

Performance Evaluation of Block Encryption Algorithms on Core 2

Authors: Junko Nakajima* and Mitsuru Matsui*

1. Introduction

This paper presents the high-speed software encryption technique, "bit-slice implementation," on Intel's new Core 2 processor, which is its first micro-architecture platform. Until now, the performance of leading-edge processors has been evaluated and compared by implementing encryption algorithms and various benchmark tests. From the standpoint of encryption implementation, it is important to study the best implementation method together with platform migration because the execution speed of the same program can change completely depending on the architecture.

In 1997, Biham proposed bit-slice implementation, where multiple blocks are processed in parallel by assuming a single software instruction with n-bit-long registers as n-sets of hardware logic gates. Therefore, when this method is used to implement an encryption algorithm on a small-sized hardware and target processor with many registers, faster speed is expected. This method has an additional advantage of providing security against side-channel attacks such as cache attacks because there is no need to refer to the tables depending on the key value.

Until now, bit-slice implementation has demonstrated its effectiveness on reduced instruction set computer (RISC) processors, whereas PC (x86) processors are considered to be unsuitable for this implementation because of the increased frequency of memory access due to few available registers, resulting in a bottleneck to the execution speed. This time, we focused on the new feature of the Core 2, that is, the significantly enhanced single instruction multiple data (SIMD) instructions compared to the Pentium 4 and Athlon 64, which is expected to improve the performance of bit-slice implementation. Consequently, we studied the software implementation and speed-up techniques of block encryption schemes (MISTY, KASUMI, AES, and Camellia) taking into account the processor architecture in a 128-bit environment.

2. Core 2 Architecture

To reduce power consumption, the design concept of the Core 2's architecture was shifted to increasing scalability rather than frequency, resulting in 14 pipeline stages. This number is less than half that of the conventional Pentium 4 core (Prescott core). Decoding is

now performed online, similar to the Pentium III, not using the trace cache. The Core 2 has also enhanced the fusion functions used in the Pentium M, and fused μ -ops have become quite similar to the macro-operation of the Athlon 64.

Table 1 shows a listing of instruction latency and throughput obtained by experiment. It is evident from this table that the throughput of logic operation instructions, which was below 2 μ -ops/cycle with the x64 instructions of the Pentium 4, is now improved to 3 μ -ops/cycle with the Core 2; and the latencies of right shift and left/right rotate instructions, which were extremely long with the Pentium 4, are now much improved with the Core 2. In addition, for the XMM instructions, throughputs of register-to-register move and logic operation instructions are improved from 1 to 3; and the latencies are also improved, achieving nearly the same performance as x64 instructions. In contrast, the greatest performance bottleneck for the Core 2 seems to be caused by the instruction fetch being limited to 16 bytes per cycle. To speed up a program, certain techniques are required, such as making the instruction length as short as possible.

Table 1 Instruction latency and throughput

Processor	Pentium4 561	Athlon64 3500+	Core2 Duo E6400
Operand type	64-bit general registers		
mov reg,[mem]	4, 1	3, 2	3, 1
mov reg,reg	1, 3	1, 3	1, 3
add reg,reg	1,2.88	1, 3	1, 3
xor/and/or reg,reg	1, 7/4	1, 3	1, 3
shr reg,imm	7, 1	1, 3	1, 2
shl reg,imm	1, 7/4	1, 3	1, 2
ror/rol reg,imm	7, 1/7	1, 3	1, 1
Operand type	128-bit XMM registers		
movaps xmm,[mem]	-, 1	-, 1	-, 1
movaps xmm, xmm	7, 1	2, 1	1, 3
paddb/w/d xmm,xmm	2, 1/2	2, 1	1, 2
paddq xmm,xmm	5, 2/5	2, 1	1, 1
xorps/andps xmm,xmm	2, 1/2	2, 1	1, 3
psllw/d/q xmm,imm	2, 2/5	2, 1	2, 1
pslldq xmm,imm	4, 2/5	2, 1	2, 1
punpcklbw/wd/dq xmm,xmm	2, 1/2	2, 1	4, 1/2
punpcklqdq xmm,xmm	3, 1/2	1, 1	1, 1
pmovmskb reg,xmm	-, 1/2	-, 1	-, 1

3. MISTY and KASUMI

KASUMI is a 64-bit block encryption algorithm used as the standard in the Universal Mobile Telecommunications System (UMTS) / Global System for Mobile

Communications (GSM). The design of KASUMI is based on MISTY and because its structure is suited for hardware, faster speed by using bit-slice implementation can be expected. The left side of Fig. 1 shows the FI function of KASUMI, and the right-hand side shows that the number of instructions required by the FI function can be reduced by an equivalent transformation. Both MISTY and KASUMI have two kinds of S-boxes, S7 and S9, within the dominant inner function FI. While MISTY has three S-boxes (S7 – S9 – S7), KASUMI has two of each kind, for a total of four S-boxes, resulting in a difference in the total number of table references (KASUMI = 96, MISTY = 76), which becomes the defining factor for KASUMI's slower encryption speed compared to MISTY. On the contrary, KASUMI's key scheduling time is much shorter than the encryption time because bit-slice implementation does not require any shift/rotate operations. Bit-slice implementation of KASUMI provides an encryption speed 4 times faster and key scheduling speed at least 30 times faster than with normal implementation (Table 2).

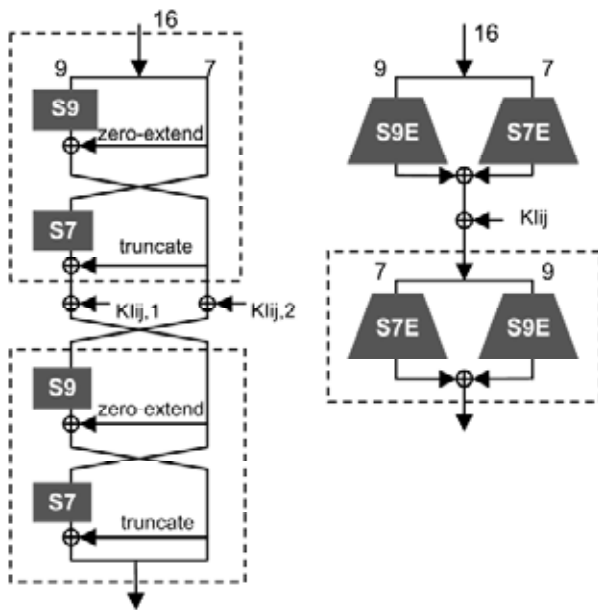


Fig. 1 Equivalent forms of the FI function of KASUMI

Table 2 Implementation performance of KASUMI and MISTY1

Processor	Pentium4		Athlon64		Core2	
KASUMI						
Style	BS	Sing	BS	Sing	BS	Sing
Cy/blk	241	300	241	272	74	290
Cy/byte	30.1	37.5	30.1	34.0	9.25	36.3
Inst/cycle	0.71	1.69	0.71	1.86	2.31	1.75
Cy/Keysch	8	104	7	64	2	78
MISTY1						
Style	BS	Sing	BS	Sing	BS	Sing
Cy/block	185	234	195	203	59	214
Cy/byte	23.1	29.3	24.4	25.4	7.38	26.8
Inst/cycle	0.72	1.82	0.66	2.10	2.26	1.99
Cy/Keysch	57	244	57	240	16	178

4. AES and Camellia

Both AES and Camellia use linearly equivalent S-boxes for the inversion functions of GF (2⁸). The number of instructions for this calculation has a dominant influence on the overall calculation speed. From the standpoint of bit-slice implementation, the smallest S-box we are aware of uses subfields, where the inversion functions of GF (2⁸) are configured by the operations of GF (2⁴), and the operations of GF (2⁴) is in turn configured by the operations of GF (2²)^[5]. In the case of bit-slice, there are approx. 200 instructions for one S-box, and considering that AES and Camellia respectively refer to the S-box 160 and 144 times per one block, the total processing amount for the S-box, e.g. for 128 blocks of parallel processing, would be 250 and 225 instructions per one block, respectively, which accounts for approx. 70% of the overall instructions. A performance comparison between the Core 2 and Pentium 4 in normal implementation shows that the Core 2 requires a slightly smaller cycle count per block, but a comparison with the Athlon 64 still shows a significant performance difference.

The reason for the difference is that the Athlon 64 can perform 64-bit memory read operations twice per cycle compared to only once per cycle for the Core 2. In contrast, in the case of bit-slice implementation using 128-bit XMM registers, the effect of enhanced SIMD instructions for the Core 2 becomes evident. This implementation provides more than 50% faster speed compared to that with normal implementation, and performance that is twice as fast as that with bit-slice implementation using 64-bit general registers, which directly reflects the register size being doubled from 64 bits to 128 bits. Considering that the speed of the Pentium 4 and Athlon 64 with bit-slice implementation is 50% slower than with normal implementation, the performance improvement of the Core 2 SIMD instructions is remarkable. Until now, Camellia's performance has never approached that of AES on any platform in normal implementation, but the implementation results for Camellia show that with the two-block parallel imple-

Table 3 Implementation performance of AES and Camellia with 128-bit key

Processor	Pentium4		Athlon64		Core2	
AES						
Style	BS	Sing	BS	Sing	BS	Sing
Cy/blk	491	256	560	170	147	232
Cy/byte	30.7	18.0	35.0	10.6	9.19	14.5
Inst/cycle	0.80	1.18	0.70	2.74	2.66	2.00
Camellia						
Style	BS	Doubl	BS	Doubl	BS	Doubl
Cy/blk	467	457	510	175	135	208
Cy/byte	29.2	28.6	31.9	10.9	8.44	13.0
Inst/cycle	0.72	0.94	0.65	2.46	2.47	2.07
Format conversion						
Cy/blk	41.5		28.1		15.4	
Cy/byte	2.59		1.76		0.96	
Inst/cycle	0.72		1.06		1.96	

mentation on the Core 2, Camellia (Table 3; Double) surpasses AES (Single). With bit-slice implementation, Camellia is faster than AES on any processor, which is mostly attributed to the fact that Camellia has 16 units fewer S-boxes.

Bit-slice operations are performed in special data format, which necessitates format conversion to maintain compatibility. This conversion requires the repositioning of all data bits and hence considerable computation time, which was one of the factors hindering practical application of the bit-slice technique. This time, bit-slice implementation on the Core 2 has achieved the performance of one cycle or less per byte, which is much faster than with normal implementation even when pre- and post-format-conversion time is added.

5. Conclusion

In this paper, the superior performance of Intel's latest processor, Core 2, in particular its SIMD instructions, was described from the viewpoint of bit-slice implementation of block encryption algorithms. It was clearly shown that AES performance with bit-slice implementation on a PC processor is higher than that with normal implementation. This fact, together with the security advantage of the bit-slice technique against implementation attacks such as cache attacks, may create a turning point in discussions on the application of public key cryptography (such as usage mode).

References

- (1) M. Matsui: New encryption algorithm MISTY (1997)
- (2) 3GPP TS 35.202 v6.1.0: 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 2: KASUMI Specification (2005)
- (3) K. Aoki, et al.: The 128-Bit Block Cipher Camellia (2002)
- (4) Federal Information Processing Standards Publication 197: Advanced Encryption Standard (2002)
- (5) M. Matsui: How Far Can We Go on the x64 Processors? (2006)