Software implementation of SHA-3 family using AVX2

Roberto Cabral, Julio López *

[1]Institute of Computing, University of Campinas
cabral@lasca.ic.unicamp.br, jlopez@ic.unicamp.br

Abstract. *The Keccak algorithm was the winner of the competition organized by NIST to choose the new standard hash algorithm, called SHA-3. In this work, we present the details of our software implementation in conformity with draft FIPS 202. We follow two approaches for the implementation of SHA-3, the first one computes the digest for a single message, and the other one computes in parallel four digests from four different messages. The performance for the single implementation was accelerated using vector instructions of 128/256 bits, and it is as fast as the best implementation optimized for 64 bits published on eBASH. The parallel implementation is about* $2.5\times$ *faster than the single message implementation. The cryptographic primitive extendable-output functions, which is part of the draft FIPS 202, were also implemented.*

1. Introduction

The family of hash functions SHA (Standard Hash Algorithm) [FIPS 2008], was standardized by the NIST (National Institute of Standards and Technology) and currently is used in many applications and protocols. Recently, several attacks on hash algorithms of SHA family were found. In 2005, [Biham et al. 2005] and [Rijmen and Oswald 2005] showed collision attacks of reduced versions of SHA-1. In the same year, [Wang et al. 2005] showed an attack that theoretically breaks the resistance to collision. The second version of SHA, SHA-2, is based on SHA-1 and already had attacks in its reduced versions, as is shown in [Indesteege et al. 2009]. In 2007, NIST started a new competition to select the new version of SHA algorithm, called SHA-3 [NIST 2007]. After two rounds of competition, five finalists were chosen: BLAKE, Grøstl , JH, Keccak and Skein. In 2012, Keccak [Bertoni et al. 2008] was announced as the winner.

This work shows how to take advantage of the new vector instructions (AVX/AVX2) introduced on Intel® Architecture Processors to implement the SHA-3 family. We developed a sequential and a parallel versions of the SHA-3 hash function for the four security levels 112, 128, 192 and 256 bits; in addition, the extendable-output functions (XOFs) were implemented for 128 and 256 bit security levels.

2. AVX2

In 2013 was released the newest Intel micro-architecture, called Haswell. This architecture contains the AVX2 (Advanced Vector Extensions 2) instruction set, which operates on 128-bit or 256-bit registers. Unlike the former AVX, AVX2 has vector instructions to perform integer arithmetic operations and permutations of words (8 - 64 bits) within registers. Using such instructions allow us to implement SHA-3 exploiting the data level parallelism present; in Table 1 one can see the instructions used in our implementation.

*The authors were supported in part by the Intel Labs University Research Office.

Category	Instructions	Latency
Logical	`XOR`, `AND`, `ANDN`	1,1,1
Shift	`SHIFT`, `VSHIFT`	1,2
Permutation	`SHUFFLE`,`VPERM`	1,3
Merge	`UNPACK`, `VBLEND`, `PRBLEND`	1,1,3

Table 1. The main AVX2 instructions used in our implementation of SHA-3.

3. SHA-3

According to the draft FIPS 202 [FIPS 2014], SHA-3 family consists of six functions, four of them are hash functions and the others are extendable-output functions. The hash functions are SHA3-224, SHA3-256, SHA3-384 and SHA3-512 and the XOFs are SHAKE128 and SHAKE256, in Table 2 are shown the parameters of these functions.

An extendable-output function maps an arbitrary-length message producing a variable-length digest. This function can be used when an application requires a cryptographic hash function with a non-standard digest length. We note that the security of these special functions is directly related to the size of the digest.

The SHA-3 family shares a sponge construction structure, which is a simple iterated construction for building a function with variable-length input and arbitrary-length output based on a permutation f operating on a state of $r + c = 1,600$ bits [Bertoni et al. 2007]; the state can be visually represented as a 5×5 matrix of 64-bit words.

The permutation function is divided into five steps: θ, ρ, π, χ and ι; the following, a short description of these steps:

1. In the θ step is computed an XOR of each word of the state with the parity of the left column and the right column rotated one bit.
2. In the ρ step each word of the state is rotated a fixed amount of bits.
3. In the π step the words of the state are permuted.
4. In the χ step is processed a non-linear function between the elements of the same row.
5. In the ι step is computed an XOR between the first element of the state with a constant value.

4. Implementations

The SHA-3 algorithm processes a state of 25 words of 64 bits using a permutation function f, which is composed of some steps that can be vectorized. We stored the state among

Function	Bitrate (r)	Capacity (c)	Security Level
SHA3-224	1,152	448	112
SHA3-256	1,088	512	128
SHA3-384	832	768	192
SHA3-512	576	1,024	256
SHAKE128	1,344	256	$\min(N/2, 128)$
SHAKE256	1,088	512	$\min(N/2, 256)$

Table 2. SHA-3 parameters.

registers in different ways for each implementation.

4.1. Single message hash computation

We developed two implementations of hash function with a single message; in the first one we represent the state as 13 registers of 128 bits, this allow us to vectorize the steps θ, ρ and χ processing two words per instruction; in the other one, the state is represented as 7 registers of 256 bits, thus we can process four words per instruction. The step π can not be vectorized, but there is an AVX2 instruction to perform this permutation, however, this instructions is too expensive for registers of 256 bits, as we shown in Table 1.

These two implementations can also be adapted to the XOF functions, the only difference is that in the squeezing phase the computation is performed just on the last state produced after the absorbing phase, thus the computation is mainly performed in registers.

4.2. 4-way hash computation

In this implementation, the state is represented as 25 registers of 256 bits and the operations between registers are performed totally in parallel, achieving the computation of four digests. Here the π permutation is implemented faster than in the single message implementation, additionally we can compute four words per instruction in all the steps, thus giving a significant speedup; the only drawback of this implementation is the amount of registers needed to store the four states, because the Haswell micro-architecture has only 16 available registers.

5. Preliminary results

We benchmark our implementations on a Core-i7 4770 processor, following the guidelines on [Bernstein and Lange 2014]. In Figure 1 we show the cycles per byte to compute the digests from messages of size 4KB to 2GB.

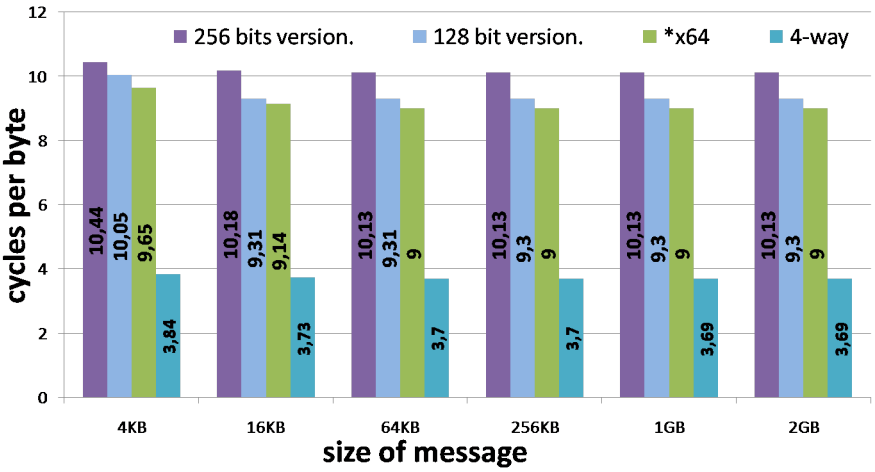

Figure 1. Cycles per byte to compute the digest from messages of size 4KB to 2GB.

The x64 implementation, developed by Ronny Van Keeris, is the fastest on eBASH for this processor. This implementation was optimized for 64 bits and does not use vector instructions.

	Absorbing	Squeezing
SHAKE128	7.79	7.7
SHAKE256	9.62	9.24

Table 3. Cycles per byte of SHAKE from a message of 4 KB.

Table 3 shows the cycles per byte to compute the absorbing and the squeezing phase in the XOFs implementations for a message of 4KB; as one can see the absorbing phase is more expensive than the squeezing phase, this happens because in the squeezing phase we do not need to process the message with the state, we only get the first r bits from state and then, process the state again through the function f until get all the digest required.

6. Conclusion

These preliminary results show that the use of vector instructions are useful for the efficient implementation of SHA-3. Using the AVX/AVX2 instructions allow us to achieve almost the same performance than the fastest 64-bit implementation for a single message setting. We observed that unlike the 64-bit implementation, the use of permutation instructions between vector registers is an expensive operation, since each permutation takes 3 clock cycles. For the 4-way setting, we obtained $2.5\times$ of speedup against the fastest single message implementation. This work is currently in progress and we are still looking for new optimization techniques to improve the results.

References

Bernstein, D. J. and Lange, T. (2014). ebacs: Ecrypt benchmarking of cryptographic systems.

Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. (2007). Sponge functions. In *ECRYPT hash workshop*, volume 2007. Citeseer.

Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. (2008). Keccak specifications. *Submission to NIST*, 42.

Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., and Jalby, W. (2005). Collisions of sha-0 and reduced sha-1. In *Advances in Cryptology–EUROCRYPT 2005*, pages 36–57. Springer.

FIPS, P. (2008). 180-3. *Secure Hash Standard*.

FIPS, P. (2014). 202. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*.

Indesteege, S., Mendel, F., Preneel, B., and Rechberger, C. (2009). Collisions and other non-random properties for step-reduced sha-256. In *Selected Areas in Cryptography*, pages 276–293. Springer.

NIST (2007). The sha-3 cryptographic hash algorithm competition.

Rijmen, V. and Oswald, E. (2005). Update on sha-1. In *Topics in Cryptology–CT-RSA 2005*, pages 58–71. Springer.

Wang, X., Yin, Y. L., and Yu, H. (2005). Finding collisions in the full sha-1. In *Advances in Cryptology–CRYPTO 2005*, pages 17–36. Springer.

