Optimizing GELI Performance

John-Mark Gurney
jmg@funkthat.com
@encthenet

BSDcan 2014
Why improve it?

- Even w/ 6 cores, ZFS+GELI was slow
- AES-NI did not significantly improve things
  - GELI is using AES-XTS, which should get >1GB/sec, but only got <150MB/sec (in userland testing)
- If it's slow, people won't use it
- Make it more maintainable
What is AES?

- **Initial Round**
  - AddRoundKey

- **Rounds**
  - SubBytes
  - ShiftRows
  - MixColumns
  - AddRoundKey

- **Final Round**
  - SubBytes
  - ShiftRows
  - AddRoundKey
Setting the correct options

- Cipher Mode
- Sector Size
- Key Size (sets number of rounds)
OpenSSL Baseline Performance

Make sure you specify -evp to openssl speed to get AES-NI
Cipher Mode could be the issue

Cipher Block Chaining (CBC) mode encryption

XEX with tweak and ciphertext stealing (XTS) mode encryption
Counter Cipher Mode

Counter (CTR) mode encryption
Instruction Pipelining

- AES has 10, 12 or 14 rounds depending upon key size
- Each round depends upon previous round
- Multiple blocks can (if mode supports it) be done at the same time
Improving the Tweak Factor

Original code:

```c
uint8_t *tweak; /* parameter */
/* Exponentiate tweak. */
carry_in = 0;
for (i = 0; i < AES_XTS_BLOCKSIZE; i++) {
    carry_out = tweak[i] & 0x80;
    tweak[i] = (tweak[i] << 1) | (carry_in ? 1 : 0);
    carry_in = carry_out;
}
if (carry_in)
    tweak[0] ^= AES_XTS_ALPHA;
```
Improving the Tweak Factor

pjd's improved code:

```c
uint64_t *tweak; /* parameter */
/* Exponentiate tweak. */
carry = ((tweak[0] & 0x8000000000000000ULL) > 0);
tweak[0] <<= 1;
if (tweak[1] & 0x8000000000000000ULL) {
    uint8_t *twk = (uint8_t *)tweak;
    twk[0] ^= AES_XTS_ALPHA;
}
tweak[1] <<= 1;
if (carry)
    tweak[1] |= 1;
```
Improving the Tweak Factor

Current code:

```c
__m128i inp, ret;
const __m128i alphamask = _mm_set_epi32(1, 1, 1, AES_XTS_ALPHA);
__m128i xtweak, ret;
/* set up xor mask */
xtweak = _mm_shuffle_epi32(inp, 0x93);
xtweak = _mm_srai_epi32(xtweak, 31);
xtweak &= alphamask;
/* next term */
ret = _mm_slli_epi32(inp, 1);
ret ^= xtweak;
```
### Tweak factor of AES-XTS

<table>
<thead>
<tr>
<th></th>
<th>127-96</th>
<th>95-64</th>
<th>63-32</th>
<th>31-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>shuffle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>srai 31</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>and amask</td>
<td>0x87</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slli 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xor</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Intrinsics vs Assembly

• Intrinsics Pros
  – Works around ABI limitations
  – Inline functions
  – Single source for i386 and amd64
  – Maintainability

• Assembly Pros
  – Tighter instruction scheduling
  – Easier non-aligned load control
  – gcc did not support AES-NI intrinsics
Adding AES-NI to toolchain

- clang did not need any work
- Our gcc/binutils is very old (tell me something I don't know)
- Added support to binutils for assembling AES-NI instructions
- gcc needed new headers
- PCLMULQDQD (carry-less multiple) added for completeness
Original Assembly

```
movdqu (%rdx),%xmm0
cmpq $0,%r8
je 1f
movdqu (%r8),%xmm1 /* unaligned load into reg */
pxor %xmm1,%xmm0 /* pxor otherwise can fault on iv */
1: pxor (%rsi),%xmm0
2: addq $0x10,%rsi
// aesenc (%rsi),%xmm0
.byte 0x66,0x0f,0x38,0xdc,0x06
decl %edi
jne 2b
addq $0x10,%rsi
// aesenclast (%rsi),%xmm0
.byte 0x66,0x0f,0x38,0xdd,0x06
movdqu %xmm0,(%rcx)
```
Intrinsics

- Provides a 128 bit data type (\_m128i and others)
- Implements functions as either a built-in or asm directive in header files
- Features must be enabled via compiler flags
- Not easy to handle unaligned data – use either:
  - explicit \_mm\_loadu\_si128 call
  - access through a packed struct
static inline __m128i aesni_enc(int rounds, const __m128i *keysched, const __m128i from)
{
    __m128i tmp;
    int i;

    tmp = from ^ keysched[0];
    for (i = 0; i < rounds; i++)
    {
        tmp = _mm_aesenc_si128(tmp, keysched[i + 1]);
    }
    return _mm_aesenclast_si128(tmp, keysched[i + 1]);
}
Adding to kernel compile

- Special rule in sys/conf/files.{amd64,i386} to enable AES-NI

    aesni_wrap.o  optional aesni  \n    compile-with  "${CC} -c  \n    ${CFLAGS:C/^-O2$/-O3/:N-nostdinc}  \n    ${WERROR} ${PROF}  \n    -mmmx -msse -maes ${.IMPSRC}"  \n
## Performance Testing

- ministat

### Statistics

<table>
<thead>
<tr>
<th>N</th>
<th>Min</th>
<th>Max</th>
<th>Median</th>
<th>Avg</th>
<th>Stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>57.1846125e+08</td>
<td>7.2917070e+08</td>
<td>7.2324988e+08</td>
<td>7.2420199e+08</td>
<td>4489811.8</td>
</tr>
<tr>
<td></td>
<td>55.4985213e+08</td>
<td>5.5033968e+08</td>
<td>5.5013242e+08</td>
<td>5.5012183e+08</td>
<td>196393.65</td>
</tr>
</tbody>
</table>

Difference at 95.0% confidence:

-1.7408e+08 +/- 4.63466e+06

-24.0375% +/- 0.639967%

(Student's t, pooled s = 3.17781e+06)

### Software Statistics

<table>
<thead>
<tr>
<th>N</th>
<th>Min</th>
<th>Max</th>
<th>Median</th>
<th>Avg</th>
<th>Stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>55667911</td>
<td>63067447</td>
<td>60749919</td>
<td>60422098</td>
<td>3022659.1</td>
</tr>
</tbody>
</table>

Difference at 95.0% confidence:

-6.6378e+08 +/- 5.58175e+06

-91.6567% +/- 0.770745%

(Student's t, pooled s = 3.8272e+06)
Using pmcstat

- Generate performance data:
  
  ```bash
  pmcstat -P BU_CPU_CLK_UNHALTED -o pmcstat.out -g ./test perf
  ```

- gprof format output cannot handle large counts

- calltree (kcachegrind) format to the rescue

  ```bash
  pmcstat -R pmcstat.out -F output.calltree
  ```
Performance Testing

- Call graph showed unexpected thread crypto_ret_proc consuming ~15% cpu
- Thread runs to deliver callbacks, if AES-NI driver is marked CRYPTOCAP_F_SYNC, callbacks are not deferred to thread, resulting in ~27% performance increase
GELI on gzero performance

- debug.witness.watch=0
- kern.geom.eli.threads=1
- kern.geom.zero.clear=0
- ~900MB/sec on AMD A10-5700 3.4GHz
Continued Improvement

- Only calls through OpenCrypto framework are improved, direct calls are not
  - Handling FPU state in non-sleepable contexts
- Better memory allocation, avoid large allocs
- Pipeline key schedule – AES-XTS needs two
- AES-GCM – Work ongoing and supported by the FreeBSD Foundation and Netgate
- Working on SHA256 (for ZFS)
Questions?