Bitsliced High-Performance AES-ECB on GPUs

Rone Kwei Lim, Linda Ruth Petzold, and Çetin Kaya Koç(✉)

Department of Computer Science,
University of California, Santa Barbara, CA 93106, USA
{rklim13793,petzold,koc}@cs.ucsb.edu

Abstract. In order to perform high-performance Monte Carlo simulations of fracture in certain composite materials, we needed fast methods for generating deterministic random numbers. We made several design choices, and due to the fact that the entire simulation was to be done on both CPUs and GPUs, we designed new methods for fast implementation of the AES in the ECB mode on such architectures. This paper describes our algorithms and summarizes the performance results. In our implementation we were able to produce a speed of 78.6 Gbits per second on the GeForce GTX 480, which was 31–62 % faster than the fastest implementations reported in the recent literature on similar devices.

1 Introduction

The purpose of this study was to develop fast methods for generating deterministic random numbers using the AES in the ECB mode. The resulting random numbers were intended to be used in high-performance Monte Carlo simulation of fracture in certain composite materials [10]. The simulations for this study were done both on CPUs and GPUs to obtain the fastest implementations, and thus, to compare the speedup gain. We were motivated to develop high-speed implementations of the 128-bit AES-ECB on the NVIDIA GTX 480 GPU, and subsequently obtained significantly faster implementations of the AES. The present paper reports our implementations along with comparisons to recent results found in the literature.

2 CPU Versus GPU Architectures

A general-purpose CPU generally has several cores to run multiple threads, and a large cache for immediate access to the data, and also, sophisticated flow control mechanisms such as branch prediction, data and instruction prefetching, and out-of-order execution. The availability of floating-point ALUs make such CPUs very suitable for scientific computing tasks, achieving double-precision floating-point arithmetic at the rates of 40-160 GFlop/second at their peak performance. In the context of the research on Monte Carlo simulations of fractures [10], we worked with Intel Core 2 Quad Q6600 CPU at 2.4 GHz and Intel Core i7 2600 CPU at 3.4 GHz. The latter CPU has 4 physical cores and 8 MB cache.

P.Y.A. Ryan et al. (Eds.): Kahn Festschrift, LNCS 9100, pp. 125–133, 2016.
DOI: 10.1007/978-3-662-49301-4_8

In contrast, a GPU, such as NVIDIA GTX 580, has a large number of execution units to process data in parallel. The original intention for designing GPUs was to create hardware that performs 3D graphics processing, however, the GPU architectures have evolved, coupled with a sophisticated computational model and a platform of computation called CUDA (Compute Unified Device Architecture). This new platform offered a C-like programming language, while the hardware provided integer, logical and floating-point instructions to support a wide range of computational needs in scientific computing. The present implementation was done on the NVIDIA GTX 580 which has 16 SMs (Streaming Multiprocessors) where each SM has 32 SPs (Shader Processors). Each SM executes independent streams of instructions while the SPs within each SM execute instruction in an SIMD fashion. The NVIDIA GTX 580 has 64K L1 cache and 768K L2 cache.

There are no sophisticated control flow mechanisms similar to CPUs, however, GPUs run large numbers of threads, providing large parallelism. If a program can be broken up into many threads all doing the same computation on different data (ideally, executing arithmetic operations), a GPU will probably be an order of magnitude faster than a CPU. On the other hand, applications with complex control flow, a CPU is going to be faster many orders of magnitude.

Figure 1, reprinted from [10], makes a comparison of the "silicon budget" (silicon area or number of transistors) for a CPU versus a GPU. The CPU uses most of its transistors for the control logic, the ALUs and the cache. On the other had GPUs spend nearly all of its available silicon area for its simple processors (ALUs).

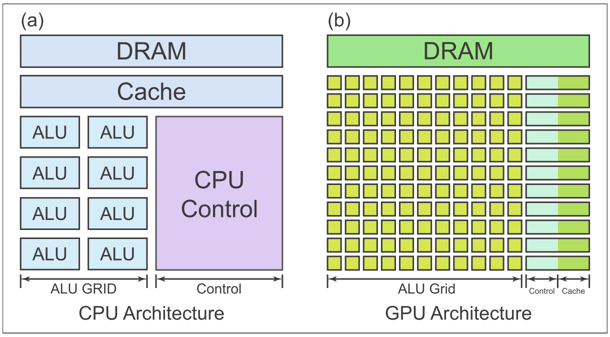


Fig. 1. The silicon area for a CPU versus a GPU [17].

2.1 GTX 480

We have implemented our algorithms on the NVIDIA GTX 480 GPU, which is based on the Fermi architecture. It has 15 SMs (Streaming Multiprocessors), where each SM has 32 SPs (Shader processors). Each SM can execute independent streams of instructions, whereas the SPs within each SM execute instructions in a SIMD (Single Instruction Multiple Data) manner. The NVIDIA GTX

480 has a 64K L1 cache per SM and a 768K L2 cache shared over all SMs. It also has 32768 registers per SM and 1.5 GB of global GPU memory. GPUs lack the sophisticated flow control mechanisms that are present on CPUs, such as branch predictor. Instead, GPUs have more transistors devoted to execution units and are designed to run large numbers of threads, which makes them suited for problems with a high degree of parallelism [22, 24]. Figure 2 shows a schematic illustration of the Fermi architecture.

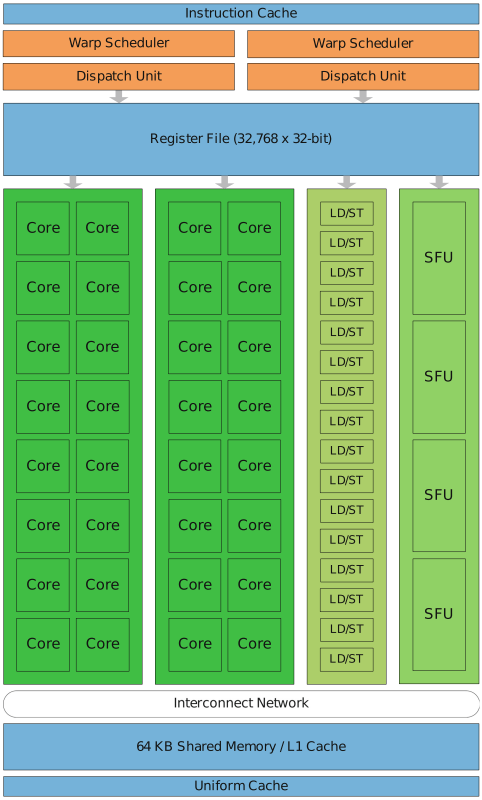


Fig. 2. Fermi architecture diagram [22].

2.2 Comparing GPUs

The GTX 285 has 30 SMs, each with 8 SPs. It has 16K L1 cache per SM and no L2 cache. It also has 16384 registers per SM and 1 GB of global GPU memory. In comparison, the 8800 GTX has 16 SMs, each with 8 SPs. It has 16K L1 cache per SM and no L2 cache. It also has 8192 registers per SM and 768 MB of global GPU memory.

We find it useful to make a comparison of various GPUs that we are referencing in the context of our AES implementations. Table 1 compares various

Table 1. Comparison of various GPUs.

	8800 GTX [18]	GTX 285 [19]	Tesla C2050 [21]	GTX 480 [20]
Bus bandwidth	4 GB/s	8 GB/s	8 GB/s	8 GB/s
Memory size	768 MB	1024 MB	3072 MB	1536 MB
Mem bandwidth	86.4 GB/s	159.0 GB/s	144 GB/s	177.4 GB/s
SP count	128	240	448	480
SP clock	1350 MHz	1476 MHz	1150 MHz	1400 MHz
CC	1.0	1.3	2.0	2.0

GPUs referenced in this paper. Here, CC refers to “Compute Capability”, which is an index assigned by NVIDIA to the CUDA devices to indicate its set of computation-related features. Higher CC indicates newer architectures, and the NVIDIA’s newest devices have a CC up to 3.5 [16].

3 AES Encryption on CPU and GPUs

Since the standardization of the Rijndael algorithm as the Advanced Encryption Standard by NIST [14], many implementations have been reported in the literature, most of which rely on known techniques. The creators of the Rijndael algorithm describe two fundamental techniques for 8-bit and 32-bit CPUs [4]. The most common use of the AES is for the 128-bit (16-byte) key; it is projected that AES will be 40 % slower [1] for 32-byte keys since it uses 14 rounds, instead of 10.

Furthermore, there are several modes of operation: the CBC (cipher-block chaining), the ECB (electronic code-book), the OFB (output feedback), and the CTR (counter) modes, etc. Moreover, there are several ways of benchmarking the AES software, making a fair comparison very difficult. Most common comparisons involve AES-ECB and AES-CTR modes. We refer the reader to a highly useful paper by Bernstein and Schwabe [1] that gives extensive analyses of various implementations, along with the most impressive benchmark results.

Earlier GPU implementations [3,5,28] used graphics pipeline and OpenGL to compute the AES round function, since CUDA was not available back then. The availability of CUDA made sophisticated high-speed implementations possible.

Another point of discussion that is relevant to the present paper is bitsliced AES implementations on various CPUs. There are several papers of interest: Rebeiro et al. [27], Matsui [12], and Matsui and Nakajima [13]. Bitsliced implementations are not as competitive with word-level implementations on CPUs due to the cost of transpositions of the ciphertext.

4 AES-ECB on the GPUs

Our implementation starts with the CPU-based bitsliced implementation of the AES by Kasper and Schwabe [8]. Their implementation processes 8 16-byte blocks at a time. A direct conversion to a GPU implementation results in poor performance, due to an insufficient number of registers. The 8 blocks alone take up 32 registers per thread, and each thread is limited to 63 registers maximum. The result is that the compiler spills variables into memory instead of keeping them in registers.

We restructured the algorithm to process 4 16-byte blocks at a time to improve performance. The sections below describe the performance improvements we made to various parts of the AES algorithm.

4.1 Bit Ordering

In our bitslicing implementation, bits from multiple blocks are collected together, i.e., bit 0 of row 0, column 0 from blocks 0, 1, 2, 3 are grouped together, as shown in Figs. 3 and 4. Each bitsliced state variable has 64 bits; there are 8 of these state variables.

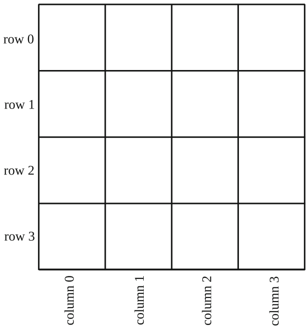


Fig. 3. The state of one block.

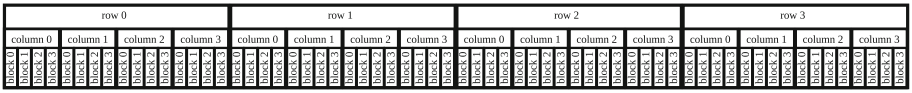


Fig. 4. The bitsliced state.

4.2 Load and Store

On GPUs, the performance of global memory is improved when it is accessed contiguously. When reading the input blocks, we first load the blocks contiguously from global memory to shared memory, and then distribute them among individual threads. Similarly, when writing the output blocks, we first write the blocks to shared memory from individual threads, and then collect them together and store to global memory contiguously.

4.3 SubBytes

The AES algorithm defined in [14] used a table lookup for the S-box. In the bitsliced implementation, the table lookup is replaced by a series of Boolean operations (`xor`, `or`, `and`) [8]. Kasper and Schwabe [8] used 163 CPU SSE instructions. In our implementation, since we restructured the algorithm to process 4 blocks at a time, extra registers are available that we use to store intermediate values, thus reducing the instruction count to 117 × 2. The doubling of the instruction count arises from the fact that the GPU registers are 32 bits, thus, each 64-bit bitsliced state requires 2 operations to process. Since the two halves can be processed independently, we utilize ILP (instruction level parallelism) to increase performance.

4.4 ShiftRows

In this step, the bytes in a block are shifted by a variable amount for each row, as shown in Fig. 5. In the bitsliced state, this operation becomes a rearrangement of nibbles (4-bits), as shown in Fig. 6. The CPU version used the `pshufb` instruction [8], but this instruction is not available on the GTX 480. Instead, we found the GTX 480 has a `prmt` instruction that rearranges bytes [23]. We combined this instruction with the standard C bit operations (`>>`, `<<`, `&`, `|`, `^`) to improve performance. The CPU version uses 8 SSE instructions [8], while our GPU version uses 32 `prmt`, 16 shift, and 16 bitwise `and` instructions. The GPU version requires more instructions since it involves handling nibbles (4 bits) instead of whole bytes (8 bits).

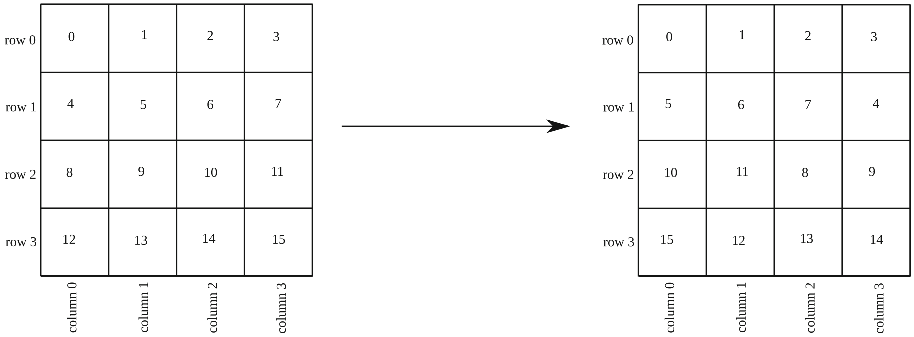


Fig. 5. The ShiftRows step.

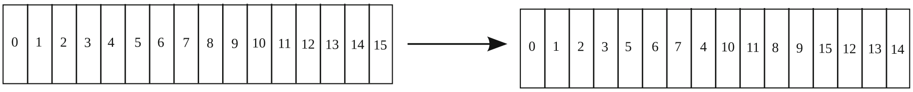
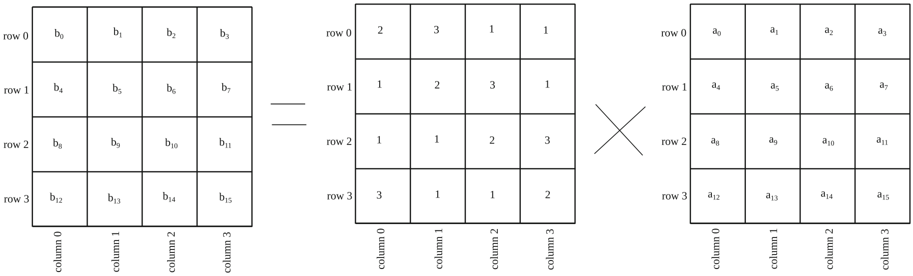


Fig. 6. The ShiftRows step for the bitsliced state.

4.5 MixColumns

This step involves a matrix multiplication over the AES finite field, as specified in [14] (see Fig. 7). Using Boolean operations, the matrix multiplication becomes a sequence of shifts and xor operations. The CPU version of Kasper and Schwabe uses 16 `pshufd` and 27 `xor` instructions [8], while our GPU version uses 27×2 `xor` and 8×2 `prmt` instructions. The 2 factor is explained in the SubBytes section.



$$\begin{array}{c|cccc} \text{row 0} & b_0 & b_1 & b_2 & b_3 \\ \text{row 1} & b_4 & b_5 & b_6 & b_7 \\ \text{row 2} & b_8 & b_9 & b_{10} & b_{11} \\ \text{row 3} & b_{12} & b_{13} & b_{14} & b_{15} \\ \hline & \text{column 0} & \text{column 1} & \text{column 2} & \text{column 3} \end{array} = \begin{array}{c|cccc} \text{row 0} & 2 & 3 & 1 & 1 \\ \text{row 1} & 1 & 2 & 3 & 1 \\ \text{row 2} & 1 & 1 & 2 & 3 \\ \text{row 3} & 3 & 1 & 1 & 2 \\ \hline & \text{column 0} & \text{column 1} & \text{column 2} & \text{column 3} \end{array} \times \begin{array}{c|cccc} \text{row 0} & a_0 & a_1 & a_2 & a_3 \\ \text{row 1} & a_4 & a_5 & a_6 & a_7 \\ \text{row 2} & a_8 & a_9 & a_{10} & a_{11} \\ \text{row 3} & a_{12} & a_{13} & a_{14} & a_{15} \\ \hline & \text{column 0} & \text{column 1} & \text{column 2} & \text{column 3} \end{array}$$

Fig. 7. Matrix multiplication in MixColumns.

4.6 AddRoundKey

This step requires only `xor` operations. Our GPU version loads the 10 round keys into shared memory to improve performance when processing multiple blocks. By loading the round keys into shared memory, we avoid having to read the round keys from GPU global memory repeatedly.

4.7 Resistance to Timing-Attack

The CPU-based algorithm of Kasper and Schwabe is resistant to timing side channels due to the use of constant time operations [8]. By using a bitslicing approach, our algorithm is also resistant to timing side channels. All operations that involve key or data use bitwise operations whose execution time does not depend on the values of the data. In contrast, other GPU-based AES implementations use lookup tables whose execution time depends on the data, i.e., these operations are not constant time. Furthermore, the bitsliced implementations are also inherently immune to the cache-timing attacks, as discussed in [1,2,26].

5 Results and Conclusion

We summarize all recent results in Table 2, along with our result in the last row. This table shows we have the fastest GPU implementation among all reported results.

Considering that CC (Compute Capability) of these devices is a good indication of their architectural richness and computational power, we notice that

Table 2. Comparing recent implementations. CPU speeds are per core.

CPU	Bernstein and Schwabe [1]	Core 2 Quad Q6600	1.82 Gbit/s
	Kasper and Schwabe [8]	Core 2 Quad Q6600	2.06 Gbit/s
		Core 2 Quad Q9550	2.99 Gbit/s
		Core i7 920	3.08 Gbit/s
	OpenSSL 1.0.1e [25]	Core i7 2600	0.98 Gbit/s
		Core i7 2600 (AES-NI)	5.78 Gbit/s
GPU	Manavski [11]	GeForce 8800 GTX	8.28 Gbit/s
	Iwai et al. [6,7]	GeForce GTX 285	35.2 Gbit/s
	Nishikawa et al. [15]	Tesla C2050	48.6 Gbit/s
	Li et al. [9]	Tesla C2050	60.0 Gbit/s
	This implementation	GeForce GTX 480	78.6 Gbit/s

the first two devices (GeForce 8800 GTX and GeForce GTX 285) have their CCs as 1.0 and 1.3, respectively, while remaining two devices (Tesla C2050 and GeForce GTX 480) are both 2.0, however, our AES-ECB implementation on a device with the same CC is 62 % faster than that of Nishikawa et al. [15] and 31 % faster than that of Li et al. [9].

Moreover, our implementation is quite practical; it is used in the deterministic RNG portion of a successful Monte Carlo simulator for fracture computation in certain composite materials, as described in [10].

References

1. Bernstein, D.J., Schwabe, P.: New AES software speed records. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 322–336. Springer, Heidelberg (2008)
2. Bernstein, D.J.: Cache-timing attacks on AES (2005). https://cr.yp.to/antiforgery/cachetiming-20050414.pdf
3. Cook, D.L., Ioannidis, J., Keromytis, A.D., Luck, J.: CryptoGraphics: Secret Key Cryptography Using Graphics Cards. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 334–350. Springer, Heidelberg (2005)
4. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, Heidelberg (2002)
5. Harrison, O., Waldron, J.: AES encryption implementation and analysis on commodity graphics processing units. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 209–226. Springer, Heidelberg (2007)
6. Iwai, K., Kurokawa, T., Nishikawa, N.: AES encryption implementation on CUDA GPU and its analysis. In: 2010 First International Conference on Networking and Computing (ICNC), pp. 209–214. IEEE (2010)
7. Iwai, K., Nishikawa, N., Kurokawa, T.: Acceleration of AES encryption on CUDA GPU. Int. J. Netw. Comput. **2**(1), 131–145 (2012)
8. Käsper, E., Schwabe, P.: Faster and timing-attack resistant AES-GCM. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 1–17. Springer, Heidelberg (2009)

9. Li, Q., Zhong, C., Zhao, K., Mei, X., Chu, X.: Implementation and analysis of AES encryption on GPU. In: 14th IEEE International Conference on High Performance Computing and Communication and 9th IEEE International Conference on Embedded Software and Systems, HPCC-ICESS 2012, pp. 843–848 (2012)
10. Lim, R.K., Pro, J.W., Begley, M.R., Utz, M., Petzold, L.R.: High-performance simulation of fracture in idealized ‘brick and mortar’ composites using adaptive Monte Carlo minimization on the GPU (Manuscript, in preparation, November 2014)
11. Manavski, S.A.: CUDA compatible GPU as an efficient hardware accelerator for AES cryptography. In: IEEE International Conference on Signal Processing and Communications, 2007, ICSpPC 2007, pp. 65–68 (2007)
12. Matsui, M.: How Far Can We Go on the x64 Processors? In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 341–358. Springer, Heidelberg (2006)
13. Matsui, M., Nakajima, J.: On the power of bitslice implementation on Intel Core2 processor. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 121–134. Springer, Heidelberg (2007)
14. National Institute of Standards and Technology: Advanced Encryption Standard (AES), FIPS 197, November 2001
15. Nishikawa, N., Iwai, K., Kurokawa, T.: High-performance symmetric block ciphers on multicore CPU and GPUs. Int. J. Netw. Comput. **2**(2), 251–268 (2012)
16. NVIDIA: CUDA GPUs. https://developer.nvidia.com/cuda-gpus
17. NVIDIA: CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit
18. NVIDIA: GeForce 8800 GTX Specifications. http://www.nvidia.com/page/geforce_8800.html
19. NVIDIA: GeForce GTX 285 Specifications. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-285/specifications
20. NVIDIA: GeForce GTX 480 Specifications. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-480/specifications
21. NVIDIA: Tesla C2050 Board Specifications. http://www.nvidia.com/docs/IO/43395/Tesla_C2050_Board_Specification.pdf
22. NVIDIA: Next Generation CUDA Compute Architecture: Fermi, v1.1. (2009)
23. NVIDIA: Parallel Thread ISA, Version 2.3 (2011)
24. NVIDIA: CUDA C Programming Guide, Version 6.5, August 2014
25. OpenSSL Group: The OpenSSL Project. http://www.openssl.org
26. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
27. Rebeiro, C., Selvakumar, D., Devi, A.S.L.: Bitslice implementation of AES. In: Pointcheval, D., Mu, Y., Chen, K. (eds.) CANS 2006. LNCS, vol. 4301, pp. 203–212. Springer, Heidelberg (2006)
28. Yamanouchi, T.: AES encryption and decryption on the GPU. GPU Gems **3**, 785–804 (2007)