Multi-Stream Hashing on the PlayStation 3

Joppe W. Bos, Nathalie Casati, and Dag Arne Osvik

EPFL IC IIF LACAL, Station 14, CH-1015 Lausanne, Switzerland
{joppe.bos, nathalie.casati, dagarne.osvik}@epfl.ch

Abstract. With process technology providing more and more transistors per chip, still following Moore's "law", processor designers have used a number of techniques to make those transistors useful. Lately they have started placing multiple processor cores on each chip; an example is the Cell Broadband Engine, which serves as the heart of Sony's PlayStation 3 game console. We present high-performance multi-stream versions of cryptographic hash functions from the MD/SHA-family. Our implementations require 1.74, 3.51 and 8.18 cycles per byte per SPE when using the cryptographic hash functions MD5, SHA-1 and SHA-256 respectively. To the best of our knowledge these are the fastest implementations of these hash functions for the Cell processor. These implementations can be useful for cryptanalytic use as well as for utilizing the SPEs as cryptographic accelerators.

Keywords: Cell Broadband Engine, Cryptology, Hashing, Single Instruction Multiple Data (SIMD), Synergistic Processing Element (SPE)

1 Introduction

While the number of transistors on a single chip keeps increasing, according to Moore's "law" [12], the microprocessor industry is looking for ways to make these transistors useful. One way of achieving this is by placing multiple processor cores on each chip. An example of such a design is the Cell Broadband Engine (Cell).

Cryptographic hash functions are mainly used as a tool for authentication, i.e. they are part of many digital signature schemes and message authentication codes, but they can also be used as checksums to detect file corruptions, or to index data in hash tables. The MD family [10, 16, 17] of cryptographic hash functions designed by Ron Rivest, and the SHA (Secure Hash Algorithm) successors designed by the National Security Agency and the National Institute of Standards and Technology [13, 14] are still widely used, despite the fact that some are (partially) insecure.

The rest of the paper is organized as follows. In section 2 we give a brief overview of the Cell architecture. In section 3 we present techniques for maximizing throughput on the computational units of the Cell, together with a brief description of our target hash functions. In section 4 performance results for our multi-stream implementations are presented.

Notation We use the abbreviations G for 10^9 and Ki for 2^{10} [9], and combine this with b for bits and B for bytes. When presenting the hash functions we denote rotating a word x by y bits to the left as $RL(x, y)$, and rotating it to the right by y bits as $RR(x, y)$. The bitwise operations *and* and *xor* are denoted by $\wedge$ and $\oplus$, the bitwise *not*, or complement, of a word x is denoted by $\bar{x}$.

2 The Cell Broadband Engine

The Cell architecture [8] is equipped with one dual-threaded, 64-bit in-order "Power Processing Element" (PPE), which can offload work to the eight "Synergistic Processing Elements" (SPE) [6, 21]. The SPEs are the workhorses of the Cell processor. Each of them consists of a Synergistic Processing Unit (SPU), 256 KiB of private memory called Local Store (LS) and a Memory Flow Controller (MFC). The latter handles communication between each SPE and the rest of the machine, including main memory, as explicitly requested by programs. All code and data must fit within the LS if one wants to avoid the complexity of sending explicit DMA (Direct Memory Access) requests to the MFC.

Most SPU instructions are 128-bit wide SIMD (single instruction, multiple data) operations performing sixteen 8-bit, eight 16-bit, four 32-bit, or two 64-bit computations in parallel. Each SPU is also equipped with a large register file containing 128 registers of 128 bits each, providing space for unrolling and software pipelining of loops, hiding the relatively long latencies of its instructions. Unlike the PPU, the SPUs are asymmetric processors, having two pipelines which are designed to execute two disjoint sets of instructions. For some computations this means one pipe is busy all the time, while the other is mostly idle.

Like the PPU the SPUs are in-order processors. However, the SPUs have no hardware branch-prediction. Instead the programmer (or compiler) can tell the instruction fetch unit in advance where a (single) branch instruction will jump to. Hence, for most code with infrequent jumps and where the target of each branch can be computed sufficiently early, perfect branch prediction is possible.

One of the first applications of the Cell processor was to serve as the heart of Sony's PlayStation 3 (PS3) video game console. The Cell contains eight SPEs, and in the PS3 one of them is disabled, allowing improved yield in the manufacturing process as any chip with a single faulty SPE can still be used. One of the remaining SPEs is reserved by Sony's hypervisor, a software layer providing a virtual machine environment for running e.g. Linux. In the end we have access to six SPEs when running Linux on (the virtual machine on) the PS3. Fortunately the virtualization does not slow down programs running on the SPUs, as they are naturally isolated and protection mechanisms only need to deal with requests sent to the MFC.

3 Multi-Stream Hashing

In general, a cryptographic hash algorithm takes as input a message of arbitrary length and produces a message digest of fixed size as output. To achieve this the

hash algorithms goes, normally, through three different steps. This description holds for the hash functions discussed in this paper, i.e. following the Merkle-Damgård construction [4] using a one-way compression function. First we have an initialization phase setting up the internal state of the hash function and extending the message such that its size becomes a multiple of the length of the message block that the underlying compression function takes as input. The second step updates the state value for each block of the extended message using the deterministic compression function. Finally the third step computes and outputs the fixed size hash digest.

The focus of this article is on members of the MD and SHA family of cryptographic hash functions. Here, the message blocks are processed using a one-way compression function which is highly sequential, making it hard to fully exploit the capabilities of a processor when computing the hash value of a single data stream.

We can overcome this problem by interleaving hash computations for two or more streams of data. To be able to do this we also need enough space to store the working state of all our simultaneous streams, and the large register file available in the Cell's SPUs supports this. The hash functions considered in this paper are all designed with 32-bit operations, which also happen to be well supported by the SPU instruction set, executing four of these in parallel for each instruction. Hence the overall design of the SPUs is good for hashing some (small) multiple of four input streams in parallel.

In the following the compression function of some members of the MD and SHA family are recalled and analyzed for their implementation on the Cell, more specifically on the SPU architecture. Typically, not all the operations performed in a compression function are directly available in the instruction set of modern processors. Hence those operations need to be implemented using some sequence of other operations, increasing the time to compute the message digest. However, the SPU instruction set has good support for the basic operations used by the hash functions we consider. As an example, the majority operation on three operands can be expressed on the SPU using only two select (`selb`) instructions. In other processors this would typically be expressed with five (three `and` and two `xor`) binary boolean operations.

From this we see that an SPU is well equipped for performing multi-stream hashing, and such implementations can be useful when people want to use the SPUs as, for instance, cryptographic accelerators or for finding hash collisions [19, 18].

3.1 The MD5 Algorithm

The second to last member of the MD-family is the MD5 algorithm. It is still widely used, despite cryptographers considering it to be broken [23, 20]. The MD5 algorithm processes its input message in blocks of 512 bits, and produces a message digest of 128 bits. All its operations are performed on words of 32 bits. The compression function performs 64 steps, often referred to as 4 rounds of 16

steps each, of the following form:

$$(d, c, b, a) := (c, b, RL((a + f_i(b, c, d) + k_i + w_g), r_i), d)$$

where the k_i and the r_i are a fixed set of constants, the w_g are the words of the message (permuted and repeated 4 times). Each 16 steps a different step function f_i is used. These step functions are defined, for a given round number i, as

$$f_i(X, Y, Z) = \begin{cases} F(X, Y, Z) = (X \wedge Y) \oplus (\bar{X} \wedge Z) & \text{for } 0 \leq i < 16, \\ G(X, Y, Z) = (Z \wedge X) \oplus (\bar{Z} \wedge Y) & \text{for } 16 \leq i < 32, \\ H(X, Y, Z) = X \oplus Y \oplus Z & \text{for } 32 \leq i < 48, \\ I(X, Y, Z) = Y \oplus (X \vee \bar{Z}) & \text{for } 48 \leq i < 64. \end{cases}$$

The F and G functions can be implemented using a single select (`selb`) instruction on the SPU, while H and I require two instructions each.

3.2 The SHA-1 Algorithm

The second member of the SHA family of hash functions has been theoretically broken [22, 2] but remains a popular hash function. The SHA-1 algorithm uses a word size of 32 bits and produces a message digest of 160 bits. Just as with MD5 the input message m is processed in blocks of 512 bits after initial padding. All the individual blocks go through two parts, first each 512-bit block is expanded to 80 blocks of 32 bits. This message expansion is done by computing

$$w_i = \begin{cases} m_i & \text{if } 0 \leq i \leq 15, \\ RL((w_{i-3} \oplus w_{i-8} \oplus w_{i-14} \oplus w_{i-16}), 1) & \text{if } 16 \leq i \leq 79 \end{cases}$$

for each block. Here, w_i and m_i denote the i^{th} word of the expansion and the message respectively. Next, this expanded block goes through 80 rounds, one round for every word of the expansion block, of the following iteration formula:

$$(e, d, c, b, a) := (d, c, RL(b, 30), a, RL(a, 5) + f_i(b, c, d) + e + k_i + w_i)$$

where the k_i are a set of fixed values (one for every 20 rounds), and the f_i are defined as follows:

$$f_i(X, Y, Z) = \begin{cases} (X \wedge Y) \oplus (\bar{X} \wedge Z) & \text{for } 0 \leq i \leq 19, \\ X \oplus Y \oplus Z & \text{for } 20 \leq i \leq 39, \\ (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z) & \text{for } 40 \leq i < 59, \\ X \oplus Y \oplus Z & \text{for } 60 \leq i < 79. \end{cases}$$

Note that for rounds 0 through 19 the f_i perform a select operation (bitwise "if X then Y else Z"), and for rounds 40 through 59 they compute the majority function.

3.3 The SHA-256 Algorithm

The SHA-256 algorithm is one of the four hash functions of the SHA family published after SHA-1, collectively known as SHA-2. Cryptanalysis over the last years [7, 11] has not revealed any weaknesses. However, researchers are wary due to SHA-2 having the same structure as the other members of its family.

SHA-256 is designed for a word size of 32 bits, and processes its message input in blocks of 512 bits. Just as for SHA-1 these blocks are extended after padding, but this time to 64 blocks of 32 bits. The words $w_0 \dots w_{15}$ remain the same. For $w_{16} \dots w_{63}$ we have

$$\begin{aligned} s_0 &:= RR(w_{i-15}, 7) \oplus RR(w_{i-15}, 18) \oplus RR(w_{i-15}, 3) \\ s_1 &:= RR(w_{i-2}, 17) \oplus RR(w_{i-2}, 19) \oplus RR(w_{i-2}, 10) \\ w_i &:= w_{i-16} + s_0 + w_{i-7} + s_1 \end{aligned}$$

Next this extended message block goes through 64 rounds of the following computations:

$$\begin{aligned} s_0 &:= RR(a, 2) \oplus RR(a, 13) \oplus RR(a, 22) \\ maj &:= (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c) \\ t_2 &:= s_0 + maj \\ s_1 &:= RR(e, 6) \oplus RR(e, 11) \oplus RR(e, 25) \\ ch &:= (a \wedge b) \oplus (\bar{a} \wedge c) \\ t_1 &:= h + s_1 + ch + k_i + w_i \\ (h, g, f, e, d, c, b, a) &:= (g, f, e, d + t_1, c, b, a, t1 + t2) \end{aligned}$$

where maj and ch represent the majority and select operations respectively.

4 Results

Multi-stream, high-throughput versions of MD5, SHA-1 and SHA-256 have been implemented for the SPU architecture. To achieve this we started either from scratch or used existing free C implementations as a starting point. The compression function, the most time consuming part of each algorithm, is implemented in assembly code carefully optimized for the SPU architecture.

We are not aware of any similar implementations for PC processors. For the sake of comparison we state performance details obtained from the ECRYPT Benchmarking of Cryptographic Systems (eBACS) project [1]. Our results are obtained when running on one SPU inside a PS3. All the multi-stream versions require input streams of equal length; if needed the code can be modified to handle multiple input streams of different lengths with slightly reduced performance.

The MD5 algorithm. The compression function of the MD5 algorithm offers very little opportunities for parallelism. In order to obtain maximum throughput the implementation from the cryptographic library XySSL [5] was transformed

Arch	Details	Results from	MD5	SHA-1	SHA-256
x86	3000MHz Intel Pentium 4	eBACS [1]	5.04 (1)	31.73 (1)	47.18 (1)
x86-64	2000MHz Intel Core 2 Duo T7300	eBACS	5.48 (1)	13.63 (1)	20.31 (1)
ppc32	2000MHz IBM PowerPC G5 970	eBACS	16.52 (1)	16.79 (1)	20.86 (1)
ppc64	1900MHz IBM POWER5	eBACS	12.42 (1)	11.79 (1)	21.85 (1)
SPU	3192MHz Sony PlayStation 3	IBM [3]	10.46 (1)	12.10 (1)	29.98 (1)
SPU	3192MHz Sony PlayStation 3	This article	1.74 (8)	3.51 (8)	8.18 (4)

Table 1. *Performance comparison, in cycles per byte, of software implementations of MD5, SHA-1 and SHA-256. The number of streams hashed in parallel is in parentheses.*

into an eight-stream version, and the compression function replaced with our own assembly code. Due to the lack of parallelism in MD5, Hashing of the eight streams is performed as two interleaved sets of 4-way SIMD computations, hiding the SPU's minimum latency of two cycles per instruction.

The SHA-1 algorithm. The SHA-1 compression function is implemented from scratch. Most of the instructions go to the even pipeline, and only a few, like loading the hash state and message, can be performed by the odd pipeline. Like for MD5, the SHA-1 compression function offers little parallelism, and again we need two interleaved sets of 4-way SIMD computations in order to hide the instruction latencies.

The SHA-256 algorithm. Our implementation of the SHA-256 compression function is based on code from the OpenSSL software suite [15]. Unlike the other algorithms, instruction scheduling is made easier by the fact that the SHA-256 compression function offers some parallelism. Hence, only one set of 4-way SIMD computations is needed to fully exploit the computing power of the SPUs.

4.1 Discussion

Performance results obtained with our multi-stream implementations are presented in Table 1. Results are expressed in cycles per byte. The x86, x86-64, ppc32 and ppc64 numbers are medians of results obtained when hashing long single stream messages, as reported on eBACS [1]. The single-stream SPU results are obtained from [3]. The new SPU results are those reported in this paper, running our multi-stream implementations on long messages. When running on all 6 SPUs of a PS3, our implementations reach speeds of 88, 43.6 and 18.7 gigabits per second, for MD5, SHA-1 and SHA-256. Note that the PPU is not included in this calculation, since its architecture is different from (though somewhat similar to) the SPU architecture, and the throughput of a PS3 will be higher when utilizing this processor as well. The results in Table 1 are stated per core. Most modern workstations have two or four cores, while a PS3 has six available SPUs and a Cell blade server has eight SPUs available per Cell processor.

Obviously, comparing single with multiple stream implementations is not fair. The results in Table 1 are given to emphasize the performance one can achieve on the SPU architecture. Still, our multi-stream implementations are up to an order of magnitude faster than the single-stream implementations.

5 Conclusion

We show that the Cell processor, present in the PlayStation 3 game console, can be used as a high-throughput multi-stream hashing machine for the widely used MD5, SHA-1 and SHA-256 cryptographic hash algorithms. By carefully implementing assembly versions of the compression functions for these algorithms, we demonstrate that the SIMD architecture of the SPE with its rich instruction set allows us to implement efficient multi-stream versions. Performance of 1.74, 3.51 and 8.18 cycles per byte is achieved for MD5, SHA-1 and SHA-256 respectively. This illustrates that the SPEs can be very useful as cryptographic accelerators or for cryptanalytic use like searching for hash collisions.

References

1. D. J. Bernstein and T. Lange, (editors). eBACS: ECRYPT Benchmarking of Cryptographic Systems. http://bench.cr.yp.to, accessed 1 April 2009.
2. C. D. Cannière and C. Rechberger. Finding SHA-1 characteristics: General results and applications. In *Asiacrypt 2006*, volume 4284 of *LNCS*, pages 1–20, 2006.
3. T. Chen, R. Raghavan, J. Dale, and E. Iwata. Cell broadband engine architecture and its first implementation: A performance view. http://www.ibm.com/developerworks/power/library/pa-cellperf/, November 2005.
4. I. Damgård. A design principle for hash functions. In *Crypto 1989*, volume 435 of *LNCS*, pages 416–427, 1989.
5. C. Devine. XySSL. http://www.xyssl.org/.
6. B. Flachs, S. Asano, S. Dhong, P. Hofstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Leenstra, J. Liberty, B. Michael, H. Oh, S. M. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, and N. Yano. A streaming processor unit for a Cell processor. *IEEE International Solid-State Circuits Conference*, pages 134–135, February 2005.
7. H. Gilbert and H. Handschuh. Security analysis of SHA-256 and sisters. In *Selected Areas in Cryptography*, volume 3006 of *LNCS*, pages 175–193, 2003.
8. H. P. Hofstee. Power efficient processor architecture and the Cell processor. In *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 258–262, 2005.
9. IEC. Letter symbols to be used in electrical technology - Part 2: Telecommunications and electronics. Technical Report 60027-2, International Electrotechnical Commission, 2000.
10. B. Kaliski. The MD2 message-digest algorithm. RFC 1319, IETF, http://www.ietf.org/rfc/rfc1319.txt, April 1992.
11. F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. Analysis of step-reduced SHA-256. In *FSE 2006*, volume 4047 of *LNCS*, pages 126–143, 2006.

12. G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38:8, 1965.
13. National Institute of Standards and Technology. Secure hash standard. FIPS 180-1, NIST, http://www.itl.nist.gov/fipspubs/fip180-1.htm, April 1995.
14. National Institute of Standards and Technology. Secure hash standard. FIPS 180-2, NIST, http://www.itl.nist.gov/fipspubs/fip180-2.htm, August 2002.
15. OpenSSL. The open source toolkit for SSL/TLS. http://www.openssl.org/.
16. R. Rivest. The MD4 message-digest algorithm. RFC 1320, IETF, http://www.ietf.org/rfc/rfc1320.txt, April 1992.
17. R. Rivest. The MD5 message-digest algorithm. RFC 1321, IETF, http://www.ietf.org/rfc/rfc1321.txt, April 1992.
18. M. Stevens, A. K. Lenstra, and B. de Weger. Predicting the winner of the 2008 US presidential elections using a Sony PlayStation 3. http://www.win.tue.nl/hashclash/Nostradamus/.
19. M. Stevens, A. K. Lenstra, and B. de Weger. Chosen-prefix collisions for MD5 and Colliding X.509 certificates for different identities. In *Eurocrypt 2007*, volume 4515 of *LNCS*, pages 1–22, 2007.
20. M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In *Crypto 2009*, volume 5677 of *LNCS*, pages 55–69, 2009.
21. O. Takahashi, R. Cook, S. Cottier, S. H. Dhong, B. Flachs, K. Hirairi, A. Kawasumi, H. Murakami, H. Noro, H. Oh, S. Onish, J. Pille, and J. Silberman. The circuit design of the synergistic processor element of a Cell processor. In *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 111–117, Washington, DC, USA, 2005. IEEE Computer Society.
22. X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *Crypto 2005*, volume 3621 of *LNCS*, pages 17–36, 2005.
23. X. Wang and H. Yu. How to break MD5 and other hash functions. In *Eurocrypt 2005*, volume 3494 of *LNCS*, pages 19–35, 2005.