BLAKE SIMD past, present, future

Jean-Philippe Aumasson, NAGRA

Joint work with Samuel Neves, Uni Coimbra

SHA-3 \in

BLAKE
Groestl
JH
Keccak
Skein

http://www.nist.gov/hash-competition
http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo

Jean-Philippe Aumasson

University of Applied Sciences Northwestern Switzerland
School of Engineering

NAGRA KUDELSKI

Luca Henzen

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

UBS

Willi Meier

University of Applied Sciences Northwestern Switzerland
School of Engineering

Raphael C.-W. Phan

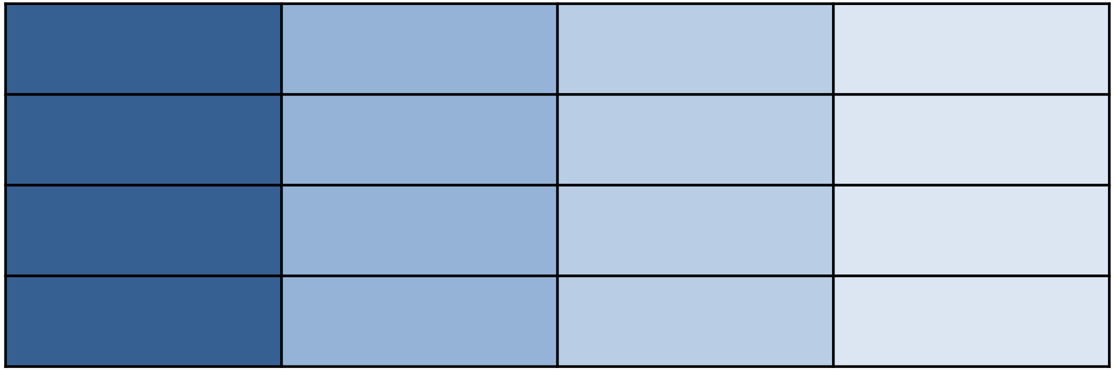
Loughborough University

BLAKE core = keyed permutation of a 4x4 state

4x4 32-bit words for BLAKE-256

4x4 64-bit words for BLAKE-512

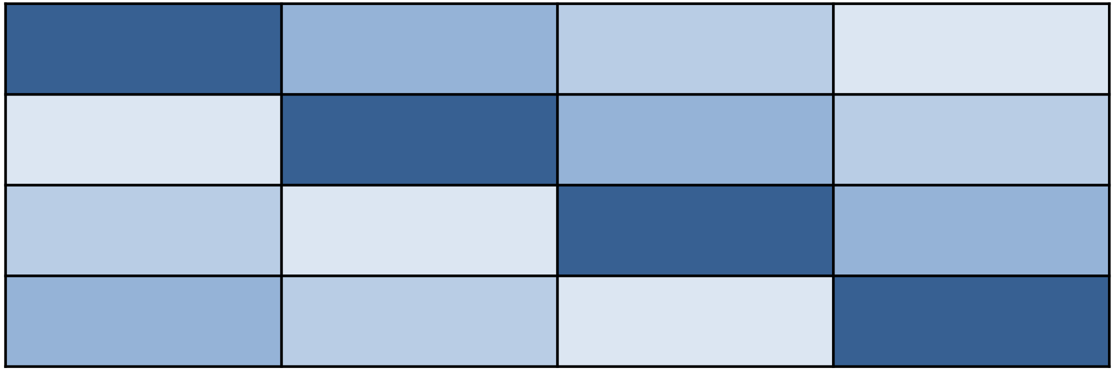
BLAKE core = keyed permutation of a 4x4 state



4x4 32-bit words for BLAKE-256

4x4 64-bit words for BLAKE-512

BLAKE core = keyed permutation of a 4x4 state



4x4 32-bit words for BLAKE-256

4x4 64-bit words for BLAKE-512

The **G** transform of (a,b,c,d)

$a \mathrel{+}= X \oplus \text{const}$

$a \mathrel{+}= b$

$d = (d \oplus a) \ggg 16$

$c \mathrel{+}= d$

$b = (b \oplus c) \ggg 12$

$a \mathrel{+}= Y \oplus \text{const}$

$a \mathrel{+}= b$

$d = (d \oplus a) \ggg 8$

$c \mathrel{+}= d$

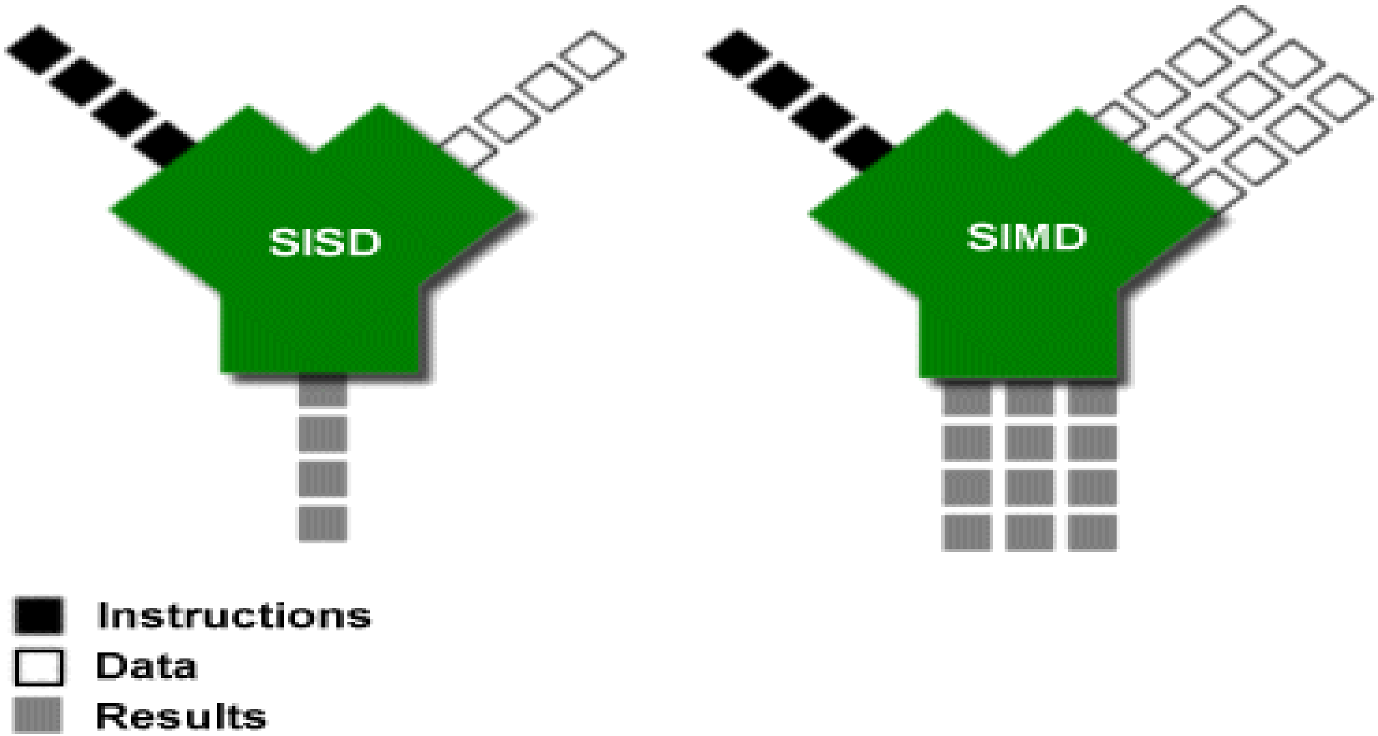
$b = (b \oplus c) \ggg 7$

ChaCha's core transform

$$
\begin{aligned}
&a \mathrel{+}= b \\
&d = (d \oplus a) \lll 16 \\
&c \mathrel{+}= d \\
&b = (b \oplus c) \lll 12
\end{aligned}
$$

$$
\begin{aligned}
&a \mathrel{+}= b \\
&d = (d \oplus a) \lll 8 \\
&c \mathrel{+}= d \\
&b = (b \oplus c) \lll 7
\end{aligned}
$$

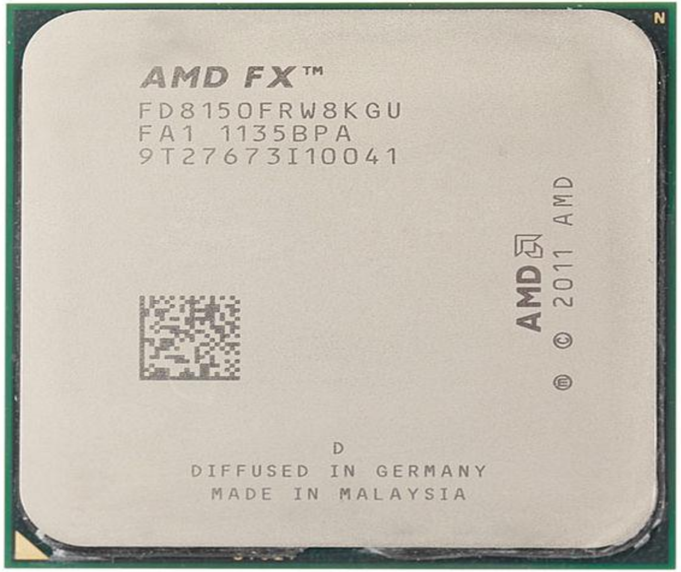
SIMD



http://arstechnica.com/old/content/2000/03/simd.ars

Desktop, laptops, servers

SSE (128b), XOP, AVX (256b)



Tablets, smartphones
ARM’s NEON (128b)

Gaming consoles
Cell SPU (128b)

Intel Many Integrated Core Architecture (512b)

BLAKE: same instruction, multiple data

a += X $\oplus$ const	a += X $\oplus$ const	a += X $\oplus$ const	a += X $\oplus$ const
a += b	a += b	a += b	a += b
d = (d $\oplus$ a) >>> 16	d = (d $\oplus$ a) >>> 16	d = (d $\oplus$ a) >>> 16	d = (d $\oplus$ a) >>> 16
c += d	c += d	c += d	c += d
b = (b $\oplus$ c) >>> 12	b = (b $\oplus$ c) >>> 12	b = (b $\oplus$ c) >>> 12	b = (b $\oplus$ c) >>> 12
a += Y $\oplus$ const	a += Y $\oplus$ const	a += Y $\oplus$ const	a += Y $\oplus$ const
a += b	a += b	a += b	a += b
d = (d $\oplus$ a) >>> 8	d = (d $\oplus$ a) >>> 8	d = (d $\oplus$ a) >>> 8	d = (d $\oplus$ a) >>> 8
c += d	c += d	c += d	c += d
b = (b $\oplus$ c) >>> 7	b = (b $\oplus$ c) >>> 7	b = (b $\oplus$ c) >>> 7	b = (b $\oplus$ c) >>> 7

Straightforward SIMD representation

How to **implement** it?

Past

SSE2

Streaming SIMD Extensions 2 (2001)

128-bit SIMD => 4-way 32-bit arithmetic

Intel Xeon, Celeron, Core i7, Atom
AMD Athlon64, Opteron
VIA C7, Nano
Etc.

Implementing **a += b** using SSE2

a1	a2	a3	a4

PADDD (4-way 32-bit integer addition)

b1	b2	b3	b4

=

a1 + b1	a2 + b2	a3 + b3	a4 + b4

Implementing $\mathbf{d = (d \oplus a) >>> 16}$ using SSE2

d1	d2	d3	d4

PXOR (XOR of two 128-bit registers)

a1	a2	a3	a4

=

$d1 \oplus a1$	$d2 \oplus a2$	$d3 \oplus a3$	$d4 \oplus a4$

Implementing **$d = (d \oplus a) \ggg 16$** using SSE2

PSLLD (4-way 32-bit left-shift)

$d1 \oplus a1$	$d2 \oplus a2$	$d3 \oplus a3$	$d4 \oplus a4$

=

$(d1 \oplus a1) \ll 16$	$(d2 \oplus a2) \ll 16$	$(d3 \oplus a3) \ll 16$	$(d4 \oplus a4) \ll 16$

Implementing **d = (d $\oplus$ a) >>> 16** using SSE2

PSRLD (4-way 32-bit right-shift)

d1 $\oplus$ a1	d2 $\oplus$ a2	d3 $\oplus$ a3	d4 $\oplus$ a4

=

(d1 $\oplus$ a1)>>16	(d2 $\oplus$ a2)>>16	(d3 $\oplus$ a3)>>16	(d4 $\oplus$ a4)>>16

Implementing $\mathbf{d = (d \oplus a) \ggg 16}$ using SSE2

$(d1 \oplus a1) \gg 16$	$(d2 \oplus a2) \gg 16$	$(d3 \oplus a3) \gg 16$	$(d4 \oplus a4) \gg 16$

PXOR (XOR of two 128-bit registers)

$(d1 \oplus a1) \ll 16$	$(d2 \oplus a2) \ll 16$	$(d3 \oplus a3) \ll 16$	$(d4 \oplus a4) \ll 16$

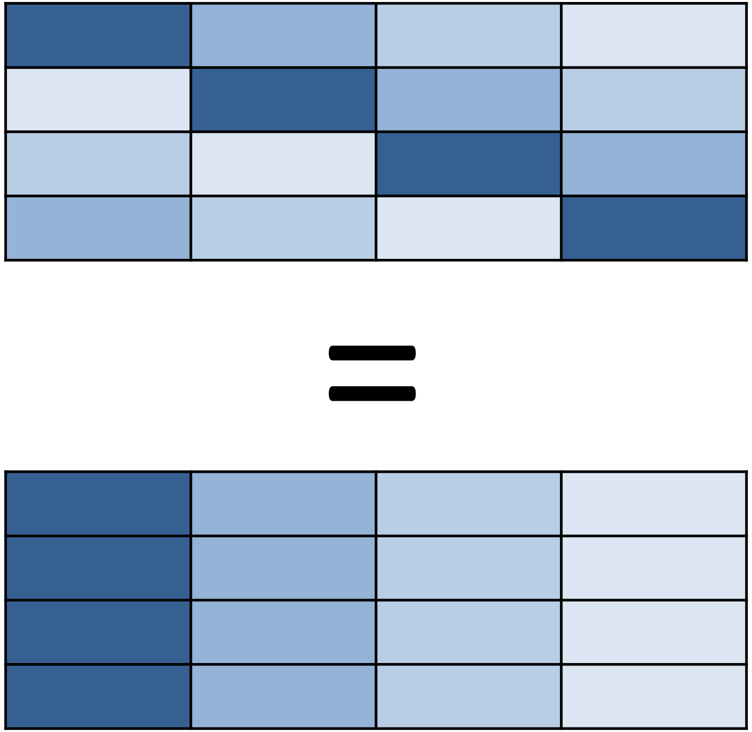
=

$(d1 \oplus a1) \ggg 16$	$(d2 \oplus a2) \ggg 16$	$(d3 \oplus a3) \ggg 16$	$(d4 \oplus a4) \ggg 16$

Etc. etc.

“Shiftrows” to work on the diagonalized state

PSHUFD (4-way 32-bit shuffle)



SSSE3

Supplemental Streaming SIMD Extensions 3 (2006)

Intel Xeon 5100, Core 2, etc. (2006+)

AMD “Bobcat” and “Bulldozer” µarchs (2011)

Byte-shuffle PSHUFB used for >>> 16 and >>> 8

PSHUFB for >>>16 of 4 32-bit words

d1H	d1L	d2H	d2L	d3H	d3L	d4H	d4L

=

d1L	d1H	d2L	d2H	d3L	d3H	d4L	d4H

SSE4.1

Streaming SIMD Extensions 4 (2006)

Intel Core (2006), AMD Phenom (2007)

Introduces conditional copying PBLENDW

Naive initialization of permuted message

m[p[6]]	m[p[4]]	m[p[2]]	m[p[0]]

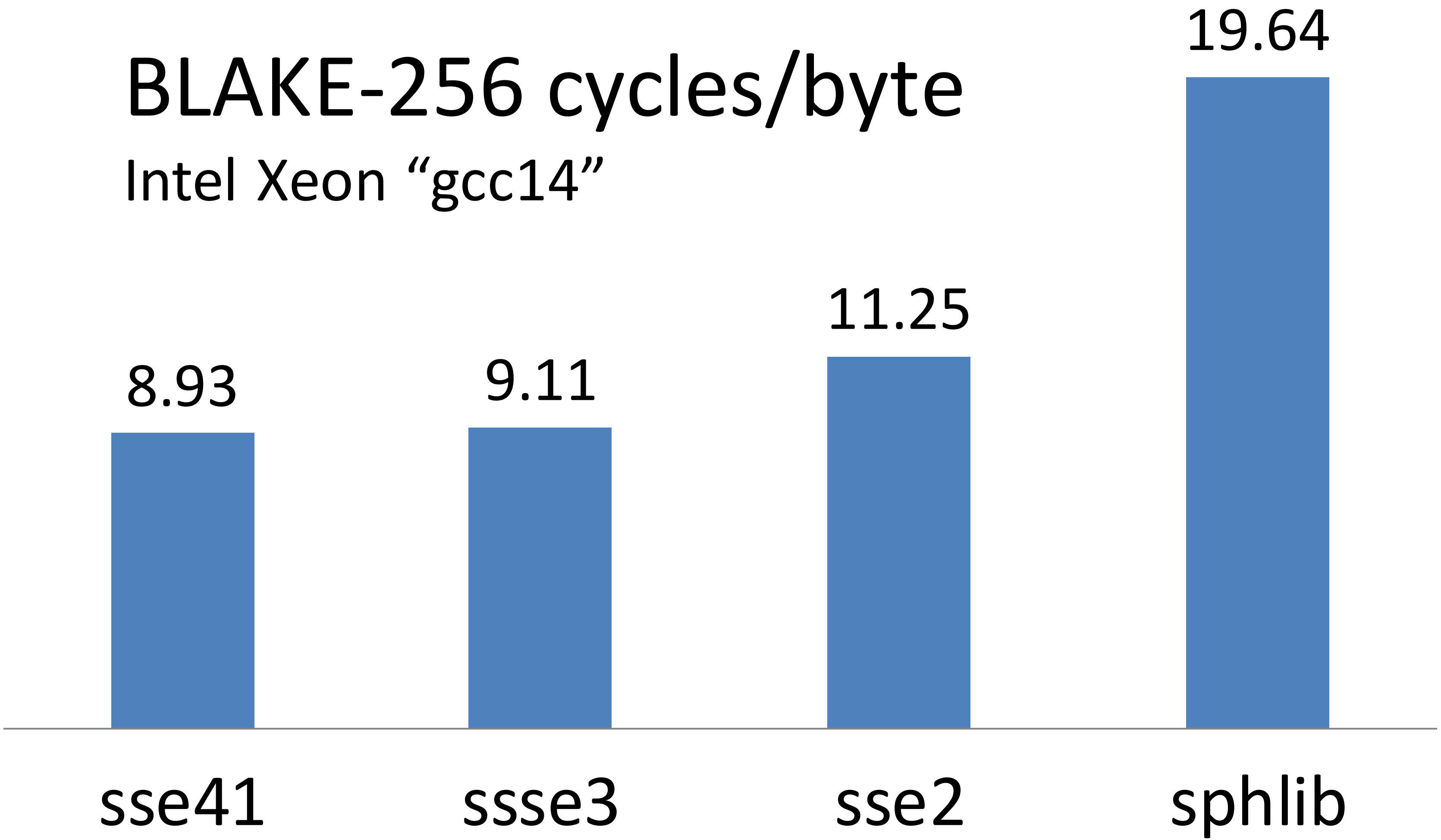
Using the C intrinsic

```
X = _mm_set_epi32( m[ p[6] ], m[ p[4] ], m[ p[2] ], m[ p[0] ] )
```

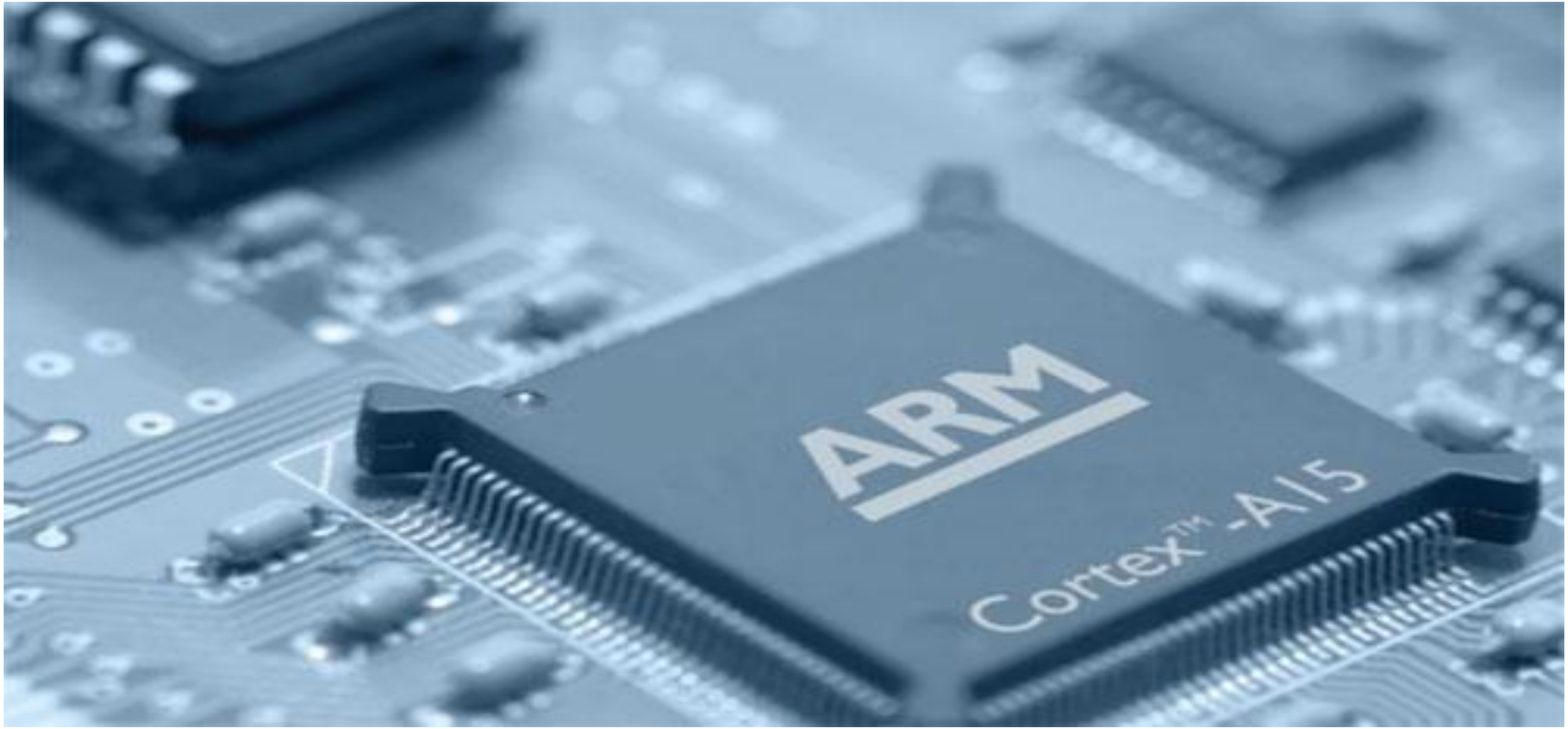
PBLENDW can be used to avoid LUTs

Example for round 2:

```
tmp0 = _mm_blend_epi16(m1, m2, 0x0C);
tmp1 = _mm_slli_si128(m3, 4);
tmp2 = _mm_blend_epi16(tmp0, tmp1, 0xF0);
buf1 = _mm_shuffle_epi32(tmp2, _MM_SHUFFLE(2,1,0,3));
tmp3 = _mm_shuffle_epi32(m2, _MM_SHUFFLE(0,0,2,0));
tmp4 = _mm_blend_epi16(m1, m3, 0xC0);
tmp5 = _mm_blend_epi16(tmp3, tmp4, 0xF0);
```


BLAKE-256 cycles/byte
Intel Xeon “gcc14”
8.93
9.11
11.25
19.64
sse41
ssse3
sse2
sphlib

Present


ARM
Cortex™-A15


APL0498 33980130
ECF750A7 1131
A5
K3PE4E400B-XGC1
S023RNP21 1131

NEON™

128-bit SIMD architecture

Packed 4-way 32-bit and 2-way 64-bit arithmetic

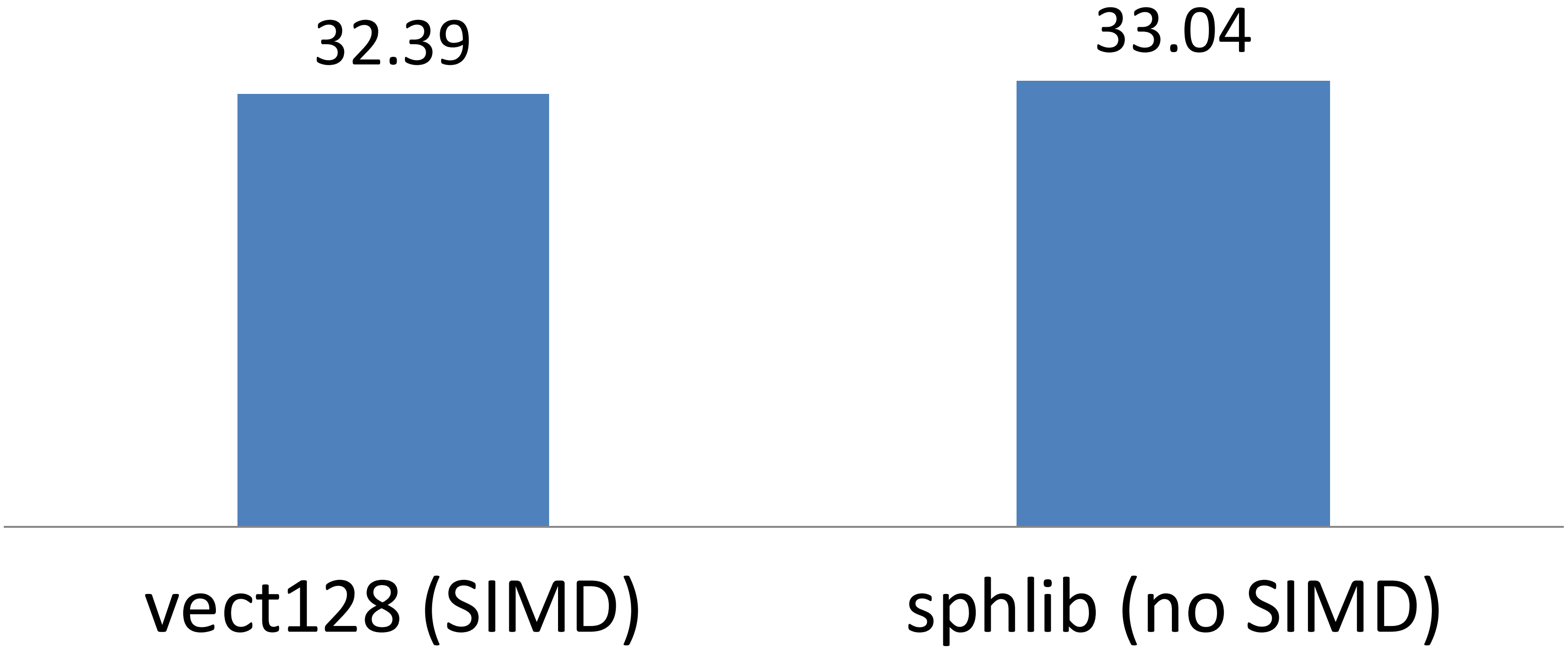
+ other useful instructions as VSWP (swap)

Leurent’s vect128 implementation

```
#elif defined(__ARM_NEON__)
…
#include <arm_neon.h>
```

BLAKE-256 cycles/byte

Cortex A8 “h1mx515”

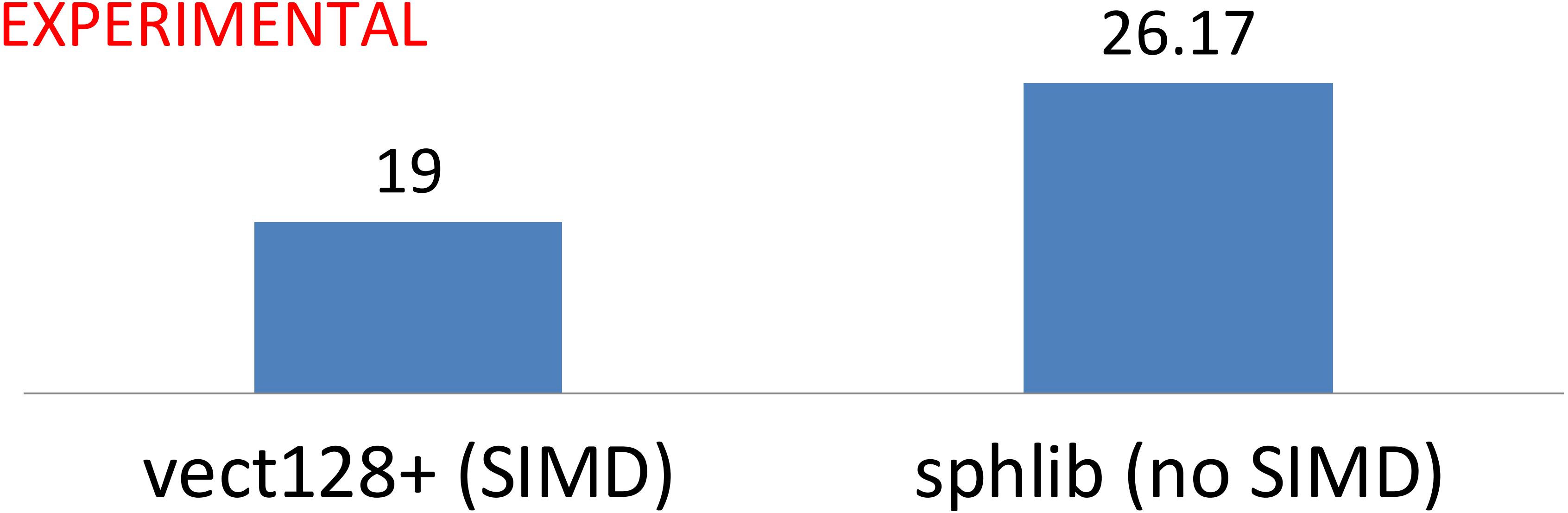


?

BLAKE-256 cycles/byte

Cortex A8 in NOKIA N900

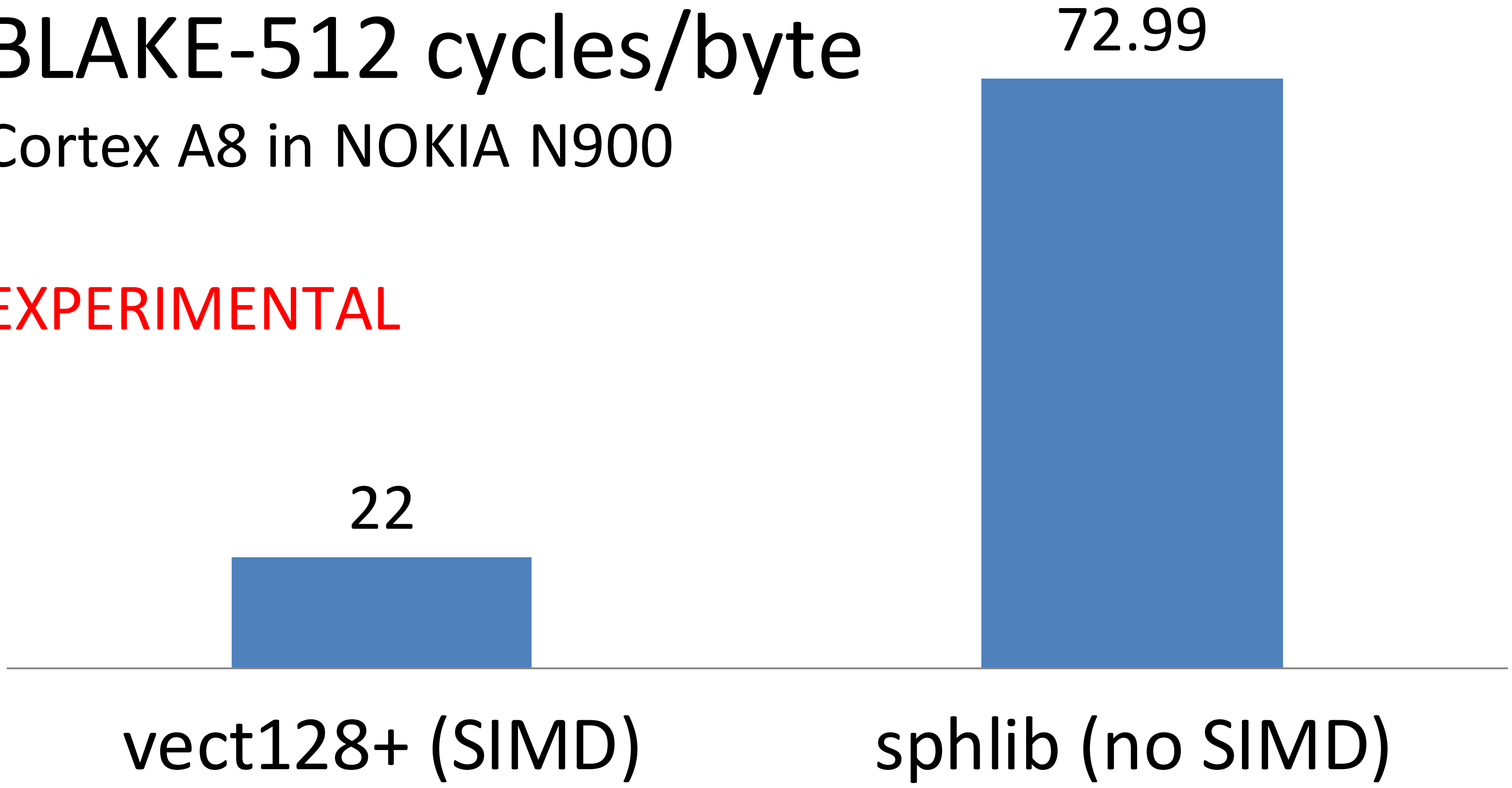
EXPERIMENTAL



BLAKE-512 cycles/byte

Cortex A8 in NOKIA N900

EXPERIMENTAL



Future

Haswell Processor Family Overview (Traditional)

- 22nm Process
- Socket: G3(947 pin) MB, H3(1150 pin) DT
- Fully Integrated VR
- Power Aware Interrupt Routing for power / performance
- Intel® Hyper-Threading Technology
- Intel® AVX 2.0 extensions and AES-NI instructions improvements
- Intel® Turbo Boost Technology

- **Processor Graphics**: 3D Gfx/Media Arch/Perf. Support for new APIs, 3 Digital Displays, DP 1.2/HDMI 1.4/eDP
- **Desktop/All-in-One**: DDR3/DDR3L-1600 2D/Ch non-ECC UDIMM/SODIMM
- **Mobile**: DDR3L (only), POR for 1600 2D/Ch 6L rPGA, non-ECC SODIMM
- DDR Power Gating for lower idle power (MB)
- PCIe 3.0 x16/2x8, 3 controllers
- Power Optimizer (CPPM) Support (MB)
- Last Level Cache (LLC) shared between CPU and Integrated Graphics
- Hotham 1.0 Support
- Overclocking Improvements

Haswell Offers Better Power & Performance with Improved I/O

AVX2 (to appear in 2013)

256-bit SIMD => 4-way 64-bit arithmetic

Implementing **a += b** using AVX2

a1	a2	a3	a4

VPADDQ (4-way 64-bit integer addition)

b1	b2	b3	b4

=

a1 + b1	a2 + b2	a3 + b3	a4 + b4

Straighforward BLAKE-512 implementation

1 SIMD instruction (4-way 64-bit op), instead of 2 with SSE (2-way 64-bit ops)

VPSHUFD for >>>32 of 4 64-bit words

d1Hi	d1Lo	d2Hi	d2Lo	d3Hi	d3Lo	d4Hi	d4Lo

=

d1Lo	d1Hi	d2Lo	d2Hi	d3Lo	d3Hi	d4Lo	d4Hi

VPSHUFB for >>>16 of 4 64-bit words

d1Hi	d1Mi	d1Mo	d1Lo	d2Hi	d2Mi	d2Mo	d2Lo	d3Hi	d3Mi	d3Mo	d3Lo	d4Hi	d4Mi	d4Mo	d4Lo

=

d1Lo	d1Hi	d1Mi	d1Mo	d2Lo	d2Hi	d2Mi	d2Mo	d3Lo	d3Hi	d3Mi	d3Mo	d4Lo	d4Hi	d4Mi	d4Mo

AVX2 introduces “gather” instructions

=> parallel table look-ups

Loading message words wrt the permutation p

m

VPGATHERDQ (4-way 64-bit gather)

p[6]	p[4]	p[2]	p[0]

=

m[p[6]]	m[p[5]]	m[p[2]]	m[p[0]]

AVX2 and BLAKE-256?

Loading message words wrt the permutation p

m

VPGATHERDD (4-way 32-bit gather)

p[6]	p[4]	p[2]	p[0]

=

m[p[6]]	m[p[5]]	m[p[2]]	m[p[0]]

No-look-up permutations

m0	m1	m2	m3	m4	m5	m6	m7

m8	m9	m10	m11	m12	m13	m14	m15

No-look-up permutations

Use any-to-any word permutation VPERMD

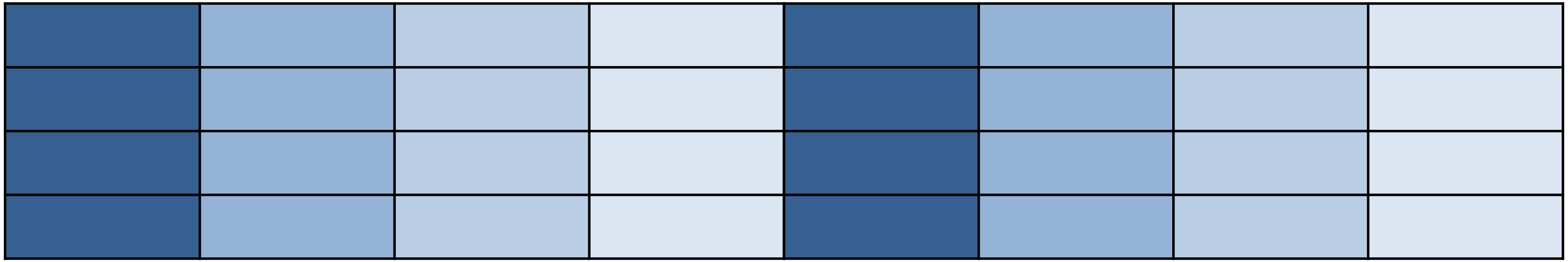
Element Width	Vector Width	Instruction	Launch
BYTE	128	PSHUFB	SSE4
DWORD	256	VPERMD VPERMPS	AVX2 New!
QWORD	256	VPERMQ VPERMPD	AVX2 New!

No-look-up permutations

```
; load relevant indices
vmovdqa ymm8, [perm1 + 00]
vmovdqa ymm9, [perm1 + 32]
; permute each message half accordingly
vpermd ymm4, ymm8, ymm10
; i.e.: “ymm4[i] = ymm10[ymm8[i]], i=0, …, 7”
vpermd ymm5, ymm9, ymm11
; take the 4 32-bit words needed
vpblendd ymm4, ymm4, ymm5, 01111101b
```

Multistream hashing

A 256-bit register contains 2 rows from 2 instances of BLAKE-256



Direct 2x speed-up if messages synchronized

Not before 2013…


2012
DOOMSDAY
We were warned

What can we do NOW?

256-bit instruction sets are already here

AVX ("Sandy Bridge" μarch)

XOP ("Bulldozer" μarch)

No full-SIMD BLAKE, but speed-ups expected

SSE4: 2-operand instructions

; xmm4 = blend(xmm4, xmm11, 00001100b)

pblendw xmm4, xmm11, 00001100b

AVX: 3 operands!

; xmm4 = blend(xmm13, xmm11, 00001100b)

vpblendw xmm4, xmm13, xmm11, 00001100b

Message caching

Store the reused permuted messages in 256-bit registers

$\Rightarrow$ 9 instead of 13 loads

Round	Permuted message
0	p0(m)
1	p1(m)
2	p2(m)
3	p3(m)
4	p4(m)
5	p5(m)
6	p6(m)
7	p7(m)
8	p8(m)
9	p9(m)
10	p0(m)
11	p1(m)
12	p2(m)
13	p3(m)

New implementations

(coding by Samuel)

AVX2 assembly for BLAKE-512 and BLAKE-256

```
%macro G 2

    vpaddq   ymm0, ymm0, %1 ; row1 + buf1
    vpaddq   ymm0, ymm0, ymm1 ; row1 + row2
    vpxor    ymm3, ymm3, ymm0 ; row4 ^ row1
    vpshufd  ymm3, ymm3, 10110001b ; row4 >>> 32

    vpaddq   ymm2, ymm2, ymm3 ; row3 + row4
    vpxor    ymm1, ymm1, ymm2 ; row2 ^ row3
    VPROTRQ  ymm1, 25    ; row2 >>> 25

    vpaddq   ymm0, ymm0, %2 ; row1 + buf1
    vpaddq   ymm0, ymm0, ymm1 ; row1 + row2
    vpxor    ymm3, ymm3, ymm0 ; row4 ^ row1
    vpshufb  ymm3, ymm3, ymm15 ; row4 >>> 16

    vpaddq   ymm2, ymm2, ymm3 ; row3 + row4
    vpxor    ymm1, ymm1, ymm2 ; row2 + row3
    VPROTRQ  ymm1, 11     ; row2 >>> 11
%endmacro
```

Tested on Intel's SDE

Ready to benchmark

AVX assembly for BLAKE-256

```
%ifdef CACHING
vmovdqa [rsp + 16*4 + 2*64 + 00], xmm4
vmovdqa [rsp + 16*4 + 2*64 + 16], xmm5
vmovdqa [rsp + 16*4 + 2*64 + 32], xmm6
vmovdqa [rsp + 16*4 + 2*64 + 48], xmm7
%endif

%endmacro

%macro MSGLOAD3 0
;m[ 3] m[ 2]  m[ 1]  m[ 0] -> m[11] m[13] m[ 3] m[ 7]
;m[ 7] m[ 6]  m[ 5]  m[ 4] -> m[14] m[12] m[ 1] m[ 9]
;m[11] m[10]  m[ 9]  m[ 8] -> m[15] m[ 4] m[ 5] m[ 2]
;m[15] m[14]  m[13]  m[12] -> m[ 8] m[ 0] m[10] m[ 6]
;xmm7 xmm6 xmm5 xmm4 <- xmm13 xmm12 xmm11 xmm10

; this one reads words from all 4 words!
vpunpckhdq xmm8, xmm10, xmm11 ; 7 3 6 2
vpalignr  xmm4, xmm13, xmm8, 8 ; 13 12 7 3
vpinsrd   xmm4, xmm4, [rsp + 11*4], 2 ; 13 11 7 3
vpshufd   xmm4, xmm4, 10110001b ; 11 13 3 7

vpblendw  xmm5, xmm13, xmm10, 00001100b ; 15 14 1 12
vpinsrd   xmm5, xmm5, [rsp + 9*4], 3 ; 9 14 1 12
vpshufd   xmm5, xmm5, 10000111b ; 14 12 1 9

vpblendw  xmm6, xmm11, xmm10, 00110000b ; 7 2 5 4
vpblendw  xmm6, xmm6, xmm13, 11000000b ; 15 2 5 4
vpshufd   xmm6, xmm6, 11000110b ; 15 4 5 2

vpunpckldq xmm8, xmm10, xmm12 ; 9 1 8 0
vpunpckhdq xmm7, xmm11, xmm12 ; 11 7 10 6
vpunpcklqdq xmm7, xmm7, xmm8 ; 8 0 10 6
```

128-bit SIMD

3-operand instructions

Message caching

7.62 cycles/byte

(Core i7 2630QM, Sandy Bridge)

Next steps?

1

ARM NEON benchmarks

2

AVX BLAKE-256 on eBASH

3

XOP implementations

4

AVX2 benchmarks (2013)

Thank you!