Multi-Stream Hashing on the PlayStation 3

Joppe Bos, Nathalie Casati and Dag Arne Osvik

Laboratory for Cryptologic Algorithms,
École Polytechnique Fédérale de Lausanne,
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 (PS3) game console. We present high-performance multi-stream versions of cryptographic hash functions from the MD/SHA-family. As an example, our MD5 implementation is capable of handling 88.17 Gb/s per PS3. To the best of our knowledge this is the fastest implementation of MD5 for the Cell. These implementations can be useful for cryptanalytic use and also for using processing cores as cryptographic accelerators.

Key words: Cell Broadband Engine, Cryptology, Hashing, Single Instruction Multiple Data (SIMD), Synergistic Processor Element (SPE)

1 Introduction

While the number of transistors on a single chip keeps increasing, according to Moore's "law", 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) which contains one dual-threaded PowerPC (the Power Processing Element, or PPE) and a number of "Synergistic Processor Elements" (SPEs).

The MD family ([1], [2], [3]) of cryptographic hash functions by R. Rivest, and the successor designed by the National Security Agency called the Secure Hash Algorithm (SHA) ([4], [5]) are still, despite the fact that some are (partially) insecure, widely used. We show how to use the Cell architecture to turn the PlayStation 3 into a high-throughput hashing machine.

The rest of the paper is organized as follows. In section 2 a brief overview of the Cell is given. In section 3 techniques for maximizing throughput on the SPEs are presented. In section 4 initial results of our multi-stream implementations are given.

Notation We use the abbreviations G for 10^9 and Ki for 2^{10}, and combine this with b for bits and B for bytes.

2 The Cell Broadband Engine

The Cell architecture is equipped with a dual-threaded, dual-issue 64-bit in-order PPE, which can offload work to the eight SPEs ([6], [7]). Both the PPE and SPEs support SIMD instructions. The SPEs are the workhorses of the Cell and consist of a Synergistic Processor Unit (SPU) and a Memory Flow Controller (MFC). Every SPU has a large register file containing 128 registers of 128 bits each, and also a Local Store (LS) of 256 KiB. Access from the SPUs to main memory is through explicit Direct Memory Access (DMA) requests to the MFC. The executable itself and all required data must fit in the LS if one wants to avoid DMA requests.

Like the PPE the SPUs are dual-issue in-order processors. However, the SPEs have no hardware branch-prediction. Instead the programmer (or compiler) can tell in advance where a (single) branch instruction will jump to. Hence, for most straight-line code where the targets of the branches 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, so in the end we have access to six SPEs when running Linux on the PS3.

3 Multi-stream hashing

When designing a high-throughput implementation of a cryptographic hash function from the MD/SHA family for the Cell architecture, one should exploit the SIMD properties of the SPEs. These hash functions are designed to run fast on architectures with some fixed word size. For all of these, except SHA-384 and SHA-512, this word size is at most 32 bits; this means the SPEs are perfect for handling some small multiple of four input streams in parallel. In general a hash algorithm takes as input a message of arbitrary length and produces as output a message digest of a fixed number of bits.

The hash algorithms extend a message with padding and the size of the message so that the resulting length becomes a multiple of the hash function's block size. Starting from a given initial state, they then update the state for each block using a compression function. This compression function is highly sequential, and in combination with the SPU's minimum instruction latency of two cycles this means we need to interleave computations for two sets of streams per SPE to fully utilize the processor.

Multi-stream implementations can be useful when people want to use the different SPEs as, for instance, cryptographic accelerators.

3.1 The MD5 algorithm

The latest member of the MD-family is the MD5 algorithm. It is designed for a word size of 32 bits and produces a message digest of 128 bits. The input message is processed in blocks of 512 bits. Each block goes through 64 rounds, every round uses a different function which takes as input three words and outputs one word. These functions are defined as

```
F(X,Y,Z) := (X and Y) or (not(X) and Z)
G(X,Y,Z) := (Z and X) or (not(Z) and Y)
H(X,Y,Z) := X xor Y xor Z
I(X,Y,Z) := Y xor (X or not(Z))
```

The SPU instruction set contains all possible two-input boolean operations, as well as the trinary select operation ("if a then b else c"); a bitwise version of the $a \mathbin{?} b : c$ operation in C. The F and G are both selects, which require just a single select (`selb`) instruction on the SPU, while H and I require two instructions each.

3.2 The SHA-1 algorithm

The SHA-1 algorithm is designed for word sizes of 32 bits and produces a message digest of 160 bits. Just as with MD5 the input message is processed in blocks of 512 bits after padding. These blocks have to go through two parts, first the 512 bit block gets expanded to 2560 bits. This is done by applying $w_i = \text{leftrotate}\,((w_{i-3} \text{ xor } w_{i-8} \text{ xor } w_{i-14} \text{ xor } w_{i-16}), 1)$, where w_i is the i^{th} word for $16 \leq i \leq 79$. Next this larger block must go through 80 rounds which use similar functions as in MD5 and update the current state.

4 Results

We have benchmarked eight-stream MD5 on the SPU using C and assembler versions of eight-stream single-block compression. The C version is derived from Cristophe Devine's free implementation and extended with eight-block compression. Our assembler version then implements the same interface, and achieves about six times the performance of the C code. We are not aware of any similar implementations for PC processors, but can give a very rough and optimistic estimate for Intel's Core2Quad running at 2.4 GHz. This processor can be used to build a machine with price comparable to the PS3, and with the highly unrealistic assumptions of no register allocation nor scheduling problems, the upper bound on its speed is about 64 Gbit/s.

Results for various message sizes are given in Fig. 1. Further speedup will be gained by compressing multiple blocks per stream for each function call.

We are also working on eight-stream implementations of SHA-1 and SHA-256, and will include results for these in the final version of this paper.

Message size in bytes	Amount of messages	Time(s)		Gb/s per SPE		Gb/s per PS3	
		C	ASM	C	ASM	C	ASM
128	10^7	6.36	3.33	1.61	3.08	9.66	18.45
256	10^7	9.13	4.02	2.24	5.09	13.46	30.57
512	10^7	14.69	5.41	2.79	7.57	16.73	45.43
1024	10^7	25.79	8.19	3.18	10.00	19.06	60.01
10240	10^6	22.58	5.83	3.63	14.05	21.77	84.31
102400	10^6	222.53	55.84	3.68	14.67	22.09	88.02
153600	10^6	333.62	83.62	3.68	14.70	22.10	88.17

Fig. 1. Time and throughput results when running the C and ASM versions of the eight-stream MD5 algorithm on a PS3 with different message sizes.

References

1. Kaliski, B.: The MD2 Message-Digest Algorithm. RFC 1319 (April 1992)
2. Rivest, R.: The MD4 Message-Digest Algorithm. RFC 1320 (April 1992)
3. Rivest, R.: The MD5 Message-Digest Algorithm. RFC 1321 (April 1992)
4. National Institute of Standards and Technology: Secure hash standard. FIPS 180-1, NIST (April 1995)
5. National Institute of Standards and Technology: Secure hash standard. FIPS 180-2, NIST (August 2002)
6. Flachs, B., Asano, S., Dhong, S.H., et al.: A streaming processor unit for a cell processor. ISSCC Dig. Tech. Papers (February 2005) 134–135
7. Takahashi, O., Cook, R., Cottier, S., et al.: 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, Washington, DC, USA, IEEE Computer Society (2005) 111–117