>11 ? memcpy(t, s, len) : memcpy11i(t, s, len)
```

How does this perform on your systems?

Best regards,

Tom

Subject: Re: String::Cat optimization
Posted by Tom1 on Thu, 01 Dec 2011 13:54:24 GMT
View Forum Message <> Reply to Message

Uno,

It is true that GetTickCount() runs by default on a 10/16 ms resolution only. However, that does not affect the results too much if your measurement period is on the order of one second or more. (I used 100000000 repetitions to get around this resolution issue.)

Best regards,

Tom

Subject: Re: String::Cat optimization
Posted by mirek on Thu, 01 Dec 2011 16:51:22 GMT
View Forum Message <> Reply to Message

unodgs wrote on Thu, 01 December 2011 08:38Quote:
Looking at it, it seems like there is something fishy about times... maybe something with TIMING is now broken? Or perhaps it does bot behave well now in linux?

Not only in linux. In Windows GetTickCount is used which resolution is in range 10 - 16 ms only.

That is actually OK, as long as the number of passes is big enough... (it statistically averages out, so with 10ms resolution you can successfully measure ns times - funny, is not it?

Subject: Re: String::Cat optimization
Posted by mirek on Thu, 01 Dec 2011 16:52:45 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Thu, 01 December 2011 08:50Hi,

I played around with Mirek's idea awhile and according to my simple '::GetTickCount()' benchmarking on MSC9/Win7x64 I managed to squeeze yet more performance out of it. The test covered all transfer lengths from 1 to 16 bytes.

The svo_memcpy() suffers a performance penalty at len==16, where secondary function call to memcpy steps in. The following macro approach helps dramatically to reduce that penalty. I also discovered that the memcpy() performance might not be reached systematically at transfer lengths above 11 bytes, so limiting the switch to <= 11 bytes should improve overall performance.

```
 inline void memcpy11i(char *t, const char *s, int len){
  switch(len) {
   case 11: t[10] = s[10];
   case 10: t[9] = s[9];
   case  9: t[8] = s[8];
   case  8: t[7] = s[7];
   case  7: t[6] = s[6];
   case  6: t[5] = s[5];
   case  5: t[4] = s[4];
   case  4: t[3] = s[3];
   case  3: t[2] = s[2];
   case  2: t[1] = s[1];
   case  1: t[0] = s[0];
  }
 }

#define memcpy11(t, s, len) (len)>11 ? memcpy(t, s, len) : memcpy11i(t, s, len)
```

How does this perform on your systems?

Best regards,

Tom

Well, that is actually even better, as MSC refuses to inline that function....

Mirek

Tom1 wrote on Thu, 01 December 2011 08:50Hi,

I played around with Mirek's idea awhile and according to my simple '::GetTickCount()' benchmarking on MSC9/Win7x64 I managed to squeeze yet more performance out of it. The test covered all transfer lengths from 1 to 16 bytes.

The svo_memcpy() suffers a performance penalty at len==16, where secondary function call to memcpy steps in. The following macro approach helps dramatically to reduce that penalty. I also discovered that the memcpy() performance might not be reached systematically at transfer lengths above 11 bytes, so limiting the switch to <= 11 bytes should improve overall performance.

```
 inline void memcpy11i(char *t, const char *s, int len){
  switch(len) {
   case 11: t[10] = s[10];
   case 10: t[9] = s[9];
   case  9: t[8] = s[8];
   case  8: t[7] = s[7];
   case  7: t[6] = s[6];
   case  6: t[5] = s[5];
   case  5: t[4] = s[4];
   case  4: t[3] = s[3];
   case  3: t[2] = s[2];
   case  2: t[1] = s[1];
   case  1: t[0] = s[0];
  }
 }

#define memcpy11(t, s, len) (len)>11 ? memcpy(t, s, len) : memcpy11i(t, s, len)
```

How does this perform on your systems?

Best regards,

Tom

Thanks, this is even better. Somehow I forgot that if compiler decides not to inline something, I can still force it by macro.

So, I am following your advice, using macro and limit:

```
#define SVO_MEMCPY(tgt, src, len) \
do { \
 const char *s__ = (const char *)(src); \
 char *t__ = (char *)(tgt); \
 switch(len) { \
 case 11: t__[10] = s__[10]; \
 case 10: t__[9] = s__[9]; \
 case  9: t__[8] = s__[8]; \
 case  8: t__[7] = s__[7]; \
 case  7: t__[6] = s__[6]; \
 case  6: t__[5] = s__[5]; \
 case  5: t__[4] = s__[4]; \
 case  4: t__[3] = s__[3]; \
 case  3: t__[2] = s__[2]; \
 case  2: t__[1] = s__[1]; \
 case  1: t__[0] = s__[0]; \
  break; \
 default: \
  memcpy(t__, s__, len); \
 } \
} while(false)
```

memcpy

TIMING Cat 40 : 1.98 s - 198.34 ns (2.15 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 599.44 ms - 59.94 ns (767.00 ms / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

inline svo_memcpy

TIMING Cat 40 : 1.45 s - 145.38 ns (1.63 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 491.79 ms - 49.18 ns (671.00 ms / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

macro

TIMING Cat 40 : 1.04 s - 103.71 ns (1.22 s / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000
TIMING Cat 18 : 302.09 ms - 30.21 ns (486.00 ms / 10000000), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 10000000

Amazing

It is interesting how year after year, we can still squeeze a bit from String....

Mirek

Subject: Re: String::Cat optimization
Posted by Lance on Fri, 02 Dec 2011 02:59:39 GMT
View Forum Message <> Reply to Message

Impressive.

Depending on how likely 0 length memory is "copied", it may be desirable to add a branch to handle it.

```
case  1: t__[0] = s__[0]; \
      case  0: \
 break; \
```

Tom's code takes care of 0 length string without memcpy, just as above.

Subject: Re: String::Cat optimization
Posted by Novo on Fri, 02 Dec 2011 05:02:26 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 01 December 2011 14:41
Thanks, this is even better. Somehow I forgot that if compiler decides not to inline something, I can still force it by macro.

There is an easier way to force inlining:

MSVC - __forceinline
GCC - __attribute__((always_inline))

Subject: Re: String::Cat optimization
Posted by mirek on Fri, 02 Dec 2011 05:39:41 GMT
View Forum Message <> Reply to Message

Lance wrote on Thu, 01 December 2011 21:59Impressive.

Depending on how likely 0 length memory is "copied", it may be desirable to add a branch to handle it.

```
 case  1: t__[0] = s__[0]; \
       case  0: \
  break; \
```

Tom's code takes care of 0 length string without memcpy, just as above.

Yes. Thanks.

Mirek

Subject: Re: String::Cat optimization
Posted by mirek on Fri, 02 Dec 2011 05:40:07 GMT
View Forum Message <> Reply to Message

Novo wrote on Fri, 02 December 2011 00:02mirek wrote on Thu, 01 December 2011 14:41
Thanks, this is even better. Somehow I forgot that if compiler decides not to inline something, I can still force it by macro.

There is an easier way to force inlining:

MSVC - __forceinline
GCC - __attribute__((always_inline))

Is it really easier?

Mirek

Subject: Re: String::Cat optimization
Posted by Tom1 on Fri, 02 Dec 2011 09:22:41 GMT
View Forum Message <> Reply to Message

OK, this was fun!

Just makes me wonder if there could be wider visibility for this optimization. I can imagine memcpy() is used all over the code, and short blocks are not that uncommon. Maybe Core could

hold this SVO_MEMCPY macro in a header included via Core.h so that any code can access it?

Best regards,

Tom

Subject: Re: String::Cat optimization
Posted by mirek on Fri, 02 Dec 2011 10:10:03 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Fri, 02 December 2011 04:22OK, this was fun!

Just makes me wonder if there could be wider visibility for this optimization. I can imagine memcpy() is used all over the code, and short blocks are not that uncommon. Maybe Core could hold this SVO_MEMCPY macro in a header included via Core.h so that any code can access it?

Best regards,

Tom

Maybe, but I am afraid it would take some time to figure out what to convert to svo_memcpy and what not. Not all cases are suitable.

For now I will just file new RM#:

http://www.ultimatepp.org/redmine/issues/204

Subject: Re: String::Cat optimization
Posted by Novo on Fri, 02 Dec 2011 17:50:53 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 02 December 2011 00:40Novo wrote on Fri, 02 December 2011 00:02mirek wrote on Thu, 01 December 2011 14:41
Thanks, this is even better. Somehow I forgot that if compiler decides not to inline something, I can still force it by macro.

There is an easier way to force inlining:

MSVC - __forceinline
GCC - __attribute__((always_inline))

Is it really easier?

Mirek

Make one define (something like FORCE_INLINE) and put it in front of any function you want to make inline. This is it. You do not need to deal with hundreds of back-slashes.

Subject: Re: String::Cat optimization
Posted by mirek on Fri, 02 Dec 2011 18:06:06 GMT
View Forum Message <> Reply to Message

Novo wrote on Fri, 02 December 2011 12:50mirek wrote on Fri, 02 December 2011 00:40Novo wrote on Fri, 02 December 2011 00:02mirek wrote on Thu, 01 December 2011 14:41
Thanks, this is even better. Somehow I forgot that if compiler decides not to inline something, I can still force it by macro.

There is an easier way to force inlining:

MSVC - __forceinline
GCC - __attribute__((always_inline))

Is it really easier?

Mirek

Make one define (something like FORCE_INLINE) and put it in front of any function you want to make inline. This is it. You do not need to deal with hundreds of back-slashes.

Actually, I already did, see 'strlen optimization' thread...

Subject: Re: String::Cat optimization
Posted by mirek on Wed, 16 May 2012 13:30:55 GMT
View Forum Message <> Reply to Message

Moved SVO_MEMCPY to defs.h and added SVO_MEMSET... keeping it as macro.
