Subject: Optimizing svo_memeq -- just for curiosity
Posted by Tom1 on Mon, 03 Mar 2014 15:32:12 GMT
View Forum Message <> Reply to Message

Hi,

I tinkered a bit with svo_memeq (svo_memeq_t below) and found that simplifying the code a bit may improve performance dramatically when compiled with MSC9/MSC10 Speed -build mode:

```
template <class tchar>
force_inline bool svo_memeq_t(const tchar *a, const tchar *b, int len){
 return !len-- ? true : *a++!=*b++ ? false : svo_memeq_t(a,b,len);
}
```

Short lengths can get an about five or six fold improvement and bigger lengths (over 12) are even better. (Anyway, this what I found on Windows on an Intel processor.)

OK, this is recursive and stack can't handle unlimited comparison lengths, so this can't replace the original code as is. So, this is just for those interested in how compilers' optimization work.

Best regards,

Tom

Subject: Re: Optimizing svo_memeq -- just for curiosity
Posted by mirek on Tue, 04 Mar 2014 07:03:02 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Mon, 03 March 2014 10:32Hi,

I tinkered a bit with svo_memeq (svo_memeq_t below) and found that simplifying the code a bit may improve performance dramatically when compiled with MSC9/MSC10 Speed -build mode:

```
template <class tchar>
force_inline bool svo_memeq_t(const tchar *a, const tchar *b, int len){
 return !len-- ? true : *a++!=*b++ ? false : svo_memeq_t(a,b,len);
}
```

Short lengths can get an about five or six fold improvement and bigger lengths (over 12) are even better. (Anyway, this what I found on Windows on an Intel processor.)

OK, this is recursive and stack can't handle unlimited comparison lengths, so this can't replace the original code as is. So, this is just for those interested in how compilers' optimization work.

Best regards,

Tom

Well, I just could not stop thinking about String::Find(String)... and got some new ideas how to optimize it even more. The key information is that with intel CPUs since about 2010 (and AMD from the same era), unaligned memory access does not have performance penalty anymore (and before that, penalty is not that high, just say 50%).

Which means that with x86-64, you can compare unaligned data up to 16 bytes with just two compares...

Mirek
