Subject: Re: BufferPainter::Clear() optimization
Posted by mirek on Fri, 22 May 2020 11:01:25 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Fri, 22 May 2020 11:59mirek wrote on Fri, 22 May 2020 12:39
In the trunk now... >=16 now handled by non-inline function. There is impact in your benchmark (the one that runs for all sizes), less impact in my benchmark (with ransom sizes), but I think this is the right move...

It looks like >32 might be better in this case... Not sure though.

BR, Tom

It in turn makes inlined part bigger.... I would rather be careful there.

OK, for what is worth, I have tried with AVX and I do not see any improvement. Here is the code (for CLANG):

```
__attribute__((target ("avx")))
never_inline
void memsetd_l2(dword *t, dword data, size_t len)
{
 __m128i val4 = _mm_set1_epi32(data);
 __m256i val8 = _mm256_set1_epi32(data);
 auto Set4 = [&](size_t at) { _mm_storeu_si128((__m128i *)(t + at), val4); };
 #define Set8(at) _mm256_storeu_si256((__m256i *)(t + at), val8);
 Set4(len - 4); // fill tail
 if(len >= 32) {
  if(len >= 1024*1024) { // for really huge data, bypass the cache
   huge_memsetd(t, data, len);
   return;
  }
  Set8(0); // align up on 16 bytes boundary
  const dword *e = t + len;
  t = (dword *)(((uintptr_t)t | 31) + 1);
  len = e - t;
  e -= 32;
  while(t <= e) {
   Set8(0); Set8(8); Set8(16); Set8(24);
   t += 32;
  }
 }
 if(len & 16) {
  Set8(0); Set8(8);
  t += 16;
 }
 if(len & 8) {
```

```
  Set8(0);
  t += 8;
 }
 if(len & 4)
  Set4(0);
}

inline
void FillX(void *p, dword data, size_t len)
{
 dword *t = (dword *)p;
 if(len < 4) {
  if(len & 2) {
   t[0] = t[1] = t[len - 1] = data;
   return;
  }
  if(len & 1)
   t[0] = data;
  return;
 }

 if(len >= 16) {
  memsetd_l2(t, data, len);
  return;
 }

 __m128i val4 = _mm_set1_epi32(data);
 auto Set4 = [&](size_t at) { _mm_storeu_si128((__m128i *)(t + at), val4); };
 Set4(len - 4); // fill tail
 if(len & 8) {
  Set4(0); Set4(4);
  t += 8;
 }
 if(len & 4)
  Set4(0);
}
```

Frankly I am sort of happy, because GCC/CLANG way of dealing with AVX is really stupid: It declines AVX instrinics, unless you compile whole function for AVX code, but then it starts generating AVX opcodes everywhere and the funciton does not run on non-AVX CPUs anymore.
