
Subject: Re: BufferPainter::Clear() optimization
Posted by [mirek](#) on Sun, 31 May 2020 22:39:13 GMT
[View Forum Message](#) <> [Reply to Message](#)

While optimizing memcpy and memset, I have tried a new look at other things, like String comparison and memhash. I think I improved String::operator== a tiny bit and now I am working on memhash function. Decided to introduce "hash_t" and to have hash value 64 bit when CPU_64.

After bit of experimenting, I have found these functions (one for 64 bit, other 32 bit) work best:

```
never_inline
uint64 memhash64(const void *ptr, int len)
{
    const byte *s = (byte *)ptr;
    uint64 val = HASH64_CONST1;
    if(len >= 8) {
        if(len >= 32) {
            uint64 val1, val2, val3, val4;
            val1 = val2 = val3 = val4 = HASH64_CONST1;
            while(len >= 32) {
                val1 = HASH64_CONST2 * val1 + *(qword *)s;
                val2 = HASH64_CONST2 * val2 + *(qword *)s + 8;
                val3 = HASH64_CONST2 * val3 + *(qword *)s + 16;
                val4 = HASH64_CONST2 * val4 + *(qword *)s + 24;
                s += 32;
                len -= 32;
            }
            val = HASH64_CONST2 * val + val1;
            val = HASH64_CONST2 * val + val2;
            val = HASH64_CONST2 * val + val3;
            val = HASH64_CONST2 * val + val4;
        }
        const byte *e = s + len - 8;
        while(s < e) {
            val = HASH64_CONST2 * val + *(qword *)s;
            s += 8;
        }
        return HASH64_CONST2 * val + *(qword *)e;
    }
    if(len > 4) {
        val = HASH64_CONST2 * val + *(dword *)s;
        val = HASH64_CONST2 * val + *(dword *)s + len - 4;
        return val;
    }
    if(len >= 2) {
        val = HASH64_CONST2 * val + *(word *)s;
```

```

    val = HASH64_CONST2 * val + *(word *)(s + len - 2);
    return val;
}
return len ? HASH64_CONST2 * val + *s : val;
}

```

```

never_inline
uint64 memhash32(const void *ptr, int len)
{
    const byte *s = (byte *)ptr;
    uint64 val = HASH32_CONST1;
    if(len >= 4) {
        if(len >= 16) {
            uint64 val1, val2, val3, val4;
            val1 = val2 = val3 = val4 = HASH32_CONST1;
            while(len >= 32) {
                val1 = HASH32_CONST2 * val1 + *(dword *)(s);
                val2 = HASH32_CONST2 * val2 + *(dword *)(s + 4);
                val3 = HASH32_CONST2 * val3 + *(dword *)(s + 8);
                val4 = HASH32_CONST2 * val4 + *(dword *)(s + 12);
                s += 16;
                len -= 16;
            }
            val = HASH32_CONST2 * val + val1;
            val = HASH32_CONST2 * val + val2;
            val = HASH32_CONST2 * val + val3;
            val = HASH32_CONST2 * val + val4;
        }
        const byte *e = s + len - 4;
        while(s < e) {
            val = HASH32_CONST2 * val + *(dword *)(s);
            s += 4;
        }
        return HASH32_CONST2 * val + *(dword *)(e);
    }
    if(len >= 2) {
        val = HASH32_CONST2 * val + *(word *)(s);
        val = HASH32_CONST2 * val + *(word *)(s + len - 2);
        return val;
    }
    return len ? HASH32_CONST2 * val + *s : val;
}

```

While other "mem*" functions are easy to write tests for, hasing is a bit more complicated; can I request some code review here? Basically, I think combination functions are OK, but I would like to be sure it reads exactly len bytes from memory (it is ok if some are read twice...).

