Fast Software Encryption with SIMD
How to speed up symmetric block ciphers with the AVX/AVX2 instruction set

Johannes Götzfried, Tilo Müller

Department of Computer Science
Friedrich-Alexander University of Erlangen-Nuremberg

April 14, 2013
Outline

1. Introduction

2. Background
   - Symmetric Ciphers
   - Advanced Vector Extensions
   - Linux Kernel

3. Implementation
   - Generic Approach
   - Example: Twofish

4. Evaluation

5. Conclusion and Outlook
Encryption is important in today’s IT-Security
- Network communication protocols (e.g. HTTP/SSL, VPNs and WiFi)
- Disk encryption

Encryption techniques are often mandatory
- Remote connections for controlling machines
- Online banking
- Employees, that work outside their office or travel a lot

Performance
- Encryption involves necessarily a performance drawback
- Low-level implementations can achieve a gain in performance
- AESNI only usable for AES but not for different ciphers
Outline

1. Introduction

2. Background
   - Symmetric Ciphers
   - Advanced Vector Extensions
   - Linux Kernel

3. Implementation
   - Generic Approach
   - Example: Twofish

4. Evaluation

5. Conclusion and Outlook
Outline

1 Introduction

2 Background
   - Symmetric Ciphers
   - Advanced Vector Extensions
   - Linux Kernel

3 Implementation
   - Generic Approach
   - Example: Twofish

4 Evaluation

5 Conclusion and Outlook
Symmetric Ciphers

- Block Ciphers: Serpent, Twofish, Blowfish, Cast-128, Cast-256
- Modes of operation for block ciphers
  - ECB, CBC, CTR, LRW, XTS
  - Suitable for parallelization (except CBC encryption mode)
Properties of the ciphers

- Encryption and decryption routines are composed of similar rounds
- Key sizes between 64 and 512 bits
- Block sizes of 64 or 128 bits
- Between 12 and 48 rounds
- Common operations: substitutions, permutations and key mixing
- Operations are usually performed on doublewords (i.e. 32 bits)
Outline

1. Introduction

2. Background
   - Symmetric Ciphers
   - Advanced Vector Extensions
   - Linux Kernel

3. Implementation
   - Generic Approach
   - Example: Twofish

4. Evaluation

5. Conclusion and Outlook
SIMD vs. scalar operations
SIMD ≡ Single Instruction Multiple Data

AVX Support
- Intel Sandy and Ivy Bridge CPUs
- AMD Bulldozer CPUs
- GCC supports AVX at least since version 4.6
- Linux kernel since version 2.6.30
AVX Registers

- 256 bit wide SIMD registers YMM0 to YMM7 or YMM15
- Lower 128 bits correspond to the XMM registers known from SSE
- Different interpretations of the stored data possible:

  SSE and AVX 128-bit types
  - 4x float
  - 2x double
  - 16x byte
  - 8x 16-bit word
  - 4x 32-bit doubleword
  - 2x 64-bit quadword
  - 1x 128-bit doublequadword

  AVX 256-bit types
  - 8x float
  - 4x double

Drawback

Integer types only available with 128 bit XMM registers
### AVX Instruction Set

#### Non-destructive three operand syntax

<table>
<thead>
<tr>
<th>SSE</th>
<th>paddd</th>
<th>%xmm1, %xmm2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVX</td>
<td>vpaddd</td>
<td>%xmm1, %xmm2, %xmm3</td>
</tr>
</tbody>
</table>

#### Suffixes

- b, w, d, q, dq

#### Instructions

- **Movement**
  - vmovdqa, vmovdqu, vbroadcastss,
  - vmovd, vpexstride, vpinsrd

- **Arithmetic**
  - vpadd, vpsubd

- **Logical**
  - vpand, vpandn, vpor, vpxor

- **Shift**
  - vpslld, vpsrld, vpsllldq, vpsrlldq

- **Shuffle and Pack**
  - vpshufd, vpunpckhqdq, vpunpckldq
AVX2 Support

- Haswell microarchitecture (launching market 2013)
- GCC supports AVX2 since version 4.7
- Testing: Intel Software Development Emulator (SDE)

AVX2 Features

- Integer instructions are able to work with 256 bit YMM registers
- Lane concept (in-lane vs. cross-lane instructions)
- New instructions (e.g. vpbroadcastd, vbroadcasti128)

Gather Operation

```
vpcmpeqd %ymm15, %ymm15, %ymm15
vpgatherdd %ymm15, 16(%rsi, %ymm1, 4), %ymm0
```

Addresses:  
\[
%rsi + %ymm1[32*i+31:32*i]*4 + 16 \quad \text{with} \quad i = 0 \ldots 7
\]
Outline

1. Introduction

2. Background
   - Symmetric Ciphers
   - Advanced Vector Extensions
   - Linux Kernel

3. Implementation
   - Generic Approach
   - Example: Twofish

4. Evaluation

5. Conclusion and Outlook
Cryptographic API

- Five types of transformations
  - AEAD, block ciphers, ciphers, compressors and hashes
- Synchronous and asynchronous interface
- Different Layers of abstraction
  (e.g. mode of operation independent of block cipher)
- Test module for verification and benchmarks (tcrypt)
- No stable API and bad documentation

Break with the design of the crypto API

Modes of operation have to be reimplemented
⇒ allow block ciphers processing blocks in parallel
Outline

1. Introduction
2. Background
   - Symmetric Ciphers
   - Advanced Vector Extensions
   - Linux Kernel
3. Implementation
   - Generic Approach
   - Example: Twofish
4. Evaluation
5. Conclusion and Outlook
AVX Approach

Considerations
- Leave key schedule untouched
- Focus on block size of 128 bits and *encryption* routine

AVX Approach (simplified)
1. Fetch input blocks from memory (two 4-block chunks, e.g. 8 blocks)
2. 4x4 matrix transposition of doublewords with unpack operations
3. Replace arithmetic and logical operations with SIMD equivalent
4. Apply inverse transposition and write output blocks back to memory
AVX2 Approach

AVX Limitations
- Complex algebraic operations (e.g. multiplication over GF($2^8$))
- Table lookups involve GPR ↔ SIMD-Register transitions

AVX2 Approach
1. Fetch input blocks from memory (two 8-block chunks, e.g. 16 blocks)
2. Two 4x4 matrix transpositions with the same number of operations
3. Replace arithmetic and logical operations with AVX2 equivalent
4. Apply inverse transposition and write output blocks back to memory

AVX2 Improvements
- Implement table lookups using the *gather*-Operation (8x32 tables)
- Data preparation: packed logical right shifts and respective bitmasks
- Data never leaves the SIMD register
Kernel Integration

- Makes the implementations usable for disk encryption
- Registration together with modes of operations
- For each mode a block cipher is registered (e.g. \texttt{cbc(twofish)}, \texttt{ecb(serpent)})
- Our ciphers are registered with a higher priority
- Provided as loadable kernel modules with own entry in \texttt{Kconfig}
Outline

1. Introduction
2. Background
   - Symmetric Ciphers
   - Advanced Vector Extensions
   - Linux Kernel
3. Implementation
   - Generic Approach
   - Example: Twofish
4. Evaluation
5. Conclusion and Outlook
Twofish

- Third best rated finalist in the AES Competition
- Feistel network
- Block size of 128 bits
- Key sizes of 128, 192 or 256 bits
- 16 rounds independent of the keysize
- Four key-dependent 8x8 S-boxes
- Key whitening
Reading and Transforming Input Blocks
AVX Implementation for 128 bit Block Ciphers

```c
#define transpose_4x4(x0, x1, x2, x3, t0, t1, t2) \
   vpunpckldq x1, x0, t0; \ 
   vpunpckhqdq x1, x0, t2; \ 
   vpunpckldq x3, x2, t1; \ 
   vpunpckhqdq x3, x2, x3; \ 
   vpunpcklqdq t1, t0, x0; \ 
   vpunpckhqdq t1, t0, x1; \ 
   vpunpcklqdq x3, t2, x2; \ 
   vpunpckhqdq x3, t2, x3;

#define read_blocks(in, x0, x1, x2, x3, t0, t1, t2) \
   vmovdqu (0*4*4)(in), x0; \ 
   vmovdqu (1*4*4)(in), x1; \ 
   vmovdqu (2*4*4)(in), x2; \ 
   vmovdqu (3*4*4)(in), x3; \ 
   transpose_4x4(x0, x1, x2, x3, t0, t1, t2)

leaq (4*4*4)(%rdx), %rax;
read_blocks(%rdx, RA1, RB1, RC1, RD1, RK0, RK1, RK2);
read_blocks(%rax, RA2, RB2, RC2, RD2, RK0, RK1, RK2);
```
Twofish Table Lookup (1)
AVX Implementation of Twofish

```c
#define G(a, x, t0, t1, t2, t3) \
    vmovq a, RGI1; \
    vpsrlq $8, a, x; \
    vmovq x, RGI2; \
\ 
    lop(t0, t1, t2, t3, RGI1, RGS1); \
    shrq $16, RGI1; \
    lop(t0, t1, t2, t3, RGI1, RGS2); \
    shlq $32, RGS2; \
    orq RGS1, RGS2; \
\ 
    lop(t0, t1, t2, t3, RGI2, RGS1); \
    shrq $16, RGI2; \
    lop(t0, t1, t2, t3, RGI2, RGS3); \
    shlq $32, RGS3; \
    orq RGS1, RGS3; \
\ 
    vmovq RGS2, x; \
    vpinsrq $1, RGS3, x, x;
```
#define lop(t0, t1, t2, t3, src, dst) \  
movb src ## bl, RID1b; \  
movb src ## bh, RID2b; \  
movl t0(CTX, RID1, 4), dst ## d; \  
xorl t1(CTX, RID2, 4), dst ## d; \  
shrq $16$, src; \  
movb src ## bl, RID1b; \  
movb src ## bh, RID2b; \  
xorl t2(CTX, RID1, 4), dst ## d; \  
xorl t3(CTX, RID2, 4), dst ## d;
#define G(a, x, t0, t1, t2, t3) \
    vpand RLOW, a, RIDX; \ 
    vpcmpeqd RFULL, RFULL, RFULL; \ 
    vpgatherdd RFULL, t0(CTX, RIDX, 4), x; \ 
    vpsrld $8, a, RIDX; \ 
    vpand RLOW, RIDX, RIDX; \ 
    vpcmpeqd RFULL, RFULL, RFULL; \ 
    vpgatherdd RFULL, t1(CTX, RIDX, 4), RIDX; \ 
    vpxor RIDX, x, x; \ 
    vpsrld $16, a, RIDX; \ 
    vpand RLOW, RIDX, RIDX; \ 
    vpcmpeqd RFULL, RFULL, RFULL; \ 
    vpgatherdd RFULL, t2(CTX, RIDX, 4), RIDX; \ 
    vpxor RIDX, x, x; \ 
    vpsrld $24, a, RIDX; \ 
    vpcmpeqd RFULL, RFULL, RFULL; \ 
    vpgatherdd RFULL, t3(CTX, RIDX, 4), RIDX; \ 
    vpxor RIDX, x, x;
Outline

1. Introduction

2. Background
   - Symmetric Ciphers
   - Advanced Vector Extensions
   - Linux Kernel

3. Implementation
   - Generic Approach
   - Example: Twofish

4. Evaluation

5. Conclusion and Outlook
Measurements were taken on an Intel Core i5-2450M.
Achieved speedups with the AVX implementations:
- Serpent: 6.1%
- Twofish: 30.8%
- Blowfish: 0.8%
- Cast-128: 115.8%
- Cast-256: 88.6%

AVX2 implementations are suspected to be a lot faster.
## Twofish Instruction and Timing Results in Userspace

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Instructions</th>
<th>Time (s)</th>
<th>Speedup (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>generic</td>
<td>35913728</td>
<td>6.215</td>
<td>-</td>
</tr>
<tr>
<td>asm_64</td>
<td>28788575</td>
<td>5.800</td>
<td>7.15</td>
</tr>
<tr>
<td>asm_64-3way</td>
<td>34493255</td>
<td>4.714</td>
<td>23.03</td>
</tr>
<tr>
<td>avx</td>
<td>28622848</td>
<td>3.605</td>
<td>30.79</td>
</tr>
<tr>
<td>avx2</td>
<td>6426624</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Userspace Results**

- 3-way implementation provides significant speedup
- AVX implementation is another 30.8% faster
- AVX implementation needs less instructions than all other implementations
- AVX2 implementation decreases instruction count drastically
Results for Different Modes with Twofish in Kernelspace
256 bit key, 8192 input bytes

Kernelspace Results

- Speedup remains clearly visible with the different modes
- CBC encryption is as slow as with the 3-way implementation but not slower
Results of CBC Decryption for Twofish and AES
256 bit key

CBC Decryption Results

- Twofish implementations slower than AES AESNI implementation but faster than AES assembler implementation
- Speedup remains approximately constant with increasing input sizes
- Absolute speed of the AVX implementation is about 24 cycles per byte
### Twofish Disk Reading Speed Results

**Ramdisk (cbc-essiv:sha256)**

<table>
<thead>
<tr>
<th>Kernel Module</th>
<th>Disk Speed (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes-x86_64</td>
<td>318.68</td>
</tr>
<tr>
<td>aesni-intel</td>
<td>1055.75</td>
</tr>
<tr>
<td>twofish-generic</td>
<td>282.15</td>
</tr>
<tr>
<td>twofish-x86_64</td>
<td>314.98</td>
</tr>
<tr>
<td>twofish-x86_64-3way</td>
<td>390.15</td>
</tr>
<tr>
<td>twofish-avx-x86_64</td>
<td>467.49</td>
</tr>
</tbody>
</table>

### Disk Reading Results

- Dimensions remain the same with the device mapper dm-crypt
- Speedup should have practical impact on disk encryption applications
Conclusion

- Generic approach to speed up symmetric block ciphers
  - Parallel processing of sequenced input blocks
  - Particularly efficient in combination with modes of operation (e.g. ECB, CBC)
- AVX variants for five different ciphers
  - Taken from the Linux Crypto-API
  - Provided as open source kernel patches
  - Four of them have been submitted and merged into mainline
- Implementations with upcoming instruction set AVX2
  - Developed on an emulator
  - Will first run on CPUs launching market in 2013
- Performance Benchmarks
  - In user and kernel mode and for the case of disk encryption for AVX
  - Performance estimation of the AVX2 implementations
Further Development
- AVX implementations are in active development within kernel tree
- Even more performance gain by rearranging instructions (e.g. another 14% for Twofish)
- Better performance on AMD Bulldozer CPUs

AVX2 implementations
- Performance evaluation on real hardware
- Potential kernel integration

Speed up different algorithms
- Similar symmetric block ciphers
- Hash algorithms (SHA-3 finalists, SHA-1, SHA-2 or MD5)

Port implementations to different architecture
- AMD XOP with packed rotations
- ARM platform with NEON extensions
Thank you for your attention!

Further Information:
http://www1.cs.fau.de/avx.crypt