

AES and DES Encryption with GPU

Brandon P. Luken, Ming Ouyang, and Ahmed H. Desoky
Computer Engineering and Computer Science Department
University of Louisville
Louisville, KY 40292

ABSTRACT

Graphics processing units (GPUs) are powerful computational devices tailored towards the needs of the 3-D gaming industry for high-performance, real-time graphics engines. As the processing power of such GPUs increases so does the possibilities for other, non-graphics related applications to be implemented on them. With the increase over the years of sensitive data that must be stored securely, the use of encryption techniques has become widespread. Although block ciphers operate at a computational complexity of $\Theta(n)$ where n is the number of blocks the sheer size of the data to be encrypted has caused many to look for improvements in the encryption algorithms. Here NVidia's CUDA language is used to implement both AES and DES algorithms to encrypt documents composed of random bytes in approximately one third and one fourth of time required by traditional CPUs.

1. INTRODUCTION

As the computation power requirements of 3-D gaming community have increased so has the capabilities of graphics processing units (GPUs). In the past, most developmental tools used to design GPU software have been very specialized; concentrating on graphics only. However, in 2006, NVidia released a new generation of general purpose GPUs (GPGPUs) that greatly expanded their possible applications. With the release of NVidia's Compute Unified Architecture[5] (CUDA) extension to the C Programming language current single-threaded algorithms can be converted into parallel GPU programs. Such non-graphical applications of CUDA include bioinformatics, data mining, and information security.

As the amount of sensitive data being collected increases data managers are facing an interesting dilemma; how to store data securely while still being able to access it quickly. While the standard symmetric block ciphers of AES[3] and DES[2] are both secure and efficient in terms of time and memory requirements the amount of information that must be encrypted simultaneously leads to response times that are inadequate. Several performance

increases for these algorithms have been found but these too are not always quick enough. Since all pure block ciphers consist of the same steps being applied to different blocks of plain text acceleration using a GPU is the next logical step.

In this work three different versions of AES and DES have been tested. The first versions (cpuAES and cpuDES) are traditional CPU only implementations and serve as control of the experiment. The other versions do the majority of their computations on the GPU. One version (gpuAESgpu and gpuDESgpu) performs all steps in the encryption process on the GPU while another (gpuAEScpu and gpuDEScpu) performs all steps except key scheduling on the GPU. The difference in placement of the key scheduling algorithm is due to the non-parallel nature of the key scheduling. While key generation is not a computationally expensive process the placement of key generation should affect overall performance of the algorithm.

Only the encryption algorithms were implemented as the decryption algorithms should have the same computational complexity. As improvements continue to be made to the speed of the algorithms they are both implemented as originally published. The results reported herein serve as a benchmark for other implementations of AES and DES.

2. DATA AND METHODS

A. Data

The data encrypted by both algorithms are randomly generated byte streams. Each file was generated using C# 3.5's random number generator. Six different file sizes were used for the test; 1KB, 10KB, 100KB, 1MB, 10MB, and 100MB. The same set of files was used by each implementation of the algorithms.

The keys used by each implementation of an algorithm are also the same. All three standard key sizes of AES were used. The Keys used by this paper are listed in Table 1.

Table 1

Keys used by each implementation of the algorithms. key in hexadecimal

Algorithm	Key (Hexadecimal)
DES	3B3898371520F75F
16 Byte AES	E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FA
24 Byte AES	E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FAFAF2F3F4F5F7F8F9FA
32 Byte AES	E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FAFAF2F3F4F5F7F8F9FAFBFCFDFEFFF00102

B. cpuAES code[4]

Advanced Encryption Standard[3] encrypts data in blocks of 16-bytes using a key size of either 16-, 24-, or 32-bytes. Before encryption can begin the key must first be expanded. This is done using a key scheduling algorithm which is shown in [4]. The key scheduling algorithm will generate up to 320 8-bit “round keys” depending on the size of the key used.

After the round keys are generated the 16-byte block of plain text is aligned into a 2-D array before undergoing the encryption process. This process consists of a sequence of four different operations ShiftRows, SubBytes, MixColumns, and AddRoundKey. The exact functions of these operations are beyond the scope of this paper. The order in which they are applied is shown below.

C. gpuAEScpu code

Due to the memory and thread structure of CUDA enabled devices several changes had to be made to CPU program. CUDA executes functions, called kernels, in threads which are organized into groups called blocks which are further grouped into grids. All threads within a grid execute the same kernel function. Threads inside the same block execute simultaneously and may communicate with each other. However, blocks of threads must be autonomous and may be executed in any order. Up to 512 threads may be placed within the same block. More than one grid can be executed at a time.

The CUDA memory structure is divided into several different types of memory. Shared and register memory are the fastest and are only accessible to one thread or block respectively. Local and global memory are the slowest memory sources and are accessible by one thread or every possible block. Constant memory can be as fast shared memory, however, this is not guaranteed.

Constant, local, and global memory is all stored in the same memory bank, although constant memory is cached.

Due to the many different types of memory available the location of most data structures required by the program must be specified at design time. For this implementation of AES the sbox and round keys were stored in constant memory. This location was chosen over shared memory as it can be assigned at design time and can be assigned by the host CPU; two qualities shared memory lacks.

The type of data structures located inside the kernel itself must also be reconfigured. When CUDA executes a kernel it attempts to locate as many variables from inside the kernel as possible into the registers. However, the number of registers available to any one thread is very limited and when an open register is not available the variable is pushed to local memory. This causes increased latency. The user can not choose which variable is moved to local memory so various techniques must be used to reduce the register’s utilization. Certain techniques include limiting the scope of a variable or reuse of variables currently not being used. The use of 2-D arrays is also not recommended as the compiler does not know how to allocate registers for them. Thus the 2-D array that is used by AES must be converted into a 1-D array.

D. gpuAESgpu code

In this implementation, all steps of the encryption process are handled by the GPU. Here the first thread of every block executes the key scheduling algorithm. The key is stored in global memory and the round keys are stored in shared memory as they are generated. The actual encryption process is the same as the gpuAEScpu implementation.

E. cpuDES code[6]

Data Encryption Standard[3] is the predecessor of AES and encrypts data in blocks of 8-bytes using an 8-byte key. Before encryption the key is first expanded using a key scheduling algorithm which is shown in [6]. The key scheduling algorithm only uses 56-bits of the 64-bit key and generates sixteen 48-bit sub-keys. Note that this algorithm is widely considered obsolete although certain versions, such as triple DES[3], are still used today.

After the generation of sub-keys a sixteen round encryption process begins. The encryption phase of DES is a Feistel network that begins with an initial permutation, followed by sixteen rounds of a Feistel operation, and is ended by a final permutation. The exact operations of the encryption phase in beyond the scope of this paper.

F. gpuDEScpu

Unlike implementation of AES very few changes had to be made to DES encryption algorithm itself. However, the memory assignments still had to be made. In this implementation all eight sboxes were stored in constant memory. The sub-keys generated by the key scheduling algorithm were also stored in constant memory. The implementation of the key scheduling algorithm is shown in [3].

G. gpuDESgpu

In this implementation of DES all steps required in the encryption process were handled by the GPU. Here, the first two threads of each block handle the key scheduling responsibilities. The key is stored in global memory and the sub-keys are stored in shared memory. The actual encryption phase is the same as the gpuDEScpu implementation.

3. Results

The performance analysis was run on a windows XP service pack 3 machine using an Intel Core 2 Duo E8400 processor at a speed of 3GHz and with 8GB of RAM available. The GPU used was NVidia's Tesla C870 1.0 that has 16 multiprocessors running at 1.35GHz with 1.5GB of device memory. All implementations were compiled in Visual Studio 2008. The CUDA version used was release 2.1. The test data are the six random byte files described earlier. The computation time was taken using CUDA's built-in timer utility. Only the key scheduling and encryption processes were considered to be part of the execution, as well as any memory operations between the device and the host.

The results for AES are located in Table 2. All time values are in milliseconds. The results for DES are location in Table 3. For both algorithms and all key sizes the CPU is faster for the smaller files sizes. However, around the file size of 100KB, the GPU implementations catch and surpass the CPU only implementations. For AES the GPU implementations operate at around the same speed and at the larger file sizes complete the encryption process around 3.5 times faster than the CPU only. The DES implementations that use the GPU can encrypt the same data approximately 4.5 times faster than the CPU only version. For DES there is a clear distinction between the gpuDESgpu and gpuDEScpu versions as the later is faster than the former.

Table 2
Performance comparasions of CPU and GPU implementations of AES for each key size

File Size	1K	10K	100K	1M	10M	100M
16-CPU	.039	.351	3.56	36.1	358	3596
16-GPU GPU	.503	1.58	2.24	11.7	102	1006
16-GPU CPU	.480	1.55	2.22	11.5	101	1003
24-CPU	.044	.419	4.21	42.9	425	4226
24-GPU GPU	.607	1.69	2.42	13.3	119	1177
24-GPU CPU	.553	1.65	2.39	13.3	117	1169
32-CPU	.052	.488	4.91	49.9	498	5010
32-GPU GPU	.687	1.74	2.60	15.1	136	1349
32-GPU CPU	.623	1.85	2.67	14.9	134	1337

Table 3
Performance comparasions of CPU and GPU implementations of DES

File Size	1K	10K	100K	1M	10M	100M
CPU	.022	.179	1.79	18.2	182	1820
GPU GPU	.247	1.39	1.81	6.24	44.3	425
GPU CPU	.178	1.23	1.65	5.59	38.7	369

4. CONCLUSION

The GPU can be used as a cost efficient hardware performance enhancer for both the AES and DES algorithms. If the amount of data is large enough both algorithms can greatly reduce the encryption time required by both algorithms. Placement of the key scheduling algorithm, however, is not guaranteed to produce a significant effect on computation time. The gpuDEScpu implementation of the DES algorithm was faster than its other GPU counterpart because there are less reads to

global memory. However, while gpuAEScpu shares this same advantage the amount of round keys that must be passed from host to device greatly exceeds the number of DES sub-keys. Passing memory from the host to the device is by far the slowest possible memory operation. Any speed advantages gained from generating the keys on the host are lost when they are passed to the device. Since the block size of the DES cipher is half of the block size used by the AES cipher the DES algorithm uses twice as many thread blocks as AES. Thus, there are more opportunities to recoup the time lost by passing the sub-keys from host to device than there is for AES. For the largest file sizes gpuAEScpu does slightly improve on the execution time of gpuAESgpu. For even larger file sizes it may be possible that gpuAEScpu will be significantly faster.

For all keys sizes and all GPU implementations there is another trend that has presented itself for larger file sizes. The percent time decrease from the CPU only implementations to the GPU implementations will level off and no longer increase. For AES key sizes of 16, 24, and 32 the GPU algorithms will execute around 3.5, 3.6, and 3.75 times faster than the CPU implementation respectively. DES will run round 4.5 times faster the original algorithm. It is theorized that for every GPU implementation of AES and DES there will be a performance increase cap at some point. Improvements in the speed of these algorithms will only raise this cap. This theory is reinforced by Manavski's[1] implementation of an optimized version of AES on the same GPU. There the performance increase cap is around 5.4. This cap may extend to all block ciphers and similar algorithms.

5. REFERENCES

- [1] S. A. Manavski, "CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography," 2007, pp. 65-68.
- [2] National Institute of Standards and Technology (NIST), "FIPS 46-3: Data Encryption Standard (DES)," 1999.
- [3] National Institute of Standards and Technology (NIST), "FIPS 197: Advanced Encryption Standard (AES)," 2001.
- [4] P. Niyaz, "Advanced Encryption Standard (AES) Implementation in C/C++ ," 2008.
- [5] NVidia, "NVidia CUDA," 2009.
- [6] M. Roth, "DES and Triple-DES encryption /decryption Algorithm," 1998.