White Paper

Breakthrough AES Performance with Intel® AES New Instructions

Kahraman Akdemir, Martin Dixon, Wajdi Feghali, Patrick Fay, Vinodh Gopal, Jim Guilford, Erdinc Ozturk, Gil Wolrich, Ronen Zohar

Abstract

Intel continues to provide leadership in developing instruction-set extensions with the recently released ISA support for the Advanced Encryption Standard (AES). This paper presents the excellent performance of the AES algorithm on the Intel® Core™ i7 Processor Extreme Edition, i7-980X, using the AES New Instructions (AES-NI). Performance results for serial and parallel modes of operation are provided for all key sizes, for variable numbers of cores and threads. These results have been achieved using highly optimized implementations of the AES functions that can achieve ~1.3 cycles/byte on a single-core Intel® Core™ i7 Processor Extreme Edition, i7-980X for AES-128 in parallel modes. The paper also has a brief description of how to code to achieve these results and a reference to the complete source code.

Introduction

AES is one of the most popular block ciphers used in cryptography and is specified in the FIPS Standard [1]. It works on 128-bit blocks with a choice of three key sizes of 128, 192 and 256 bits. Intel introduced 6 new instructions in the Intel® Core™ i7 Processor Extreme Edition, i7-980X processor to accelerate the execution of the AES algorithm - AESENC, AESENCLAST, AESDEC, AESDECLAST, AESIMC, and AESKEYGENASSIST.

Instruction	Description
AESENC xmm1, xmm2/m128	Perform one round of an AES encryption flow, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.
AESENCLAST xmm1, xmm2/m128	Perform the last round of an AES encryption flow, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.
AESDEC xmm1, xmm2/m128	Perform one round of an AES decryption flow, using the Equivalent Inverse Cipher, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.
AESDECLAST xmm1, xmm2/m128	Perform the last round of an AES decryption flow, using the Equivalent Inverse Cipher, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.
AESIMC xmm1, xmm2/m128	Perform the InvMixColumn transformation on a 128-bit round key from xmm2/m128 and store the result in xmm1
AESKEYGENASSIST xmm1, xmm2/m128, imm8	Assist in AES round key generation using an 8-bit Round Constant (RCON) specified in the immediate byte, operating on 128 bits of data specified in xmm2/m128 and stores the result in xmm1.

Table 1: AES-NI (New Instructions) Summary

Detailed information about these instructions can be found in the AVX instruction-set reference [2] and the white-paper by S Gueron [4] of Intel. Gueron's white-paper also mentions that the new instructions provide important security benefits over software-based AES implementations.

This paper describes the unprecedented performance of the AES algorithm on the Intel® Core™ i7 Processor Extreme Edition, i7-980X processor. We show the performance of a serial and a parallel mode of operation of the cipher, measured on varying numbers of cores and threads. The Intel® Core™ i7 Processor Extreme Edition, i7-980X processor that was used for this study ran at 3.33 GHz frequency and had 6 cores with hyper-threading enabled (effectively giving a maximum of 12 threads). We measured the results without turbo-mode. Turbo mode is disabled just to simplify the performance analysis. Enabling turbo mode will result in the same or better performance. To achieve these results, we developed highly optimized implementations of AES encrypt/decrypt functions for the various key-sizes and modes. The library of source code can be found in [3].

This paper is organized as follows. We start with a brief description of the performance testing methodology. In the next section, AES encryption/decryption algorithms that work in CBC mode are described. We then provide an overall performance summary and discussion. The last section discusses possible improvements that can be achieved on the AES key scheduler.

Methodology

In this section, we briefly explain the performance measurement methodology. We created a test configuration structure that builds tests using the provided configuration options, runs them on the Intel® Core™ i7 Processor Extreme Edition, i7-980X processor, and reports the timing. A specific test has various configuration parameters such as the number of threads, buffer size of the input, data alignment in the memory, name of the key generation procedure, and the name of the main AES algorithm.

Depending on the number of required threads, the AES algorithm can be run on up to 12 threads on 6 cores, as each core is capable of running 2 threadswith Intel® Hyper-Threading Technology (Intel® HT Technology) enabled. The performance testing structure controls the multiple threads. When a test is called, it is first built using the provided options and the required algorithm is run 1000 times to warm up the cache. The timing is measured using the rdtsc() function which returns the processor time stamp counter (TSC). The TSC is the number of clock cycles since the last reset. The 'TSC_initial' is the TSC recorded before the specific AES algorithm is called . Then, the function is called for the specified number of times. After the runs are complete, the rdtsc() is called again to record the new cycle count 'TSC_final'. The effective cycle count for the called routine is computed using

# of cycles = (TSC_final-TSC_initial)/(number of iterations).

AES Modes of Operation

There are many block cipher modes such as the cipher-block-chaining (CBC) mode. Performance of the modes varies primarily due to the inherent serial or parallel nature of the processing. CBC-Encrypt mode has a serial processing flow and exhibits the worst-case performance behavior. CBC-Decrypt, however, can be explicitly parallelized with efficient software coding for best performance. We show the performance of these two modes for all key-sizes:

- <u>AES CBC Mode Encryption (Serial):</u> The result (cipher text) of a block encryption is used as an input to the encryption of the following block. It is described by the equation: **Cipher[n] = Encrypt$_K$(Cipher[n-1] ⊕ Plaintext[n])**
- <u>AES CBC Mode Decryption (Parallel):</u> This mode can be parallelized due to property in the equation: **Plaintext[n] = Decrypt$_K$(Cipher[n]) ⊕ Cipher[n-1].** The plaintext for many blocks can be processed in parallel since they depend only on ciphertext blocks which are all available. We implemented this mode decrypting 4 blocks in parallel. The code essentially performs round *i* for 4 consecutive blocks followed by round *i+1* for these 4 blocks until the last round. After the final round, the next 4 blocks are processed in an iterative loop. Note that for buffer-sizes that are not multiples of 4 blocks, the remainders are handled one-by-one at the beginning. We could achieve approximately the same performance by processing 3 blocks in parallel, but for efficiency of implementation (specifically, calculating number of remainder blocks), we chose 4 blocks.

CBC Encrypt performance can be improved on a single-thread in some applications that permit processing multiple independent buffers concurrently. For instance, if we process 3 (or more) independent buffers concurrently, the latency of the instructions can be hidden perfectly achieving approximately the same performance of CBC Decrypt. Such optimizations are however, out of the scope of the current paper and will not be considered in the performance discussions.

Key Generation

In our implementations of AES Encrypt/Decrypt, for each of these modes, key scheduling is done at the beginning (but within the timing loop) before the Encrypt/Decrypt routines are called for a given data buffer. For the encryption operation, an optimized key scheduling algorithm is used (with the AESKEYGENASSIST instruction) to generate the round keys. The decryption round keys are computed in two steps. First, the encryption round keys are generated using the encryption key scheduler routine. Next, the AESIMC instruction is utilized in order to compute the decryption round keys.

The AES Encrypt/Decrypt routines take the expanded round keys as their input and implement the encryption/decryption round operations (using the AESENC, AESENCLAST, AESDEC, AESDECLAST instructions) thereafter on the given data buffer. For a given buffer, we measure the **total time** for the 2 steps as described by the following pseudo-code:

```
for (number of iterations){
        round_keys[] = expand_key_schedule(user Key);
        output = encrypt/decrypt(input, buffer_length, round_keys[]);
```

Performance Results

The performance of AES in CBC mode has been measured on the Intel® Core™ i7 Processor Extreme Edition, i7-980X using our highly optimized implementations. The trending values for large buffers (32Kbyte) in terms of cycles per byte have been shown in Table 2.

Parallel CBC decrypt is ~3x faster than serial CBC encrypt

Cycles/Byte	CBC Encrypt			CBC Decrypt		
	128	192	256	128	192	256
1 Core 1 Thread	4.20	4.95	5.70	1.30	1.56	1.80
2 Cores 2 Threads	2.11	2.48	2.86	0.67	0.80	0.91
4 Cores 4 Threads	1.06	1.25	1.44	0.35	0.41	0.47
6 Cores 6 Threads	0.72	0.84	0.97	0.25	0.29	0.33
6 Cores 12 Threads	0.36	0.43	0.49	0.24	0.28	0.32

Hyper-threading provides ~2x speedup on CBC encrypt

Table 2: Performance Summary in Cycles/Byte

Note that on any row (except for the '6 cores and 12 threads' case which will be explained shortly), for a specific key size, there is a ratio of ~3 between CBC encryption and decryption. In other words, CBC decryption performs 3X faster than CBC encryption (e.g. 1.3 cycles/byte compared to 4.2 for AES128 on 1 core). For the serial (encrypt) case, we are limited by the latency of the AES round instructions in the AES pipeline, which is 6 cycles. However, for the parallel (decrypt) case, the algorithm is only limited by the throughput of these instructions, which is 2 cycles.

In order to show the effect of hyper-threading on the performance, we compare the '6 cores and 6 threads' case with the '6 cores and 12 threads' case. All the cases are single threaded except the '6 cores and 12 threads' case: in the '6 cores and 12 threads' case, each core has two threads on it.

An important observation is the ~2Xspeed scaling of the 12 thread case in comparison to the 6 thread case for the CBC encryption. CBC encryption is a serial mode which does not fully utilize the AES pipeline. To be more specific, it only uses 1/3rd of the pipeline. Consequently, when we have two threads running on a single core, pipeline utilization increases to 2/3rd and the number of blocks that are processed will be twice as many as the single-thread. Hyper-threading in CBC decryption provides little performance gain because CBC decrypt already utilizes the AES pipeline fully due to its parallel nature.

Detailed graphs that show performance in these modes for smaller buffers can be found in Appendix A. In particular, note the excellent performance than can be achieved for buffers as small as 64-bytes: on a single core with 1 thread, AES128 CBC-Encrypt can be performed at 5.31 cycles/byte and AES128 CBC-Decrypt can be performed at 4.00 cycles/byte. The performance in Appendix A includes the key expansion. See Appendix B for details of the system on which the results were obtained

The trending values for large buffers (32Kbyte) in terms of cycles per block (16 byte-blocks) have been shown in Table 3. This represents the same data in Table 2 multiplied by 16 and is merely shown for convenience.

Cycles/Block	CBC ENC			CBC DEC		
	128	192	256	128	192	256
1 Core 1 Thread	67.21	79.22	91.23	20.81	25.00	28.86
2 Cores 2 Threads	33.73	39.74	45.75	10.65	12.74	14.63
4 Cores 4 Threads	17.01	20.01	23.03	5.56	6.59	7.58
6 Cores 6 Threads	11.46	13.45	15.47	3.92	4.61	5.28
6 Cores 12 Threads	5.83	6.83	7.86	3.79	4.47	5.15

Table 3: Performance Summary in Cycles/Block

Implementing more efficient key scheduling

As discussed earlier the key scheduling for the AES algorithms implemented in this paper are handled at the beginning before the encryption/decryption starts. However, this might affect the performance for small buffers. In this section, we briefly describe an improved method.

We propose generating the round keys on the fly for best performance. In other words, merging the initial round key generation process with encryption/decryption could minimize the performance overhead caused by the initial key generation latency. In case of CBC encryption, due to its serial nature, one block of data is encrypted (Enc1) while the round keys are generated (KG). Next, one-by-one block encryption continues. This idea is illustrated in the following figure.

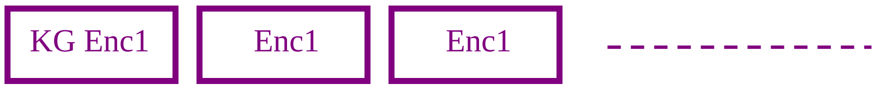

Figure 1: Efficient key generation for CBC encryption

CBC decryption case is also handled in a similar fashion. Since our implementation of this algorithm works in a four block parallel fashion, we propose handling the remainder blocks (modulo 4) at the beginning in parallel with the key generation (KG). More specifically, for a buffer size of N blocks, depending on the value of the remainder (rem = N % 4) where % is the modulus operator, we do one of the following:

- [KG Dec3] when rem = 3.
- [KG Dec2] when rem = 2.
- [KG Dec1] when rem = 1.
- [KG Dec4] when rem = 0.

Then, four-by-four decryption (Dec4) continues (N-rem)/4 times until the end of the buffer is reached. The notation Dec*i* represents a method of processing *i* blocks in parallel. Note that the approach we present here is a generic one which is applicable to all buffer sizes.

The idea is summarized in the following figure.

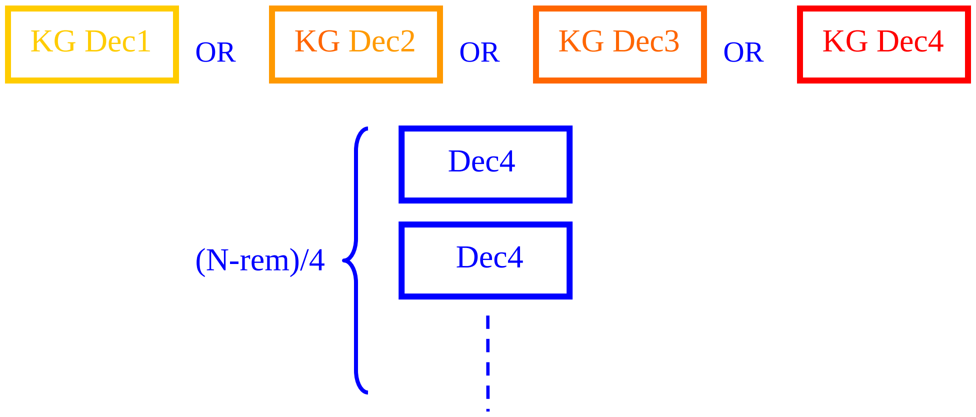

Figure 2: Efficient key generation for CBC decryption

The following example code segments show how this idea could be implemented for CBC encryption and decryption. For the encryption case, the shown example is for [KG Enc1], and for the decryption case, the shown example is for [KG Dec4]. The enc_key_expansion_128 routine in both of these code segments takes the previous round's encryption key in xmm4 register as its input and returns the current round encryption key in the same register. For decryption key expansion, we first compute the encryption round keys, then convert them into decryption round keys using the AESIMC instruction. The computed round keys are then used to encrypt/decrypt the data blocks.

```
enc_key_expansion_128                  ; Generating enc. round key 1
aesenc xmm0, xmm4                      ; 1. block round 1 encryption
```

```
enc_key_expansion_128                  ; Generating enc. round key 1
aesimc xmm5, xmm4                      ; Generating dec. round key 1
aesdec xmm0, xmm5                      ; 1. block round 1 decryption
aesdec xmm1, xmm5                      ; 2. block round 1 decryption
aesdec xmm2, xmm5                      ; 3. block round 1 decryption
aesdec xmm3, xmm5                      ; 4. block round 1 decryption
```

Figure 3: Encryption (top) and decryption (bottom) code examples

Conclusion

We are able to achieve excellent AES performance on the Intel® Core™ i7 Processor Extreme Edition, i7-980X using the new instructions. With optimized code, it is possible to achieve ~0.24 cycles/byte on 6 cores for AES128 on parallel modes for large buffers.

When we run a single thread per core, the serial modes such as CBC Encrypt are ~3X slower than the parallel modes. Whereas the serial modes are slower (~0.72 cycles/byte on 6 cores and 6 threads), hyper-threading gives ~2X performance gain on these modes (~0.36 cycles/byte on 6 cores, 12 threads) compared to running a single-thread per core.

It is possible to improve the key-scheduling by interleaving key-scheduling with the encrypt/decrypt code for better performance on small buffers.

References

[1] *FIPS PUB 197, Advanced Encryption Standard (AES)*, Nat'l Inst. of Standards and Technology, Nov. 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] http://softwarecommunity.intel.com/isn/downloads/intelavx/Intel-AVX-Programming-Reference-31943302.pdf

[3] http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni/

[4] Advanced Encryption Standard (AES) Instructions Set Rev 3, Shay Gueron http://software.intel.com/en-us/articles/advanced-encryption-standard-aes-instructions-set/

System Details

Appendix A – Performance Details

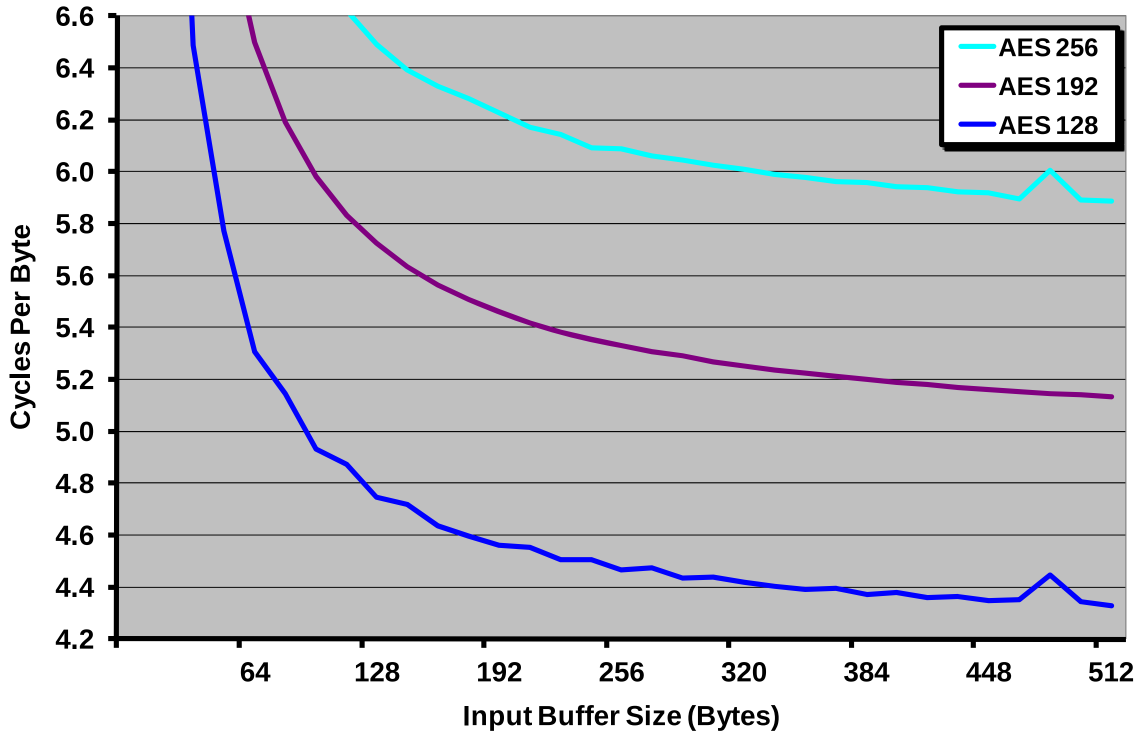

Figure 4: CBC ENC Performance Summary for buffer sizes smaller than 512 bytes

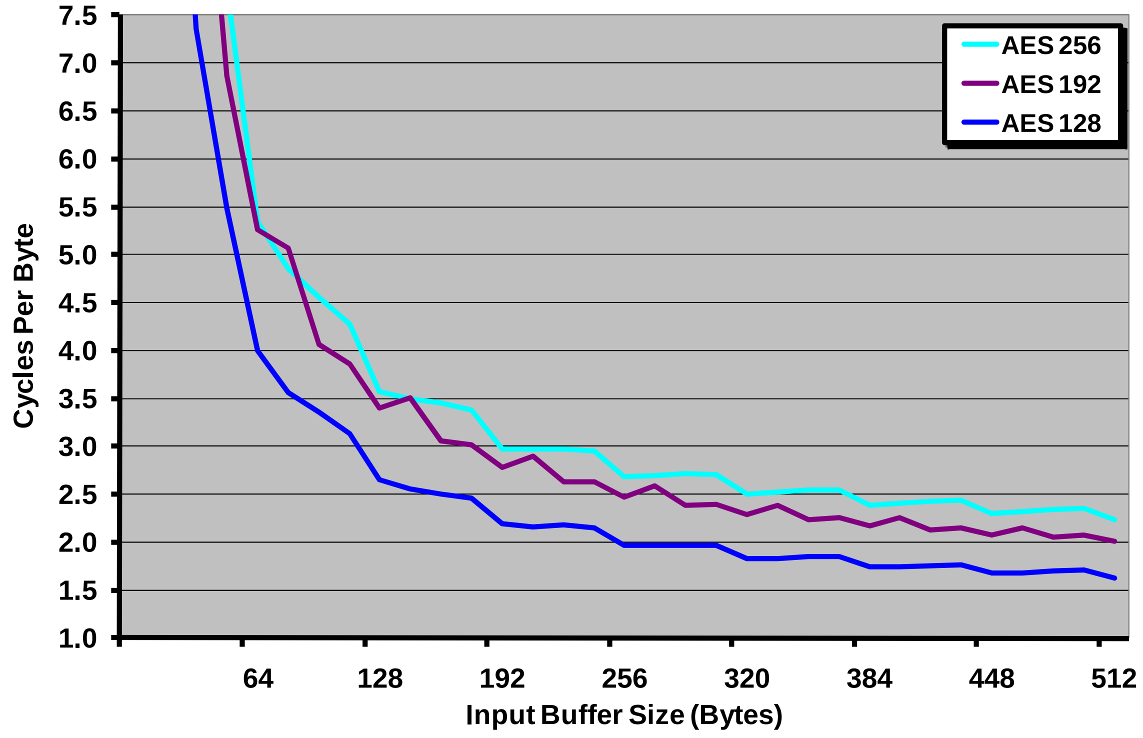

Figure 5: CBC DEC Performance Summary for buffer sizes smaller than 512 bytes

Appendix B – System Configuration

CPU: Intel(R) Core(TM) i7 CPU X 980 @ 3.33GHz (Engineering Sample, 6 cores, 12 logical threads. Turbo disabled. When we show the multi-threaded, not hyper-threaded performance, we are careful to ensure that each software thread runs on a different core.
Chipset: Intel X58.
Memory: 6 GBs DDR3 at 667 MHz. Note that timings in this paper are expected to be independent of memory speed since we are running in cache.
OS: Microsoft Windows* 7 (6.1) Ultimate Edition (Build 7600), 64bit
YASM version: yasm-0.8.0
Timing source code contained in sample source code library [3] in intel_aes_lib\src\aessampletiming.cpp.
Compiler for aessampletiming.cpp: VS2008 SP1 (x64). Options: /O2 /Zi

Notices