IDEA: A Cipher for Multimedia Architectures?

Helger Lipmaa

Küberneetika AS, Akadeemia 21, 12617 Tallinn, Estonia
helger@cyber.ee

Abstract. MMX is a new technology to accelerate multimedia applications on Pentium processors. We report an implementation of IDEA on a Pentium MMX that is 1.65 times faster than any previously known implementation on the Pentium. By parallelizing four IDEA's we reach an unprecedented 78 Mbits/s throughput per output block on a 166MHz MMX. In the light of rapidly increasing popularity of multimedia applications, causing more dedicated hardware to be built, and observing that most of the current block ciphers do not benefit from MMX, we raise the problem of designing block ciphers (and encryption modes) fully utilizing the basic operations of multimedia.
Keywords: block ciphers, fast implementations, IDEA, multimedia architectures, Pentium MMX.

1 Introduction

The second main objective besides security in designing cryptographic primitives is speed: even 10% difference in speed (by the same security level) may bias industry to prefer one cipher to another. Still, it is not an easy task to compare ciphers by virtue of speed. The reasons are manifold, depending on the human factor (the best known implementation may not be the best possible implementation) but also on the hardware available: ciphers optimized for 32-bit processors may not be optimal on 64-bit processors and vice versa. Application of new microprocessor techniques (DSP — Digital Signal Processing, VLIW – Very Long Instruction Word, SIMD — Single Instruction Multiple Data) in current general-purpose microprocessors will significantly sway our beliefs in the speed ratio of available ciphers [Cla97].

Because of the quickly increasing importance of multimedia, dedicated hardware will be commonplace tomorrow. Today's multimedia extensions (to name a few, Intel's MMX, Sun's VIS, HP's MAX-2, Cyrix's MMX, AMD's 3DNow!) are just the first flowers. New generations of multimedia enhanced processors will even more change our judgment of what it means to be "software" optimized.

MMX, incorporated in every new Intel processor (e.g., in the Pentium with MMX and the Pentium II), is a relatively new extension made to accelerate multimedia applications. Considering the worldwide spread of MMX capable computers, design and implementation of cryptographic primitives utilizing the basic operations of multimedia applications should be considered very seriously. Some work in this area has already been done by designing new hash functions

S. Tavares and H. Meijer (Eds.): SAC'98, LNCS 1556, pp. 248–263, 1999.

and stream ciphers [HK97,Cla97,DC98]. Biham viewed a 64-bit processor as a SIMD parallel computer, which can compute 64 one-bit operations simultaneously, getting significant acceleration of DES [Bih97]. Using the same method ("*bit-slicing*"), several papers [SAM97,Kwa98] have later improved Biham's results.

There is a wide variety of block ciphers in more or less general use. The popularity of some of those ciphers is based on the trust in the design of the cipher, the popularity of some other ciphers is based on the high throughput in combination with reasonable security. In particular, the block cipher IDEA [LM90,LMM91] is believed to be very secure due to the proper interaction between three different group operations. Although, apart from DES, IDEA seems to be the most studied block cipher, no currently known attack (e.g., [BKR97], [DGV94] or [Haw98]) against the full IDEA performs better than exhaustive search. Interaction between three different group operations adds confidence in IDEA's security, but the frequent use of multiplication does not allow fast software implementations on common microprocessors (Table 1).

Block cipher	Block size	Cycles	Mbits/s
Square	128	244	87.1
Blowfish	64	158	67.2
RC5-32/16	64	199	53.4
CAST5	64	220	48.3
DES	64	340	31.2
SAFER (S)K-128	64	418	25.4
Shark	64	585	18.2
IDEA	64	590	18.0
3DES	64	928	11.4

Table 1. Performance in clock cycles per block of output and Mbits/s of several block ciphers on a 166MHz Pentium by Antoon Bosselaers [PRB98].

We describe an implementation of IDEA on MMX, that is significantly faster than the best *possible* implementation of IDEA on the standard Pentium. One attempt to optimize IDEA on MMX has already been taken: Masayasu Kumagai's implementation of non-standard IDEA [Kum97] encrypts three IDEA blocks in parallel, achieving 45.6 Mbits/s per individual encryption on a 200MHz Pentium MMX. Our implementation includes a fast version of standard IDEA and a parallel version that is about twice as fast as Kumagai's.

The MMX architecture was chosen for it being the *de facto* standard, IDEA was chosen because no other current "industry-standard" block cipher seems to benefit from the Pentium MMX and because of its practical importance. Moreover, in the following we demonstrate that IDEA utilizes only about one third of the Pentium MMX and is, additionally, easily parallelized without a significant parallelization overhead. The resulting parallel "4-way IDEA" is faster than any of the 64-bit block ciphers in Table 1; by doing this we transform a relative slow (and as generally believed, a *very* secure) cipher into a very fast (and still *very* secure) cipher. Observing that, we raise a question of designing new, multimedia optimized block ciphers.

Section 2 gives a background to MMX and multimedia extensions. Section 3 outlines the basics of the IDEA algorithm. Section 4 describes our implementation of IDEA on MMX. Section 5 describes shortly the fast parallel implementation of IDEA. Section 6 takes a more broad view of multimedia architectures and Sect. 7 gives a short description of "why can't most of the block ciphers be parallelized on the MMX" and raises the problem of designing new, multimedia-like constituted block ciphers. In Sect. 8 we outline the results and finally, Sect. 9 acknowledges the people who have to be acknowledged.

2 Introduction to MMX

At the time of writing this paper Intel's Pentium was the most widely used general purpose processor. We shall not present a detailed outline of Intel Pentium's architecture (an interested reader may turn to [Int97b] or [BGV96]).

MMX (MultiMedia eXtensions) is a relatively new technology to enhance performance of advanced media and communication applications. The MMX technology introduces new general-purpose instructions that operate in parallel on multiple data elements packed into 64-bit quantities (the 'SWAR' — SIMD Within A Register — architecture, [Die97]). These instructions accelerate the performance of multimedia applications such as motion video, combined graphics with video, image processing, audio synthesis, speech synthesis and compression, telephony, video conferencing, 2D graphics, and 3D graphics. These applications were broken down to identify the most compute-intensive routines, which were then analyzed in detail using advanced computer-aided engineering tools. The results of this extensive analysis showed many common, fundamental characteristics across these diverse software categories. The key attributes of these applications were:

- Small integer data types (for example: 8-bit graphics pixels, 16-bit audio samples).
- Small, highly repetitive loops.
- Frequent multiplies and accumulates.
- Compute-intensive algorithms.
- Highly parallel operations.

The new MMX instructions work on 8 new 64-bit registers called `%mm0` … `%mm7`. Some of the instructions have an 8-way parallel 8-bit, a 4-way parallel 16-bit, a 2-way parallel 32-bit and a 64-bit version but most of the operations (like multiplication and addition) have only versions corresponding to some subset of these possibilities. There are more operations for 8-bit and 16-bit data than for larger data types (the "small data types" paradigm).

All microprocessors in the Pentium family have another level of parallelism, called *super-scalar parallelism.* In particular, most of the MMX instructions can be executed in both U and V pipelines (in parallel with any other instruction), with the following exceptions.

– Multiplication requires three cycles (has *latency* 3) but can be pipelined, resulting in one multiplication operation every clock cycle (has *throughput* 1). Multiplication instructions cannot pair with other multiplication instructions.
– Shift, pack and unpack instructions cannot pair with each other.
– MMX instructions that access memory or integer registers can only execute in the U-pipe and cannot be paired with any instructions that are not MMX instructions.
– After updating an MMX register, one additional clock cycle must pass before that MMX register can be moved to either memory or to an integer register.

Throughput is 1 for every operation, latency is 1 for every operation but multiplication. It is important to understand the difference between the SIMD-parallelism provided by the MMX technology and the super-scalar parallelism. The first permits to execute *the same* operation on up to eight different data entities as one instruction, the second makes it possible to execute two *possibly different* instructions during the same machine cycle. Hence, the total level of parallelism inside a Pentium MMX can be up to 16.

Still, most of the applications do not benefit from MMX. Some of the limitations of MMX (and the Pentium family in general) are outlined below (cf [Int97b,Int97a] for more information):

Maximum two operands. Pentium/MMX instructions have the maximum of two operands, causing a high frequency of the move (`movq`) instructions in Pentium/MMX programs.

Lack of registers. There are only 8 MMX registers, which is rather insufficient for most of the compute-intensive applications.

Slow interaction with integer registers and memory. Data in memory has to be aligned to 64-bit boundaries (misalignment costs three cycles on the Pentium processor family) and arranged in a way that minimizes the number of cache misses. Correct data alignment may significantly expand the data structures (in the worst case, expanded data will not fit into the cache). The delay for a cache miss is at least eight internal clock cycles. Pairing limitations were already mentioned.

Limited number of instructions. MMX has only a limited set of specific operations. Because of the slow interaction between integer and MMX register sets, small programs using intensively both integer and MMX instructions will generally not benefit from MMX.

No flags register. The MMX command set does not change the flags register and therefore the wide variety of branch instructions available on the Pentium is not useable. The only two comparison operators on MMX (`pcmpgt*` and `pcmpeq*`; greater than, equal to) act on signed data and change the corresponding bits of the destination register to 1 (true) or 0 (false). Emulating different — especially unsigned — comparisons takes additional time.

No commands with immediate operands. Immediate operands have to be loaded from memory or generated by other means (e.g., by xoring or comparing a register to itself).

Only 16-bit signed multiplication. Applications intensively using the unsigned multiplication may become significantly slower. IDEA multiplication $\odot$ (Section 3), which is expensive to emulate using unsigned multiplication is even more expensive to emulate using only the signed multiplication (see Section 4). Emulation of $\odot$ using the available MMX instructions needs two multiplications: one to calculate the higher 16 bits of the result (`pmulhw`) and another to calculate the lower 16 bits (`pmullw`).

Standard reference for MMX optimization is [Int97a].

Definition 1. *Let the subscript s (resp. u) under a binary operator denote signedness (resp. unsignedness) of the corresponding operation. Let $*_s$ and $*_u$ be respectively the signed and unsigned multiplication operations from $\mathbb{Z}^2_{2^{16}}$ to $\mathbb{Z}_{2^{32}}$ ($*_u$ is the standard multiplication, expandable to $\mathbb{Z}^2_{2^{32}}$). Let $\mathsf{True}(\phi)$ be $2^{16}-1$ if ϕ is true and 0 otherwise. Next we define several basic operators corresponding one-to-one to the instructions of MMX. Actually the correspondence is* 4-way, *i.e., the MMX instructions execute four such operations in parallel. Let*

$$\begin{aligned}
\mathsf{Cmpeq}(a,b) &:= \mathsf{True}(a=b)\\
\mathsf{Cmpgt}(a,b) &:= \mathsf{True}(a >_s b)\\
a \oplus b &:= a \textit{ bitwise ``xor'' in } \mathbb{Z}_{2^{16}}\\
a \mathbin{\&} b &:= a \textit{ bitwise ``and'' in } \mathbb{Z}_{2^{16}}\\
a \boxminus b &:= a - b \mod 2^{16}\\
a \boxplus b &:= a + b \mod 2^{16}\\
\mathsf{Subus}(a,b) &:= (a \boxminus b) \mathbin{\&} \mathsf{True}(a >_u b)\\
\mathsf{Mull}(a,b) &:= (a *_s b) \mathbin{\&} (2^{16}-1)\\
\mathsf{Mulh}(a,b) &:= \lfloor (a *_s b)/2^{16} \rfloor \ .
\end{aligned}$$

3 Introduction to IDEA

IDEA is — like most of the advanced block ciphers — an iterated cipher. IDEA consists of 8 identical rounds that map the 4-tuple of 16-bit round input, $(X_i^r)_{i=1}^4$, and the 6-tuple of 16-bit round subkeys $(Z_i^r)_{i=1}^6$ (expanded from the 128-bit key using the key expansion algorithm) into the $(X_i^{r+1})_{i=1}^4$. After eight rounds the output transformation will be executed. A round consists of several applications of three group operations, whole IDEA can be presented as a directed labeled graph with labels from the set $\{\oplus, \boxplus, \odot\}$ (Figure 1).

Technically, let $d : \mathbb{Z}_{2^{16}} \to \mathbb{Z}^*_{2^{16}+1}$, $d(x) = 2^{16}$ if $x = 0$ and $d(x) = x$ otherwise. The group operations used in IDEA are $a \oplus b$, $a \boxplus b$ and $a \odot b$, where

$$a \odot b := d^{-1}(d(a) \cdot d(b) \mod 2^{16} + 1).$$

In particular, these operations were chosen for no two of them to be distributive or associative to each other [LMM91]. This fact guarantees that all operations in the IDEA schematics must be executed in an order not contrary to the data dependencies. Operations not dependent on each other's output can be executed in parallel: M_1^r in parallel with M_2^r, A_1^r and A_2^r; E_1^r with E_2^r; E_3^r with E_4^r. On a SIMD architecture where only similar operations can be executed simultaneously, M_1^r and M_2^r cannot be performed in the same instruction as A_1^r and A_2^r.

IDEA satisfies most of the key attributes of multimedia applications used by designing MMX, therefore being an almost ideal candidate cipher to get benefit from MMX:

- IDEA has small integer data types (all the operations work on 16-bit data). Having only small data values enables to pack several of them into one register and thereafter process multiple plaintext blocks in parallel (one of the main factors in effective parallelization).
- IDEA processes the same data over and over without requiring random memory accesses, therefore needing less interaction with the slow memory. Additionally, IDEA lacks operations necessitating expensive, non-parallelizable, table lookups (another main factor in effective parallelization).
- IDEA is based on two 16-bit operations that are common in multimedia applications (16-bit multiplication and addition) and on exclusive or that is a primitive instruction in almost every microprocessor. Although IDEA's multiplication is not trivial to implement on MMX, MMX still provides *some* speedup (compared to the Pentium) per every multiplication (an important factor to get an overall speedup).

4 Fast Implementation

We have addressed all problems mentioned above and completed a fast implementation of IDEA on a Pentium MMX. Some of the tasks we had to solve are outlined below. We assume the plaintext to be in an MMX register and the pointer to the key schedule in an integer register. The ciphertext can be read afterwards from the same MMX register.

General optimization. Optimal use of registers, with minimized number of move instructions. Minimized use of memory: only constants and subkeys are read from memory. Subkeys and constants are correctly aligned to avoid time

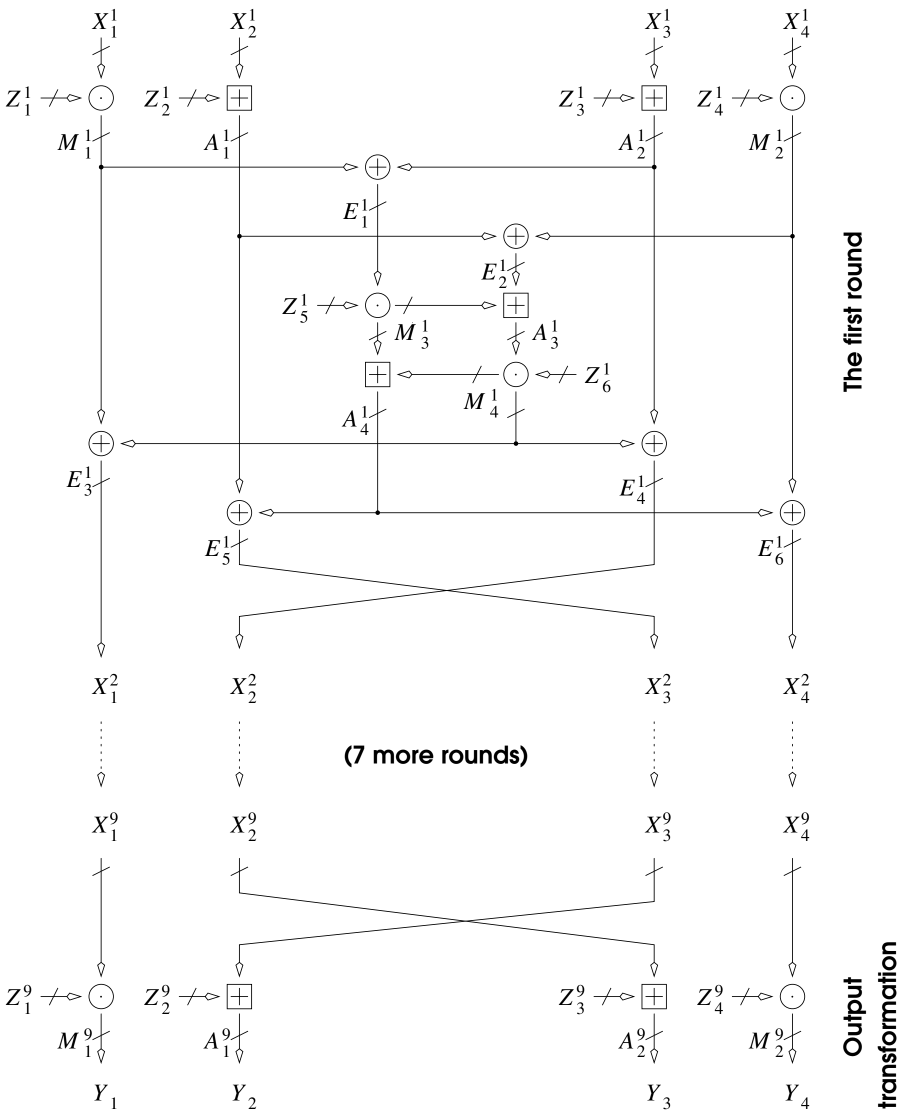

Fig. 1. The schematics of IDEA

penalties due to data misalignment (the key schedule has therefore expanded from 104 to 136 bytes). Data in memory is kept compactly and reused to reduce the number of cache misses. From the integer registers only one is used (as a pointer to the round subkeys). Nothing is written from MMX registers to memory or integer registers.

Effective use of super-scalar parallelism. In our implementation, 693 instructions are paired into 358 cycles. Excellent pairing (0.517 cycles per instruction) is a little miracle (cf to > 0.56 cycles per instruction got by Bosselaers when implementing hash functions for the Pentium, [Bos97]) and is definitely one of the sources of the effectiveness of our implementation.

Use of SIMD parallelism. M_1^r and M_2^r (resp, A_1^r and A_2^r, ...) are calculated in parallel by using the SWAR capability of MMX processors. This is another main factor in increasing the speed of IDEA.

Emulation of $\odot$, using the available MMX instructions, is done, as we believe, optimally. In the following we shortly explain how.

Lemma 1.

$$\begin{aligned} a *_u b =& 2^{16} \cdot (\mathit{Mulh}(a,b) + (a \mathbin{\&} \mathit{Cmpgt}(0,b)) + (b \mathbin{\&} \mathit{Cmpgt}(0,a))) + \\ & \mathit{Mull}(a,b) \enspace . \end{aligned}$$

Proof:

$a, b \geq_s 0$**.** In this case $a *_s b = a *_u b$.
$a \geq_s 0, 0 >_s b$**.** In this case, a is a positive and b is a negative number. Thus, $a *_s b = a *_u (b - 2^{16}) = a *_u b - 2^{16} *_u a$.
$b \geq_s 0, 0 >_s a$**.** Complementary to the previous case.
$0 >_s a, b$**.** In this case, $a *_s b = a *_u b - 2^{16} *_u a - 2^{16} *_u b - 2^{32} \equiv a *_u b - 2^{16} *_u a - 2^{16} *_u b$.

Results got by analyzing the four cases can be generalized by simple means to complete the proof. ∎

As already mentioned, MMX lacks unsigned comparison instructions. Our implementation needs one of them, which will be emulated using existing instructions. Let $\mathsf{Cmpleu}(a,b) = \mathsf{True}(a \leq_u b)$. It is easy to see that

$$\mathsf{Cmpleu}(a,b) = \mathsf{Cmpeq}(\mathsf{Subus}(a,b), 0) \enspace .$$

Lemma 2. *Let a, $b \in \mathbb{Z}_{2^{16}}$. Let $h := (a *_u b)/2^{16}$ and $l := (a *_u b) \mathbin{\&} (2^{16} - 1)$ be calculated by the previous lemma. Then*

$$\begin{aligned} a \odot b =& ((1 \boxminus a \boxminus b) \mathbin{\&} \mathit{Cmpeq}(h,l)) \boxplus \\ & ((1 \boxplus l \boxminus h \boxplus \mathit{Cmpleu}(h,l)) \mathbin{\&} (\mathit{Cmpeq}(h,l) \oplus (2^{16} - 1))) \enspace . \end{aligned}$$

Proof: The claim follows easily from Lemma 2 of [LM90] by noticing that $h = l$ iff $a *_u b = 0$. ■

These formulas give a direct way to break down the $\odot$ operation into basic operations, corresponding one-to-one to MMX instructions. For example, $\mathsf{Cmpeq}(h, l)$ corresponds to the instruction `pcmpeqw`, $\mathsf{Cmpgt}(h, l)$ to `pcmpgtw`, $\mathsf{Subus}(a, b)$ to `psubusw`, $\mathsf{Mull}(a, b)$ to `pmullw`, $\mathsf{Mulh}(a, b)$ to `pmulhw`. The given formula for emulation of 16-bit unsigned multiplication is, as far as we know, faster than any previously published algorithm for MMX and therefore interesting in itself.

Including also the necessary move instructions, the minimal number of MMX instructions needed to emulate $\odot$ by the procedure given above is 26. Additional highly processor (and algorithm) dependent mechanisms enable to get rid of three more instructions per IDEA multiplication, therefore resulting in 69 instructions per round (remember, we do M_1^r and M_2^r in parallel). Everything else (e.g., parallel adding, xoring) is accomplished in 13 instructions, hence a round takes 82 instructions. The output transform takes 29 instructions and the necessary endianness conversion takes 8 instructions, therefore the full IDEA has 693 instructions. After pairing, IDEA encryption takes 358 clock cycles or 29.7 Mbits/s on a 166MHz MMX. Note that our implementations are not subject to timing attacks [Koc96] due to the lack of jump instructions and any variable duration instructions.

Remark 1. Schneier and Whiting [SW97] have conjectured that there exists an IDEA implementation for the Pentium with ≈ 400 cycles, which is unrealistic for the next reason. Every round has four sequential emulations of $\odot$. The critical path of the $\odot$ operation contains integer multiplication (with latency 9) and at least 6 other instructions (moving one of the operands into the accumulator and afterwards converting the result of $*_u$ to the result of $\odot$) that cannot be paired with each other, therefore the multiplications take at least 60 cycles per round. The XOR and addition operators present in the IDEA schematics cannot be paired with emulations of $\odot$ and therefore take additional time. Adding the output transform and endianness conversion, there seems to be no obvious way to significantly better the implementation of Bosselaers.

5 Parallel Execution of Four IDEA-s

In MMX, four 16-bit multiplications are executed simultaneously. The same is true for every other instruction used to emulate $\odot$. During each round, three such 4-way multiplications are done, giving a 64-bit result, only a part of which is really used in the implementation described in Section 4 (Figure 2, left):

First multiplication (M_1^r, M_2^r). The first (bits $0 \ldots 15$) and the fourth (bits $48 \ldots 63$) word of the result are used. (This multiplication is also done during the output transformation.)

Second multiplication (M_3^r). The second word (bits $16 \ldots 31$) of the result is used.

Third multiplication (M_4^r). The third word (bits 32 … 47) of the result is used.

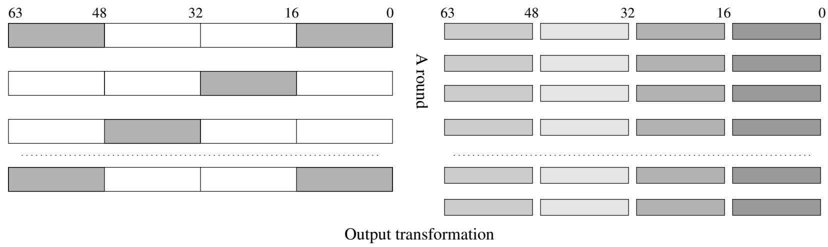

Fig. 2. Using of SWAR data during IDEA (unused fields are left blank). Left: one IDEA in parallel, right: four IDEA's in parallel.

We extended our implementation to encrypt four blocks in parallel, by fully using the results of all four multiplications at every step (Fig. 2, right. Note that such implementation will require unparallelizing the execution of M_1^r and M_2^r). Two 4×4 matrix transpositions (to (un)parallelize four 64-bit blocks), additional endianness conversions and extensive memory access (due to the lack of registers) will "slow" the implementation down to $\approx$ 135 cycles per IDEA encryption. A not-fully optimized implementation encrypts one IDEA block in 135.75 cycles (543 cycles/1056 instructions for 4-way IDEA), Table 2. This scales up to about 212 Mbits/s on a 450 MHz Pentium II, compared to the 300 Mbits/s of the fastest (known) hardware solution by Ascom.

Cipher	IDEA	4-way IDEA
166 MHz MMX, seconds	8.97 – 9.07	3.53 – 3.56
166 MHz MMX, Mbits/s	28.2 – 28.5	71.8 – 72.4
MMX, cycles (with overhead)	372 – 376	147 – 148
MMX, cycles (w/o overhead)	358	135.75
233 MHz Pentium II, seconds	7.78 – 7.96	2.38 – 2.43
233 MHz Pentium II, Mbits/s	32.2 – 32.9	105.1 – 107.2
Pentium II, cycles (with overhead)	453 – 464	139 – 142

Table 2. Test data. The "real life" throughput of IDEA-ECB on the Pentium MMX and on the Pentium II. Seconds - the time to encrypt four million 64-bit blocks.

6 Different Multimedia Extensions

If MMX had the unsigned multiplication instruction, the number of instructions per IDEA multiplication would decrease by 6. If MMX had the unsigned comparison instruction `pcmpgtuw`, the number of instructions per IDEA multiplication would decrease by 2. In the presence of both of these instructions, IDEA encryption on MMX machines could be done much faster than DES (we estimate $\approx 250 - 255$ cycles); 4-way IDEA would be faster than Square [DKR97] or any of the recently proposed AES candidate ciphers (we estimate $\approx 95 - 100$ cycles). Conditional move instructions, present in the Cyrix's — but not in the Intel's — version of MMX, would further speed up IDEA. If even such imperceptible changes fastened up a cipher significantly, what about the multimedia extensions that differ from MMX in major aspects?

Lately, in May 1998, Motorola unveiled their new multimedia architecture called AltiVec [Mot98], claimed to be much more powerful than any of the previously mentioned architectures. In particular, AltiVec has increased parallelism (128-bit vector registers) and a family of instructions to perform up to *eight* 16-bit (un)signed multiplications (with accumulate) in parallel. Additionally, AltiVec has a special inter-element byte permutation instruction and several vector rotation instructions and therefore allows to implement new fast ciphers using data-dependent rotations and byte permutations. One of the goals of AltiVec (unlike the MMX) was to accelerate data encryption algorithms [Mot98, page 1-4]. A short comparison between MMX and AltiVec is given in Table 3.

Architecture	MMX	AltiVec
Company	Intel	Motorola
Year	1997	1999
Endianness	little	both
Max no of operands	2	4
$\sharp$(vector registers)	8 (FP)	32 (separate)
Width of vector registers	64	128
8-bit parallelism	8	16
16-bit parallelism	4	8
32-bit parallelism	2	4
16×16-bit signed multiplication	Yes	Yes
16×16-bit unsigned multiplication	No	Yes
Signed comparison	Yes	Yes
Unsigned comparison	No	Yes
Data-dependent rotation	No	Yes

Table 3. Short comparison of MMX and AltiVec.

Vector processors provide even more parallelism. Krste Asanović has reported an implementation of 32-way IDEA on a 40 MHz T0 [AJ96] reaching 112 Mbits/s. No "industry-standard" block cipher has that level of inner parallelism.

7 Block Cipher Parallelization

Ciphers using S-boxes and/or lookup tables (e.g., DES, alleged RC4, SEAL, Blowfish, Khufu) do not take major advantage from the multimedia extensions of MMX (though they could benefit from the larger cache or word-size) as the MMX registers cannot be used as memory pointers. Parallelization of these ciphers would need accessing several "randomly" chosen memory cells simultaneously. RC5 [Riv95], which does not use S-boxes, does not benefit from MMX either because of the expensive non-parallelizable variable rotation involved.

It is interesting to note that some of the newest block ciphers, including the AES candidates MARS [BCD$^+$98] and RC6 [RRSY98], rely on the 32-bit unsigned multiplication. The reasoning of the authors is that such multiplication is very cheap on nowadays common microprocessors. This claim is indeed true, but MMX technology cannot be used to accelerate these ciphers (and neither can AltiVec) because of the lack of a 32-bit parallel multiplication. There is a certain tradeoff (and even a contradiction) here. MARS and RC6 are optimized for the new 32-bit processors (mainly for the Pentium II), utilizing fully the 32-bit operations provided by such processors. At the same time, these ciphers ignore the multimedia extensions existing in the very same processors.

Further work can be done in trying to optimize different conventional ciphers for the Pentium MMX, but as it was pointed out, most of the commonly known block ciphers do not benefit from MMX. Still, in some cases interleaving Pentium integer and MMX instructions may result in some speedup. In particular, bit-slice MMX implementations of different block ciphers should be more than twice as fast because of the longer wordsize and additional logical operations.

One could think that MMX was designed "especially" to accelerate IDEA, but it would be more correct to say that IDEA is a cipher with key attributes very similar to those of multimedia applications (cf Sect. 3), by a loose definition of multimedia applications as applications benefiting from the Pentium MMX (different vendors have optimized their processors to be optimal for different subsets of multimedia applications).

A family of new block ciphers can be designed to take full advantage of MMX. A straightforward way would be to iteratively execute four copies of the IDEA round function in parallel and then mix their outputs in a suitable way. Would it be sufficient to apply a well chosen 8/16-bit word permutation to the 256-bit output of every round of this 4-way IDEA to get a secure cipher? A way providing more efficient diffusion would be to use Pseudo-Hadamard Transforms [Mas94,SKW$^+$98]. Further research in this area is deferred to a future work. An interested reader may turn to [Cla97], where parallelized versions of the stream cipher Wake were proposed.

A more general task is to study design principles of secure ciphers based on the same basic operations (e.g., massively parallel 16-bit multiplication and addition of sequential data) as the existing multimedia applications. Such ciphers would perform well on nowadays microprocessors, therefore reducing the need for separate encryption and multimedia hardware (it can be compared to the approach of [BP97] that uses the same hardware for RSA and IDEA). Efficient confusion on such ciphers may be achieved by using 16-bit multiplication mixed with other 8-bit and 16-bit operations; diffusion may be achieved by additionally using 32-bit and 64-bit operations (e.g., shifts — but remember the "small data type" paradigm).

Yet another task is to study encryption modes allowing fast parallel encryption and decryption. The ECB mode can be used for both parallel encryption and decryption, but it has limited security in real life situations. The CBC mode can be used for parallel decryption but not for parallel encryption. The resulting throughput of IDEA encryption on a 233 MHz Pentium II would be 32 – 33 Mbits/s for encryption and 105 – 107 Mbits/s for decryption in standard CBC mode (Table 2). Encryption modes allowing both fast parallel encryption and decryption are needed. Note that such encryption modes are not only important for software but also for hardware architectures. The hardware solution mentioned before provides a throughput of 300 Mbits/sec in ECB mode, and a throughput of 100 Mbits/sec in the other modes. An example candidate is the counter mode [MOV96, Sect. 7.2.2] which allows parallel encryption/decryption while providing almost ideal security in the random oracle model [BDJR97] but which is not suited for use with differentially weak ciphers [BK98].

One could see the problem also from the viewpoint of a processor designer and ask what (minimal) extensions should be added to an existing general-purpose processor to achieve significant speedup of industry-standard cryptographic primitives. While the general answer seems to be out of our reach due to the diversity of cryptographic primitives, suggestions can be given to accelerate any fixed primitive (see discussion in the beginning of Sect. 6).

8 Conclusion

We have shown that it is possible to speed up the IDEA block cipher significantly by using the MMX extensions of Intel's Pentium processor. This is remarkable when taking into account the unfriendliness of the instruction set of MMX. Our fast implementation is

- 1.65 times faster than the best known assembler implementation on the Pentium by Antoon Bosselaers,
- $\approx$ 2.55 times faster than the C version on the Pentium in the popular library `SSLeay v0.90b`, when compiled with `egcs 1.0.2` and full optimization.

By parallelizing four IDEA's, the encryption speed is increased by a factor of about 2.64 times, giving a total acceleration of 4.35 times compared to the implementation of Bosselaers. Implications (including the massive parallel key search) of using such parallel versions of conventional ciphers were already described in [Bih97] and were not repeated in this paper.

By noting that most of the nowadays "industry-standard" block ciphers do not benefit from MMX, we raise the problem of designing block ciphers (and encryption modes) fully utilizing the basic operations of multimedia.

9 Acknowledgements

Complete development was done on a `Linux` machine, using `g++` compiler and `gas` assembler. We are thankful to Antoon Bosselaers for proof-reading and for providing the speed numbers in Table 1, and to Mark Tehver and anonymous referees for several valuable remarks.

References

AJ96. Krste Asanović and David Johnson. Torrent Architecture Manual. Technical report, The International Computer Science Institution, Berkley, December 1996. Technical report TR-96-056.

BCD+98. Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr., Luke O'Connor, Mohammad Peyravian, David Safford, and Nevenko Zunic. MARS — A Candidate Cipher for AES. Available at `http://www.research.ibm.com/security/mars.html`, June 1998.

BDJR97. Mihir Bellare, Anand Desai, E. Jokipii, and Phil Rogaway. A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation. In *Proceedings of 38th Annual Symposium on Foundations of Computer Science*, 1997.

BGV96. Antoon Bosselaers, René Govaerts, and Joos Vandewalle. Fast Hashing on the Pentium. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 298–312. Springer-Verlag, 1996.

Bih97. Eli Biham. A Fast New DES Implementation in Software. In Eli Biham, editor, *Fast Software Encryption '97*, volume 1267 of *Lecture Notes in Computer Science*, pages 261–272. Springer-Verlag, 1997.

BK98. Alex Biryukov and Eyal Kushilevitz. From Differential Cryptanalysis to Ciphertext-Only Attacks. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 72–88. Springer-Verlag, 1998.

BKR97. Johan Borst, Lars R. Knudsen, and Vincent Rijmen. Two Attacks on Reduced IDEA. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 1–13. Springer-Verlag, 1997.

Bos97. Antoon Bosselaers. Even faster hashing on the Pentium. Presented at the rump session of Eurocrypt'97, 1997.

BP97. Ahto Buldas and Jüri Poldre. A VLSI implementation of RSA and IDEA encryption engine. In *NORCHIP '97*, 1997.

Cla97. Craig S. K. Clapp. Optimizing a Fast Stream Cipher for VLIW, SIMD, and Superscalar Processors. In Eli Biham, editor, *Fast Software Encryption '97*, volume 1267 of *Lecture Notes in Computer Science*, pages 273–287. Springer-Verlag, 1997.

DC98. Joan Daemen and Craig S. K. Clapp. Fast Hashing and Stream Encryption with Panama. In Serge Vaudenay, editor, *Fast Software Encryption '98*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74. Springer-Verlag, 1998.

DGV94. Joan Daemen, René Govaerts, and Joos Vandewalle. Weak Keys for IDEA. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 224–231. Springer-Verlag, 1994.

Die97. Hans Dietz. Technical Summary: SWAR Technology. Technical report, School of Electrical and Computer Engineering, Purdue University, February 1997. Available at `http://dynamo.ecn.purdue.edu/~hankd/SWAR/over.html`.

DKR97. Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *Fast Software Encryption*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer-Verlag, 1997.

Haw98. Philip Hawkes. Differential-Linear Weak Key Classes of IDEA. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 112–126. Springer-Verlag, 1998.

HK97. Shai Halevi and Hugo Krawczyk. MMH: Software Message Authentication in the Gbit/Second Rates. In Eli Biham, editor, *Fast Software Encryption '97*, volume 1267 of *Lecture Notes in Computer Science*, pages 172–189. Springer-Verlag, 1997.

Int97a. Intel. *Intel Architecture Optimization Manual*, 1997. Order Number 242816-003.

Int97b. Intel. *Intel Architecture Software Developer's Manual. Volume 1: Basic architecture*, 1997. Order Number 243190.

Koc96. Paul Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '88*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, 1996.

Kum97. Masayasu Kumagai. Implementation of IDEA on MMX. Available at `http://www2s.biglobe.ne.jp/~kumagai/idea_mmx.zip`, April 1997.

Kwa98. Michael Kwan. Bitslice DES. Unpublished. Information available from `http://www.cs.mu.oz.au/ mkwan/bitslice/Welcome.html`, May 1998.

LM90. Xuejia Lai and James Massey. A proposal for a new block encryption standard. In Ivan Bjerre Damgård, editor, *Advances in Cryptology — EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer-Verlag, 1990.

LMM91. Xuejia Lai, James L. Massey, and Sean Murphy. Markov Ciphers and Differential Cryptanalysis. In D. W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer-Verlag, 1991.

Mas94. J. Massey. SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm. In Ross Anderson, editor, *Fast Software Encryption*, volume 809 of *Lecture Notes in Computer Science*, pages 1–17. Springer Verlag, 1994.

Mot98. Motorola. *AltiVec Technology Programming Environments Manual*, May 1998. A preliminary revision 0.2.

MOV96. Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

PRB98. Bart Preneel, Vincent Rijmen, and Antoon Bosselaers. Recent Developments in the Design of Conventional Algorithms. In B. Preneel, R. Govaerts, and J. Vandewalle, editors, *Computer Security and Industrial Cryptography, State of the Art and Evolution*, volume 1528 of *Lecture Notes in Computer Science*, pages 90–115. Springer-Verlag, 1998.

Riv95. Ronald L. Rivest. The RC5 Encryption Algorithm. In Bart Preneel, editor, *Fast Software Encryption*, volume 1008 of *Lecture Notes in Computer Science*, pages 86–96. Springer-Verlag, 1995.

RRSY98. Ronald L. Rivest, Matt J. B. Robshaw, R. Sidney, and Y. L. Yin. The RC6 Block Cipher. Available at `http://theory.lcs.mit.edu/~rivest/rc6.ps`, June 1998.

SAM97. Takeshi Shimoyama, Seiichi Amada, and Shiho Moriai. Improved Fast Software Implementation of Block Ciphers. In *International Conference on Information and Communications Security '97*, volume 1334 of *Lecture Notes in Computer Science*, pages 269–273, September 1997.

SKW+98. Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-Bit Block Cipher. In *Selected Areas in Cryptography '98*, June 1998. Lecture Notes in Computer Science (these proceedings).

SW97. Bruce Schneier and Doug Whiting. Fast Software Encryption: Designing Encryption Algorithms for Optimal Software Speed on the Intel Pentium Processor. In Eli Biham, editor, *Fast Software Encryption '97*, volume 1267 of *Lecture Notes in Computer Science*, pages 242–259. Springer-Verlag, 1997.