SHA3 2014 WORKSHOP

August 22, 2014, University of California, Santa Barbara

Parallelized hashing via j-lanes and j-pointers tree modes with applications to SHA-256

Shay Gueron[1,2]

[1] **University of Haifa**

Department of Mathematics, Faculty of Natural Sciences, University of Haifa, Israel

[2] **Intel Corporation**

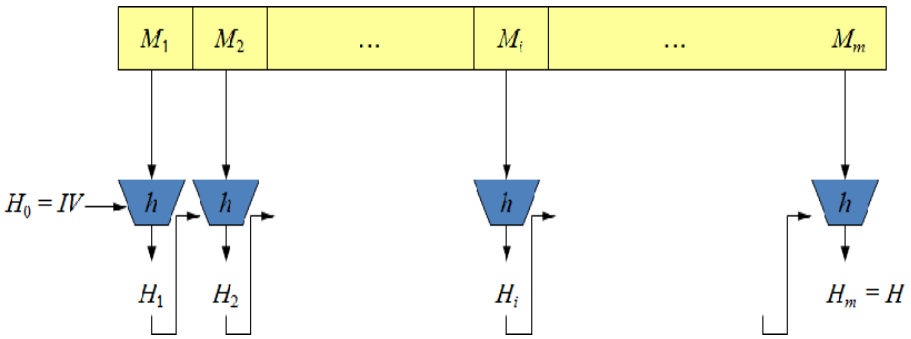
Intel Architecture Group, Intel Corporation, Israel Development Center, Haifa, Israel

shay@math.haifa.ac.il

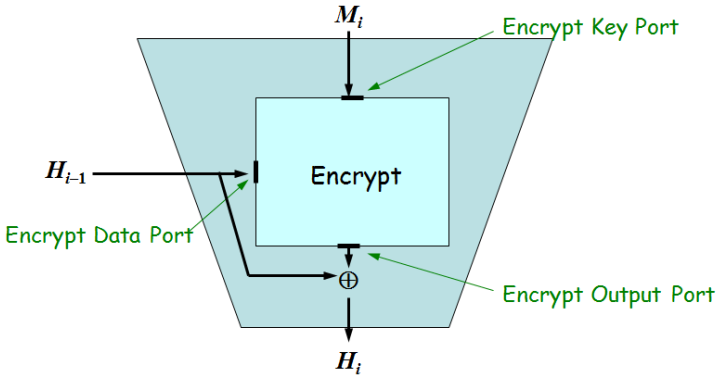
We discuss SHA-256 here

SHA-256 brief

Merkle-Damgård



Davies-Meyer Compression Function



- SHA-1 SHA-256 SHA-512
- A **Serial** algorithms
  - Merkle-Damgård construction
  - Davies-Meyer Compression Function
  - Padding (to encode message length)
- Consume 64B message block
- Update a "state" of 8 32-bit dwords
- Process **in 32-bit dwords**
  - 32-bit **Add/XOR/Rotate/Shift**
  - **Ternary logical operations**

    e.g., Maj (a,b,c) = (a&b)^(a&c)^(b&c)

SHA-256 is serial, operates on 32-bit dwords, ADD/XOR/Rotate/Shift/Ternary-logical

SHA-256 performance (***serial***)

- (2008) SHA-256 optimized ALU code: **~18** C/B (Core2)
  - Was set as the minimum performance bar for SHA-3 candidates
  - 2.3x slower than (the compromised hash function) SHA-1 (~8 C/B)
  - Performance is an important aspect of a hash algorithm
    - Probably a gate for migration

- (2012) the **SMS** method: **10.8** C/B (Ivy Bridge architecture)
  - **S**imultaneous **M**essage **S**cheduling: parallelizing *part* of SHA-256 using SIMD
    (S. Gueron, V. Krasnov [1], [2])

- **SIMD** (**S**ingle **I**nstruction **M**ultiple **D**ata) brief
  - Registers hold multiple "elements" in "lanes" (e.g., bytes, words, dwords)
  - One instruction applies the same operation to all the lanes (elements)
    - e.g., add 8 dwords simultaneously : paddd ymm1, ymm2, ymm3

SHA-256 performance (***multiple*** msg's)

- (2012) **S**imultaneous **Hash**: doing everything with SIMD
  - Processing multiple messages simultaneously (S. Gueron, V. Krasnov [3])
  - Produce n *standard* digests
  - n = the number of messages to parallel-process.
    - Choice of n depends on the architecture

- S-HASH Performance

  (n x 8KB buffers; Haswell Microarchitecture)

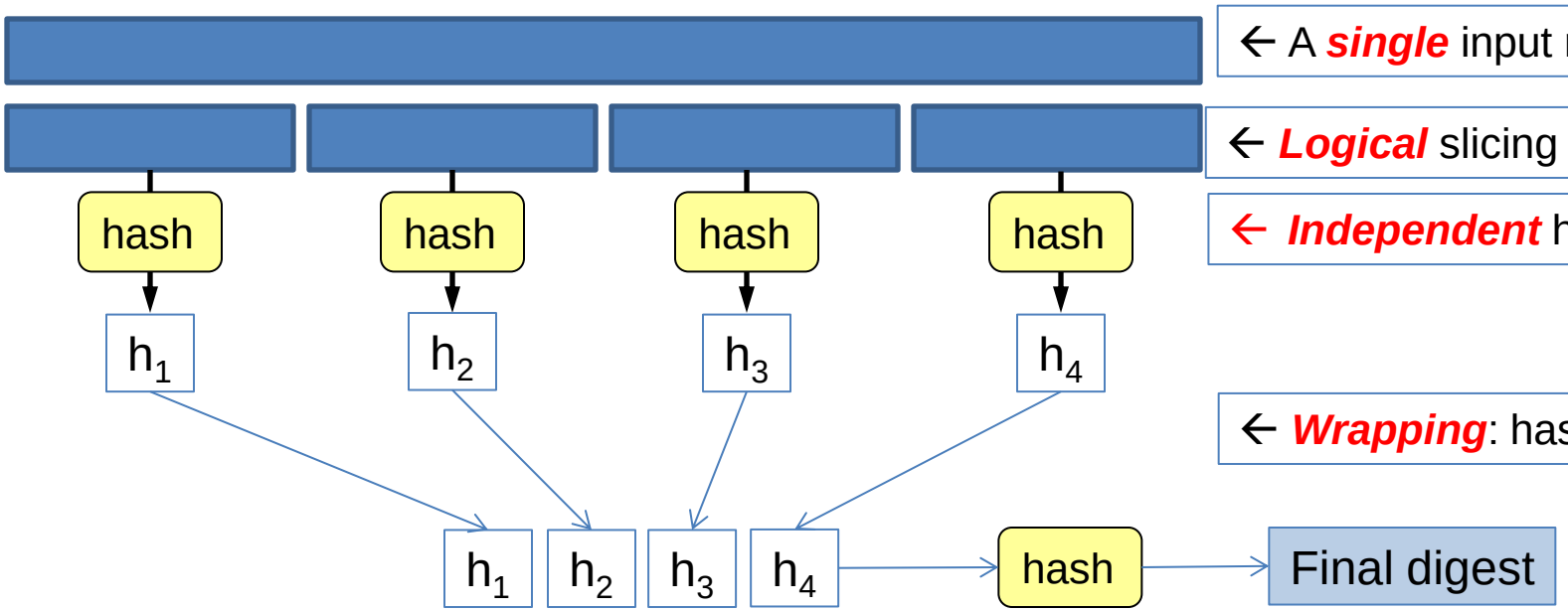
  Compare to Serial SHA-256 (Haswell): **7.8** C/B

  - AVX (using 128 bit XMM registers) → n=4: **5.2** C/B
  - AVX2 (using 256-bit YMM registers) → n=8: **2.7** C/B
  - AVX512 (future 512-bit ZMM registers) → n=16: estimated ~1 C/B

- OpenSSL 1.02beta already has this Multi Buffer feature for msg's > 64KB

If the computational workload requires hashing of multiple messages, use S-HASH

Parallelized hashing - ***single*** msg : a flat tree



Efficient?

- More compression function invocations
  - Pad and finalized each slice
  - Overhead (wrap)
    - Hash an extra message of length j digests (j x 32B)
- But step #2 can be parallelized
  - Using SIMD

Flat tree: questions

The approach (recap):

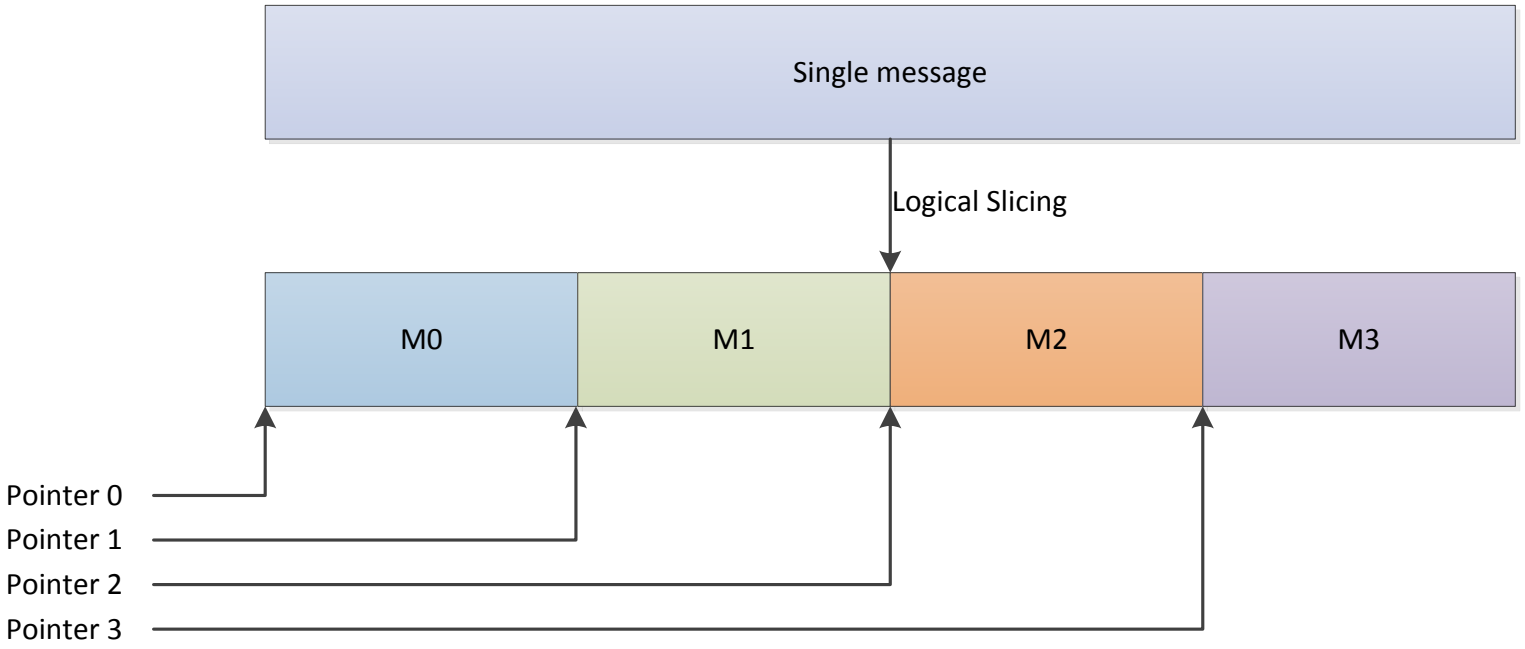
1. **Slice** (logical slicing of a single message to multiple (j) disjoint slices)
2. **Parallelized processing** (hash each slice independently to get j digests)
3. **Wrap** (hash the j digests into a final output digest)

The questions:

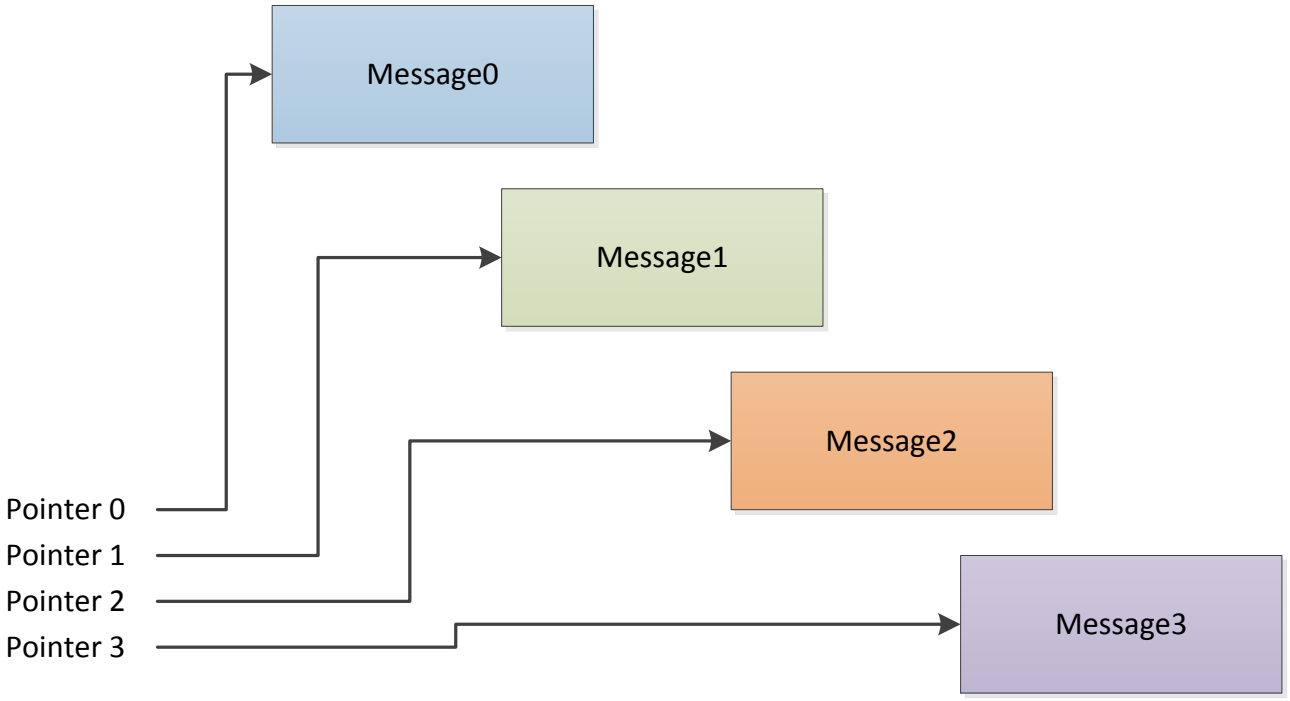
- How to define the logical slicing of a message?
- How many slices?
- How to distinguish from standard SHA?

Full details in S. Gueron [4], [5]

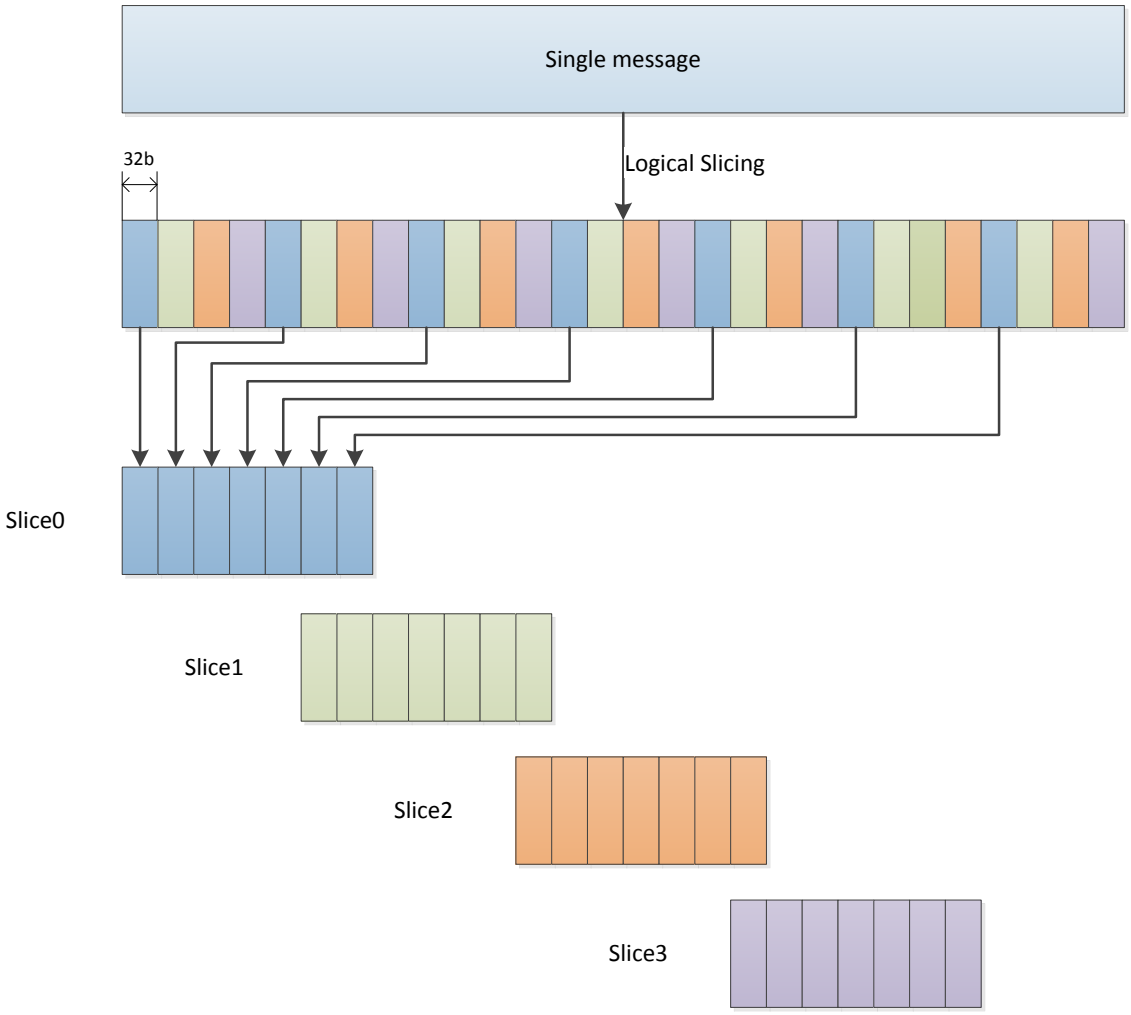
j-pointers (“contiguous” message)



j-pointers (“scattered” message)



j-lanes (“interleaved” message)



j-lanes vs. j-pointers logical slicing

- j-pointers: view message as a concatenation of j disjoint buffers
  - No memory locality: buffers can be contiguous or scattered
    - Useful for a scattered message
    - Amenable for multi-core processing (or any “distributed” computation)
  - For SIMD parallelization: requires transposing the data (into SIMD registers) before actual processing

- j-lanes: view message as j interleaved slices (interleaving 32-bit dwords)
  - Memory locality: load from a single pointer + offset
  - Loading automatically sets data in SIMD registers ready for actual processing

- Both have similar performance characteristics
- j-lanes is slightly faster (no need to transpose data)

What j to use?

- Smaller j ➔ less ”overheads”
- Larger j ➔ more parallel processing on wider SIMD architectures

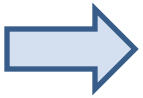
- The optimal choice of j for a given SIMD architecture
  - j = number of dwords in the SIMD registers = (SIMD bits) / 32
    (SHA-256 operates on 32-bit dwords)

  - AVX/SSE – 128-bit xmm registers: j=4
  - AVX2 – 256-bit ymm registers best: j=8
  - AVX512 –512-bit zmm registers: j=16 (future architecture)

- Note: an architecture can support any choice of j
  - e.g., can use j=16 with AVX2;

j-lanes and AVX512 future architecture

- AVX512 support (optimally) j=16 lanes
- More available registers: keep state and message schedule in registers
- New instructions “ternlog” and “rotate” - useful for SHA-256 algorithm
  - **vpternlogd** – ternary logic
    - vpternlogd a, b, c, 0x96 = a xor b xor c
    - vpternlogd a, b, c, 0xe8 = majority (a,b,c)
    - vpternlogd g, f, e, 0xd8 = choose (e,f,g)
  - **vprord** – packed rotate right (replacing a SW sequence);
    - vpsrld tmp, src, imm
    - vpslld dst, src, 32-imm
    - vpxor dst, dst, tmp

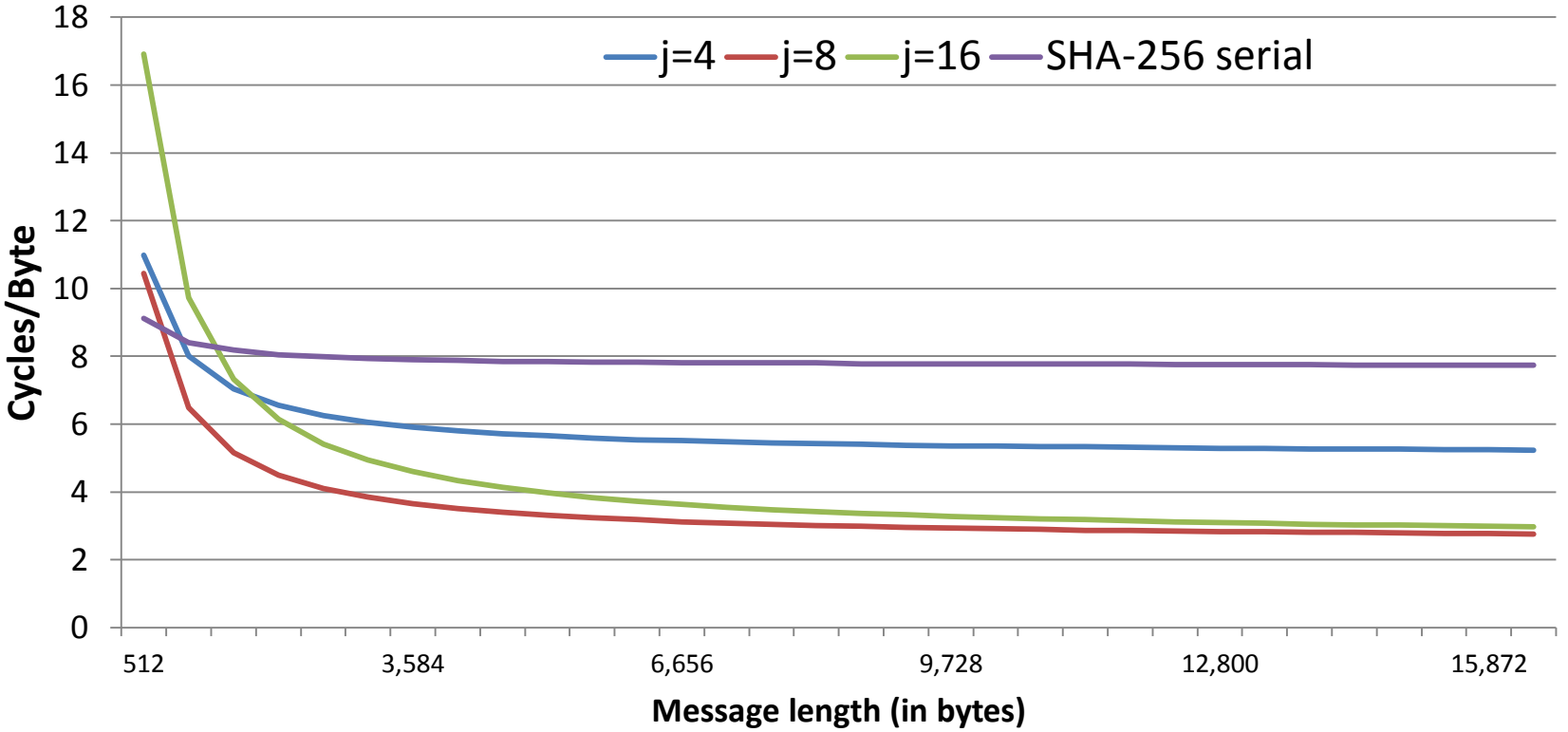


- vprord dst, src, imm

Expect AVX512 (with j=16) to improve the performance of parallelized hashing

j-lanes SHA-256 performance

(AVX2 results on Architecture Codename Haswell)

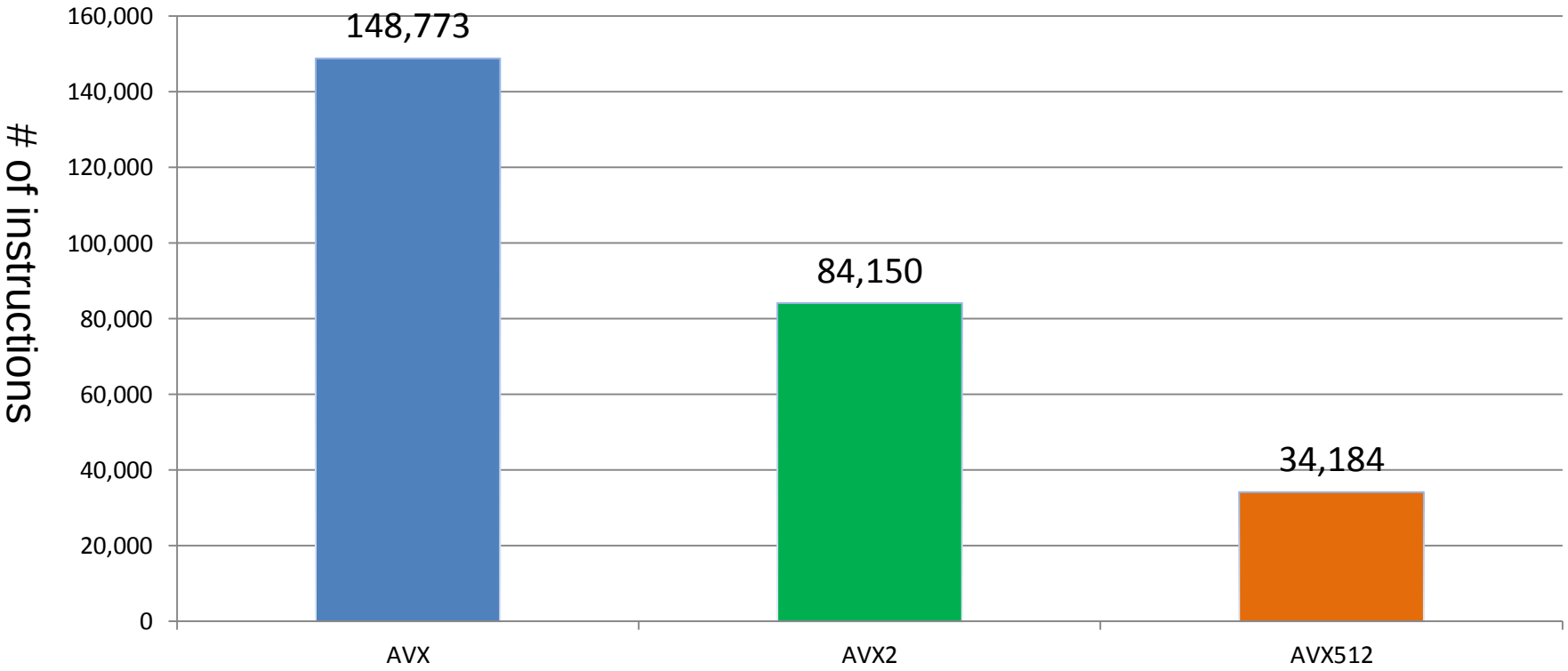


- Best performance on AVX2 architecture uses j=8 lanes
- Long messages: j=8 → ~2.76 C/B; j=16 →2.97 C/B → ~7.5% difference
- For j=4/8/16: j-lanes SHA-256 outperforms serial SHA-256 already from 1.5KB

j-lanes SHA-256 future performance

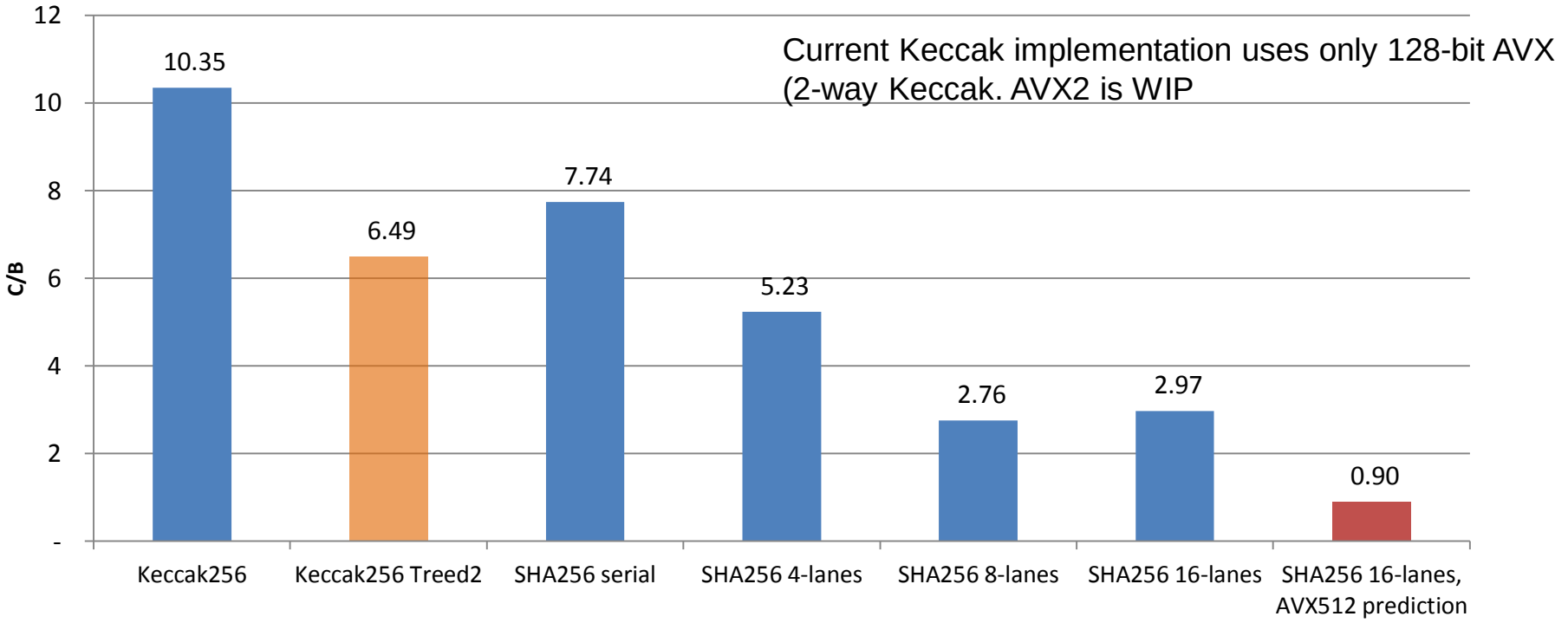
AVX512 future architecture

8KB message j=16 # of instructions



- The use of AVX512 with j=16 lanes reduces the number of instructions by ~2.5X
- Expect AVX512 (with j=16) to improve the performance of parallelized hashing

Current performance comparison:
Keccak256, Treed Keccak256, SHA-256, j-lanes SHA-256
Haswell microarchitecture (AVX2)



Keccak numbers from http://bench.cr.yp.to/results-hash.html, wintermute machine ("long message")

- j-lanes SHA-256 has a significant performance advantage
  - Even with "less-then-optimal" j=16 on current architecture
- Expect AVX512 with j=16 to improve further

Distinguishing tree hash from serial hash via different IV's

- Goal: avoid the situation of SHA-256 (M1) = j-lanes-SHA-256 (M2)

- A general approach: using a prefix block for each computed hash
  - Prepend prefix blocks to the j+1 hashes to effectively modify the IV's
- An option that allows full flexibility (values can be pre-computed)
  - Prefix block Pre[i] = j||i||type|| HASH||0's
    - j: the number of lanes/pointers (fixed length)
    - i: the "index" of the lane/pointer (fixed length)
    - Type: 0x0 indicates j-lanes hash, 0x1 indicates j-pointers hash.
    - HASH, the name of the underlying hash function, e.g. HASH = "SHA256"
    - 0's to fill the block
- Option for only one pefix: Pre[i] = j||0||type|| HASH||0's

Summary, conclusions, and call for action

Conclusion:

- j-lanes hash & j-pointers offer a significant performance advantage on current and future architecture. It would be a useful standard

Options to consider:

- Best to allow full flexibility:
  - Any j (=4/8/16), both j-lanes and j-pointer, any underlying HASH function
  - j+1 Prefix values (1 Prefix might suffice?)
- If only one j can be standardized: suggest j=16
  - Small performance compromise on current CPU's & high forthcoming potential

Call for action:

- Standardize j-lanes / j-pointers hash as a hashing mode

References

1. Gueron, Krasnov: “Parallelizing message schedules to accelerate the computations of hash functions”, Journal of Cryptographic Engineering 2: 241-253 (2012).
2. Gueron, Krasnov: “Efficient implementations of SHA256 and SHA512, using the Simultaneous Message Scheduling method”, https://rt.openssl.org/Ticket/Display.html?id=2784&user=guest&pass=guest
3. Gueron, Krasnov: “Simultaneous Hashing of Multiple Messages”, Journal of Information Security, 3: 319-325 (2012).
4. Gueron: “A j-Lanes Tree Hashing Mode and j-Lanes SHA-256”, Journal of Information Security, 4: 7-11 (2013).
5. S. Gueron: “Parallelized Hashing via j-Lanes and j-Pointers Tree Modes, with Applications to SHA-256”, Journal of Information Security 5:91-113 (2014).

Backup

AVX512 (code snippet)

```
.macro round A,B,C,D,E,F,G,H,i,w0,w1,w9,w14
   vprord   $6, \E, %zmm8
   vprord   $11, \E, %zmm9
   vprord   $25, \E, %zmm10
   vpternlogd $0x96, %zmm10, %zmm9, %zmm8
   vpaddd   \w0, \H, %zmm12
   vmovdqa64   \G, %zmm13
   vmovdqa64   \A, \H
   vprord   $2, \A, %zmm9
   vprord   $13, \A, %zmm10
   vprord   $22, \A, %zmm11
   vpternlogd $0x96, %zmm11, %zmm10, %zmm9
   vpbroadcastd   (%rbx), %zmm11
   vpternlogd  $0xd8, \E, \F, %zmm13
   vpternlogd  $0xe8, \C, \B, \H
   vpaddd   %zmm12, %zmm11, %zmm11
   vpaddd   \w9, \w0, \w0
   vpaddd   %zmm13, %zmm11, %zmm11
   vpaddd   %zmm8, %zmm11, %zmm11
   vmovdqa64  \w1, %zmm8
   vmovdqa64  \w14, %zmm13
   vprord   $17, %zmm13, %zmm10
   vprord   $19, %zmm13, %zmm12
   vpsrld   $10, %zmm13, %zmm13
   vpternlogd $0x96,%zmm10,%zmm12,%zmm13
   vpaddd   %zmm11, \H, \H
   vpaddd   %zmm11, \D, \D
   vpaddd   %zmm9, \H, \H
   lea      4(%rbx), %rbx
   vprord   $7, %zmm8, %zmm11
   vprord   $18, %zmm8, %zmm12
   vpsrld   $3, %zmm8, %zmm8
   vpternlogd $0x96,%zmm11,%zmm12,%zmm8
   vpaddd   %zmm13, \w0, \w0
   vpaddd   %zmm8, \w0, \w0
.endm
```