2008

Survey and future trends of efficient cryptographic function implementations on GPGPUs

Adrian Boeing
Edith Cowan University

DOI: 10.4225/75/57b26b3a40cb8
Originally published in the Proceedings of the 6th Australian Digital Forensics Conference, Edith Cowan University, Perth Western Australia, December 3rd 2008.

Survey and future trends of efficient cryptographic function implementations on GPGPUs

Adrian Boeing
School of Computer and Information Science
Edith Cowan University
a.boeing@ecu.edu.au

Abstract

Many standard cryptographic functions are designed to benefit from hardware specific implementations. As a result, there have been a large number of highly efficient ASIC and FPGA hardware based implementations of standard cryptographic functions. Previously, hardware accelerated devices were only available to a limited set of users. General Purpose Graphic Processing Units (GPGPUs) have become a standard consumer item and have demonstrated orders of magnitude performance improvements for general purpose computation, including cryptographic functions. This paper reviews the current and future trends in GPU technology, and examines its potential impact on current cryptographic practices.

Keywords

Cryptography, Graphics Processing Unit, GPU, RC4

INTRODUCTION

Graphics Processing Units (GPUs) have become a standard consumer item that provide high performance parallel processing. GPUs have demonstrated orders of magnitude performance improvements for cryptographic functions, similar or superior to the performance gains achieved by Application Specific Integrated Circuit (ASIC) approaches.

Although the high performance capabilities of ASIC systems are widely known, the difficulty of developing and obtaining these systems has hindered their acceptance. Thus, ASIC based hardware cryptography has typically only been available to a few large organizations.

Many cryptographic standards that can be efficiently cracked with hardware solutions are still in common use. These may be in use due to poor administration or to enhance the performance of the system. Vinadsius (2006) indicates that many network administrators choose to employ insecure security standards such as WEP (37%), or not apply security to their networks at all (57%). Internet server administrators often choose to employ lower security standards in order to gain performance (Lowenthal, 2001). Encryption key sizes of less than 40 bits are typical (Lowenthal, 2001). These choices are often made as it has been considered unlikely that an attacker would own the required hardware resources to dedicate to an attack.

In recent years graphics card processors have become standard consumer items, offering vast performance increases for certain problem tasks over traditional CPUs. The peak floating point performance of nVidia GPUs and Intel CPUs is illustrated in a graph in Figure 1. Cope et. al. (2005) compared the performance of GPUs with Field Programmable Gate Arrays (FPGAs) and found the GPU outperformed the FPGAs for a certain signal processing tasks.

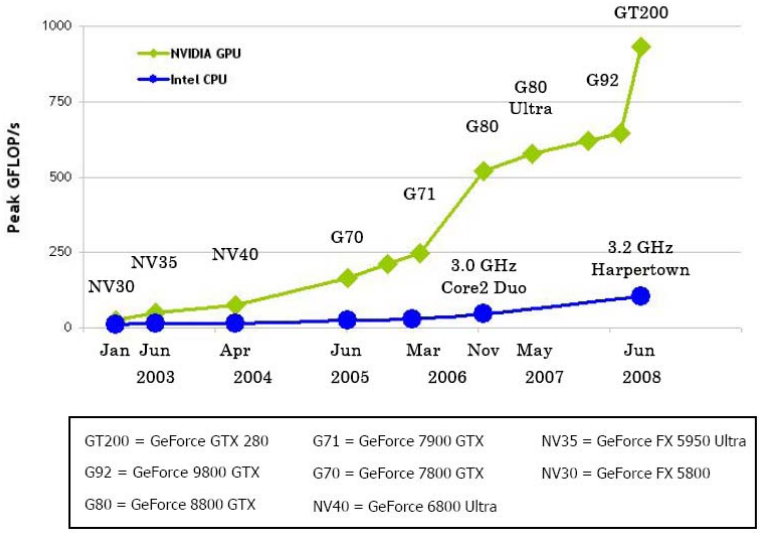

Figure 1 - Floating point performance of nVidia GPUs and Intel CPUs in billions of floating point operations per second. **(nVidia, 2008)**

In the remained of the paper GPU technology will be reviewed and the future trends of stream processing will be investigated. Cryptographic implementations on GPUs will then be discussed and contrasted to CPU and FPGA implementations. Finally, the RC4 algorithm will be investigated in detail and the conclusions will be presented.

GPU TECHNOLOGY REVIEW

General Purpose Graphics Processing Units (GPGPUs) are a type of stream computers that offer price effective high performance computing (Owens, et al., 2007). Unlike traditional CPUs, GPGPUs perform calculations on streams of data. This is similar to a Single Instruction Multiple Data (SIMD) vector processor, however the typical GPGPU processor is composed of a large number of independent processing elements, or threads. As a result the GPU can be seen as both a SIMD and Signal Program Multiple Data (SPMD) device. The exact implementation of the stream computing device is vendor specific and thus the capabilities of each vary.

The common feature of GPUs are a large number of processor cores. Each core will then contain a number of general purpose vector processors and execution contexts, an optimized memory processing unit (or texture unit), standard CPU components (such as instruction decoding) and specialized functionality for rendering graphics (such as the rasterizer and image blend units). This design enables the GPU to execute a large number of threads concurrently. The cache design for each GPU is carefully chosen to enhance performance for streaming data applications.

There are four major vendors that are producing streaming processors. The traditional GPU manufacturers are nVidia and ATI (purchased by AMD in 2006 (AMD, 2006)). The other manufacturers produce more general processors that are highly optimized for graphics tasks. These are the CELL processor, produced by Sony, Toshiba and IBM, and the Larrabee GPU produced by Intel.

nVidia was an early developer of GPUs for consumer systems, and pioneered a number of GPGPU technologies. The traditional target market for nVidia was accelerating rasterized graphics rendering, however modern nVidia GPUs target image processing, physics calculations and financial applications. The GTX 280 (nVidia), is the current top-end consumer GPU available from nVidia. It consists of thirty processing cores, each core containing an eight-wide SIMD processing unit. Each SIMD unit can perform three operations per clock cycle, and the device runs at 1.3 GHz. This results in an overall potential peak performance of just under one trillion operations per second (Fatahalian, 2008). This is the equivalent computational power as the worlds

fastest super computer from 1996, the Sandia Labs ASCI Red (TOP500.Org, 2008). It is worth emphasising that the retail price for this hardware is currently a few hundred dollars.

nVidia also provides support for a number of GPU programming environments, as well as providing their own language to enable high performance general computing for the GPU. Compute Unified Device Architecture (CUDA) is a language environment similar to C that provides extended functionality to program the GPU.

ATI is another GPU manufacturer with a strong history of applying general purpose processing techniques to GPU technology. ATI was acquired by AMD in 2006, and have since focused on combining GPU and CPU technology onto a single chip (AMD, 2006). Similar to nVidia's traditional GPUs AMD focuses on graphics processing as a primary purpose of their hardware, and design the GPU accordingly. AMD has provided double-precision floating point number support to their GPUs to encourage high performance scientific uses for their GPUs.

The current top end GPU model from ATI/AMD is the Radeon 4870. It contains ten processing cores, each with 16 SIMD units. Each SIMD unit can complete ten operations per cycle, and the chip runs at 750 MHz. This provides a total peak performance of 1.2 trillion operations per second (Fatahalian, 2008), almost 30% faster than nVidia's already impressive offering.

Similarly to nVidia, AMD provides support for programming their GPUs in a number of graphics contexts. AMD also provides its own environments, APIs and languages that support its GPU, this includes Brook+, Close To the Metal, Compute Abstraction Layer, and have announced support for OpenCL and Microsoft's DirectX 11 Compute Shader. Brook+ and OpenCL are both similar technologies to nVidia's CUDA, they are all C-like language environments with extensions to support the stream-based programming model through the use of compute kernels.

Both Intel and the STI (Sony, Toshiba, IBM) technologies are aimed at performing, or supporting graphics processing, however are also intended to be far more applicable to general purpose computing. Intel's upcoming Larrabee GPU technology is based on x86 technology, thus making it directly compatible with most existing PC software (Seiler, et al., 2008). STI's CELL processor technology is designed to be a more general processor that operates alongside the GPU, rather than aimed at displacing it.

The CELL processor is available in a wide range of configurations. The most readily available CELL processor is inside the Playstation 3. The PS3 CELL processor contains one power PC CPU and eight Synergistic Processor Elements (SPE) cores. Each SPE has four SIMD units and each unit can complete two operations per second. The SPEs run at 4000 MHz, giving a total performance of 0.25 trillion operations per second.

Whilst the SPE peak performance values appear far less impressive than the performance of the competing GPU technologies, the actual performance difference in an application may not be as great. This is due to the difficulty in programming the GPUs in such a way as to achieve the optimal performance.

Intel has announced the architecture for their new GPU technology scheduled for release in 2009. Some of the exact details on the chips architecture are presently unavailable (Seiler, et al., 2008) (Fatahalian, 2008). The Larrabee chip contains an unknown number of processors, however some experts estimate the likely number of processors for their top end model to be 32 cores (Fatahalian, 2008) and 24 cores at launch (Gruener & Polkowski, 2007). Each processor is based on the Pentium x86 design, however contains a new vector processor design. Each vector processor contains sixteen processing units, with each unit capable of completing two operations per second. The exact clock speed of Larrabee is also unknown at this point, however industry sources report that it will range from 1.7 to 2.5 GHz (Gruener & Polkowski, 2007). This gives a total of around 2 trillion operations per second. This figure is comparable to the expected performance projections for AMD and nVidia GPUs.

Technology	Specifications	Peak billion operations per second
nVidia GTX 280	30 (cores) * 8 (SIMD units) * 3 (operations per unit) * 1300 MHz (clock speed)	936
ATI/AMD Radeon 4870	10 (cores) * 16 (SIMD units) * 10 (operations per unit) * 750 MHz (clock speed)	1200
Sony, Toshiba, IBM CELL	8 (SPE cores) * 4 (SIMD units) * 2 (operations per unit) * 4 GHz (clock speed)	256
Intel Larrabee (predicted)	32 (cpu cores) * 16 (SIMD units) * 2 (operations per unit) * 2 GHz (clock speed)	2000
Intel, Core 2 E7200	2 (CPU cores) * 4 (SIMD units) * 2 (operations per unit) * 2.53 GHz (clock speed)	40

Table 1 – Peak performance characteristics of computing technology

To provide a benchmark figure, the current Intel Core 2 E7200 CPU provides roughly 40 billion operations per second, 50 times less than the projected performance of their GPU offering.

The implications of Intel's entrance to the GPU market and AMD's consolidation with ATI is that both manufacturers standard CPUs will be integrated with GPU technology. This places an extremely large processing power at the hands of every consumer PC.

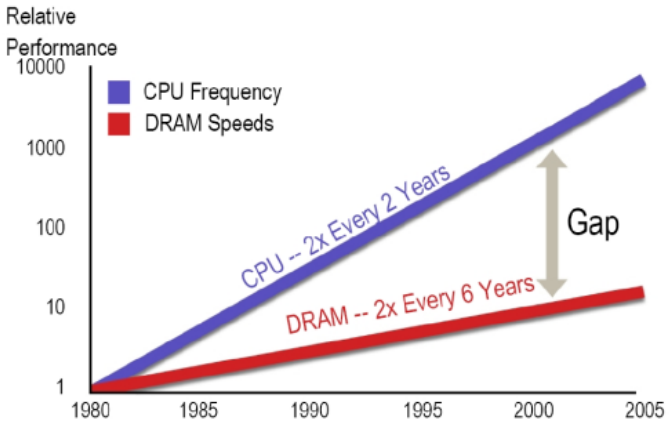

Figure 2 Trends of CPU and Memory performance increase (Mayo, 2008)

The processing tasks that GPU processors are aimed at tend to be highly parallel (such as rasterization). Therefore GPUs can gain performance in each generation by increasing the number of cores. For example, the nVidia 8600GT has 4 multiprocessors, the 9800GX2 has 16 multiprocessors, and the GTX280 has 30 multiprocessors. As with other integrated circuit technology GPUs follow the benefits from improved processes, as predicted by Moore's Law. Thus, we can reasonably expect GPU FLOP performance to continue to exponentially increase in the future at a very high rate.

Since GPUs are not aimed at general processing tasks we can expect that the memory processing capabilities of the GPU will not be expanded at the same rate. It is likely that the architectural structure of GPUs will follow other stream processors, such as the Cell's SPEs. Current GPUs have limited memory caches (eg: 16kb for a nVidia GeForce 9800GX2) compared to cache sizes upwards of four megabytes for typical CPU's. The CELL SPEs have 256KB local memory (Flachs, et al., 2005). Memory technology does not increase at the same rate as CPU technology (See Figure 2), so it is highly likely that GPU caches will remain small and limit the performance of general purpose GPU computing in the future.

REVIEW OF GPU CRYPTOGRAPHY

Applying cryptographic functions on hardware designed for graphics processing is not a recent idea. An early application of cryptography to GPUs was in 1999, where a custom built GPU architecture "PixelFlow" was used to do a brute force cryptanalysis of the DES and RC4 ciphers (Kedem & Ishihara, 1999). This demonstrated that devices designed for graphics processing could also be leveraged for cryptanalysis.

A common approach to improving the performance of cryptographic functions is to implement them on Application Specific Integrated Circuits (ASICs), such as Field Programmable Gate Arrays (FPGAs). Jarvinen et al. (2005) present an overview of 32 different AES implementations on Xilinx Virtex FPGAs (Jarvinen, Skytta, & Tommiska, 2005). The throughput of the implementations reviewed ranged from 23.57 Gbps to 0.215 Gbps, this reflects the range in the implementations performance depending on the skill of the implementers and the design goal (eg: minimize power consumption, etc.). Similar results were presented for common cryptographic algorithms such as IDEA, MD5, and SHA.

Moss et al. (2007) implemented a 1024bit modular exponentiation algorithm (a key part of the RSA public key cryptosystem) on a nVidia 7800-GTX using the OpenGL Shading Language. The GPU program was written to employ the 24bit mantissa of the floating point values and the GPU special texture-memory. Five versions of the GPU algorithm were tested:

- an un-optimized 12bit-half-word version (A)
- a texture-memory optimized half-word version (B)
- a texture-memory optimized 24bit-full-word version (C)
- a texture-memory optimized smooth-word version, (D)
- a texture-memory optimized smooth-word version with decomposed-modulus operations. (E)

The performance of their system is given in **Table 2**. These results indicate that memory optimizations provided greater performance enhancement than the algorithmic calculation changes.

CPU	17ms
GPU A	21.5ms
GPU B	12.5 ms
GPU C	11.5ms
GPU D	9.8ms
GPU E	5.7ms

Table 2 – Performance of 1024 bit modular exponentiation in ms **(Moss, Page, & Smart, 2007)**

An early AES GPU implementation was presented by Cook, et al in 2004. The AES implementation was tested on an nVidia Gefore3 TI 200, ATI Radeon 7500, and a nVidia TNT 2 GPU. The peak performance for the XOR component of their system was 105MBps, 41.5MBps, and 38.4 MBps respectively. Whilst the CPU performance of a 1.8Ghz Pentium 4 processor was 139 MBps. The AES implementation could not run completely on the GPU, and still employed the CPU for some processing. The peak CPU performance measured for the AES program was 64MBps, whereas the best GPU performance obtained was 1.53MBps with the Geforce3.

In 2007, Harison and Waldron (2007) implemented an AES cipher on an nVidia GeForce 6600GT and an nVidia GeForce 7900GT using OpenGL Pixel Buffer Objects. The AES XOR operation and memory access patterns were investigated. Three XOR approaches were tested:

- Calculating the 8bit by 8 bit XOR operation via a 65kb lookup table
- Decomposing the XOR operation into 4bit sections, thus reducing the lookup table size
- Using the final render phases internal XOR operation and render buffers

Three AES memory access techniques were tested:

- Multi Input Gather, where 4 independent texture blocks operate simultaneously
- Single Input H-Gather, where 4 consecutive texture elements from one block are read
- Single Input S-Gather, where 4 texture elements that are organized within a 2D section are read from one block

Technique	8bit XOR	4 bit XOR	Native
Multi Input	25.86	39.23	108.86
S-Gather	26.06	39.18	N/A
H-Gather	25.99	39.08	N/A

Table 3 – AES results for a GeForce 7900GT in Mbytes/s

The major limiting factor from the Harison and Waldron AES implementation was the use of native XOR operation, and was not limited by the memory access speeds. This differs from Cook et al., likely due to the newer GPU used which provided greater general purpose capabilities.

Manavski (2007) also implemented an AES cipher in CUDA and tested the performance on an nVidia 8800GTX compared to an OpenGL implementation on the ATI X1900. The performance was compared to the AES algorithm implemented in the OpenSSL library on a 3.0 GHz Pentium 4 CPU.

The CUDA language enabled the full use of 32bit XOR operations, removing the need for XOR lookup tables present in OpenGL based implementations. Precomputed T-box values were stored in the GPU's faster constant memory, and the input and output buffers of the routines were stored in the GPU's fast shared memory. As a result of these optimizations, a 19.6 fold speedup over the CPU performance was reported.

The MD5 cipher has also been implemented on GPUs. Tzeng and Wei (2008) implemented an MD5 cipher for the purpose of generating white noise in computer graphics applications. The MD5 cipher was selected over other cryptographic algorithms such as AES since it has higher memory access requirements than MD5 (Tzeng & Wei, 2008). Two versions of the MD5 algorithm were implemented, a reference version and a memory optimized version with loop unrolling.

The system processed 1024x1024 keys in 3821.5ms for the CPU version, 229.7ms for a GPU implementation, and 6.3ms for a GPU optimized version. The large speedups were gained due to the suitability of the MD5 algorithm to GPU processing. This leveraged the low memory requirements of the MD5 algorithm and the high parallel computation speed of the GPUs multiprocessor architecture.

Commercial and open-source products have been released which provide GPU-based MD5 cryptography. A benchmark of competing GPU and CPU MD5 products was completed by Aleksandrovich (2008). Six different GPU MD5 implementations were benchmarked, the best achieving 350 million keys searched per second.

There have been a number of different cryptographic algorithms implemented on GPUs. The performance characteristics of the reviewed implementations is summarized in **Table 4**. It is clear that the newer GPUs outperform earlier implementations, and that the GPU implementations speeds increase drastically when a new programming technology is applied (eg: CUDA). Some algorithms such as the MD5 algorithm received very large speedups on GPUs. Likewise the AES performance increased dramatically when the programmability of the GPU increased. These algorithms are computationally intensive, and few memory intensive algorithms have been investigated.

Author	Algorithm	Manufacturer	GPU Model	Model Year	API	GPU Speedup
Moss et al.	RSA-MOD	nVidia	GeForce 7800GTX	2005	OpenGL	3.77
Cook et al.	AES-XOR	nVidia	GeForce 3TI 200	2001	OpenGL	0.76
Cook et al.	AES-XOR	ATI	Radeon 7500	2000	OpenGL	0.30
Cook et al.	AES-XOR	nVidia	TNT 2	1999	OpenGL	0.28
Harison et al.	AES	nVidia	GeForce 7900GT	2006	OpenGL PBO	2.34
Harison et al.	AES	nVidia	GeForce 6600GT	2004	OpenGL PBO	0.98
Manavski.	AES	nVidia	8800GTX	2006	CUDA	19.60
Tzeng et al.	MD5	nVidia	GeForce 8 Series (Unspecified)	2006	GLSL	606.59
Boeing.	RC4	nVidia	GeForce 9800GX2	2008	CUDA	8.57

Table 4 – GPGPU Cryptography implementations. GPU speedup is given relative to the baseline CPU performance used in the original article.

RC4 CRYPTOGRAPHY

The RC4 cipher is commonly used to encrypt data communications, such as the Wired Equivalent Privacy (WEP) and Wi-Fi Protected Access (WPA) protocols. It is also used in Microsoft's Remote Desktop Protocol (Microsoft, 2008), and other standard desktop products such as Word and Excel (Schneier, 2005). The RC4 algorithm is a memory intensive cipher.

Recently, Kwok and Lam (2008) presented a highly parallelized FPGA based RC4 implementation that achieved $1.07x10^7$ keys per second, requiring approximately 28.5 hours to break a 40-bit RC4 encryption (Kwok & Lam, 2008). Simulation results for a high end FPGA chip indicated the system could achieve more than twice the given performance. In 2002 Tsoi et. al. presented a comparison of RC4 CPU implementations and their FPGA implementation, the FPGA managed to search $6.06x10^6$ 56 bit RC4 keys per second, approximately fifty times faster than the fastest CPU implementation they tested.

There have not been many RC4 implementations implemented on GPUs due to the high memory requirements of the algorithm, making it a difficult algorithm to speed up on the GPU architecture.

The RC4 algorithm operates in three steps. The initial step is the key-scheduling algorithm which initializes the data structures required for the encryption process. This is a 256 byte "S" permutation array. It is initialized with a sequence of values from 0 to 255, and then permuted according to the given RC4 key. The pseudo code for the key-scheduling algorithm is given in **Listing 1**.

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap(S[i],S[j])
endfor
```

Listing 1 - The RC4 key-scheduling algorithm

The next step is to generate the random byte stream with the Pseudo-Random Generation Algorithm (PRGA). In each iteration, the PRGA increments i, adds the value of S pointed to by i to j, exchanges the values of S[i] and S[j], and then outputs the value of S at the location S[i] + S[j] (modulo 256). Each value of S is swapped at least once every 256 iterations. The final step is to XOR the output of the PRGA with the input plaintext.

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap(S[i],S[j])
    RandomByte = S[(S[i] + S[j]) mod 256]
    output =  RandomByte XOR input[i]
endwhile
```

Listing 2 - The RC4 pseudo-random generation algorithm

Unlike other encryption algorithms the RC4 algorithm makes heavy use of pseudo-randomly swapping bytes within a memory buffer. This implies the RC4 algorithm will be largely memory bound, rather than computation bound.

The RC4 algorithm was implemented in CUDA 2.0 on a nVidia GeForce 9800GX2. Four versions of the algorithm were implemented. A naive RC4 implementation for the GPU as well as three optimized versions. The first optimization was to employ the GPU's multiprocessors faster shared memory to store the S array. This provided a significant speed up. The third optimization was to align the memory structures to fit within the GPU's cache, and finally some of the mathematical operations of the RC4 algorithms were optimized, such as employing bitwise operations rather than the modulo operation.

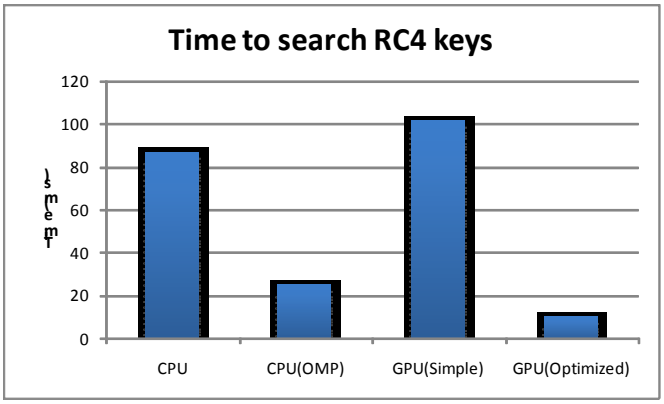

Figure 3 – Performance of RC4 key search on CPU and GPU architectures for searching 65 thousand RC4 keys

The CPU RC4 implementation achieved a 3.5x speedup when optimized for a four processor system. The naive GPU implementation was in fact slower than the CPU RC4 version. Simply porting code directly from one platform to another did not achieve a speedup, illustrating the need for GPU-specialized optimizations due to the nature for the GPU processing cores and memory system. After optimization the GPU achieved a ten fold speedup over the CPU RC4 implementation.

The impact of the different optimizations is illustrated in **Figure 4**. The largest performance gain came from placing the RC4 permutation array into the GPU's shared memory. The impact of the memory system relative to the integer maths performance of the system was investigated by varying the block size. The block size represents the number of RC4 permutation arrays that would be stored in one of the GPU's multiprocessors.

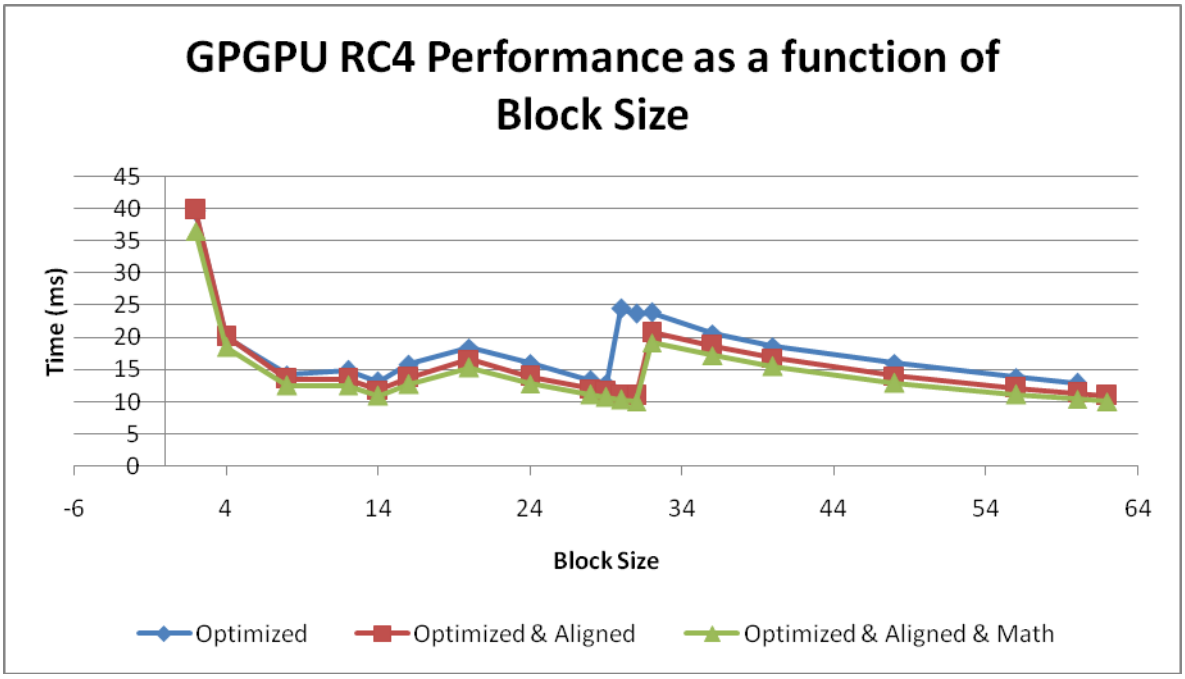

Figure 4 – GPU performance as a function of the problem block size

The results show a clear relationship between the block size and the performance of the system. A noticeable jump occurs at a block size close to 32. This indicates the memory accesses of the RC4 algorithm is a key factor in determining its performance.

The large influence of the memory characteristics can be further drawn from the performance gains from aligning the memory structures relative to optimizing the maths operations. Compacting the memory structure again provided a 14% increase in performance, whereas optimizing the mathematical operations provided only an 8% increase in performance over the memory optimized version. Overall the GPU performed $6.5x10^6$ RC4 key searches per second. This is approximately 40% slower than current state-of-the art FPGA implementations (Kwok & Lam, 2008) .

CONCLUSION

GPGPUs are a very good platform for the highly parallel calculations used in cryptographic key searches. Very high performance gains have been seen for computationally intensive ciphers such as MD5 and AES. The GPGPU performance has been comparable or superior to more traditional FPGA based acceleration approaches. The overall performance gain of the RC4 cryptographic implementation was lower than that of other cryptographic algorithms. The results indicate that this is largely due to the memory intensive nature of RC4. The RC4 algorithm operates on 128 byte buffers, whereas the AES algorithm operates on only 4x4 byte buffers.

The prevalence of GPU technology and the indication by CPU manufactures to incorporate the technology on-chip will result in very high performance cryptographic accelerators to be ubiquitous. Cryptographic standards will need to be altered to keep up with the pace of computing technology. New cryptographic functions and standards should be memory intensive in order to limit the achievable performance gains by future GPU and stream computing technology.

REFERENCES

Aleksandrovich, S. M. (2008, 07 25). *MD5 bruteforce benchmark.* Retrieved 10 24, 2008, from http://3.14.by/en/read/md5_benchmark

AMD. (2006, July 24). *AMD and ATI to Create Processing Powerhouse*. Retrieved August 1, 2008, from http://www.amd.com/us-en/Corporate/VirtualPressRoom/0,,51_104_543~110899,00.html

AMD. (2006, October 25). *AMD Completes ATI Acquisition and Creates Processing Powerhouse*. Retrieved 09 1, 2008, from AMD News Room: http://www.amd.com/us-en/Corporate/VirtualPressRoom/0,,51_104_543~113741,00.html

Cope, B., Cheung, P., Luk, W., & Witt, S. (2005). Have GPUs made FPGAs redundant in the field of Video Processing? *Field-Programmable Technology*, (pp. 111-118).

Fatahalian, K. (2008). Running Code at a Teraflop: How GPU Shader Cores Work. *ACM SIGGRAPH 2008 - Course Notes: Beyond Programmable Shading.*

Flachs, B., Asano, S., Dhong, S., Hotstee, P., Gervais, G., Kim, R., et al. (2005). A streaming processing unit for a CELL processor. *IEEE International Solid-State Circuits Conference*, (pp. 134-135).

Gruener, W., & Polkowski, D. (2007, June 1). *Intel set to announce graphics partnership with Nvidia*. Retrieved 09 1, 2008, from tgdaily: http://www.tgdaily.com/content/view/32282/118/1/1/

Harrison, O., & Waldron, J. (2007). AES Encryption Implementation and Analysis on Commodity Graphics Processing Units. *Workshop on Cryptographic Hardware and Embedded Systems. 4727*, pp. 209-226. Springer-Verlag.

Jarvinen, K., Skytta, M., & Tommiska, J. (2005). Comparative survey of high-performance cryptographic algorithm implementations on FPGAs. *IEEE Proceedings on Information Security , 152* (1), 3-12.

Kedem, G., & Ishihara, Y. (1999). Brute Force Attack on UNIX Passwords with SIMD Computer. *Brute Force Attack on UNIX Passwords with SIMD Computer*, (pp. 93-98). Washington.

Kwok, S., & Lam, E. (2008). Effective Uses of FPGAs for Brute-Force Attack on RC4 Ciphers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems , 16* (8), 1096-1100.

Lowenthal, B. (2001, January 19). *Best Practices in HTTP Security*. Retrieved October 1, 2008, from Oracle: www.oracle.com/technology/products/ias/pdf/best_practices/security_best_ practices.pdf

Manavski, S. A. (2007). CUDA compatible GPU as an efficient hardware accelerator for AES cryptography. *Proc. IEEE International Conference on Signal Processing and Communication*, (pp. 65-68).

Mayo, K. (2008, September 23). *Multi-Core Computing: Why you need to think in parallel.* Retrieved 10 24, 2008, from http://www.ivec.org/pages/tiki-download_file.php?fileId=120

Microsoft. (2008, 09 25). *Remote Desktop Protocol*. (Microsoft) Retrieved 10 23, 2008, from MSDN: http://msdn.microsoft.com/en-us/library/aa383015.aspx

Moss, A., Page, D., & Smart, N. P. (2007). Toward Acceleration of RSA Using 3D Graphics Hardware. In *Cryptography and Coding.* Springer Berlin / Heidelberg.

nVidia. (2008). *nVidia CUDA Programming Guide 2.0*.

nVidia. (n.d.). *NVIDIA GeForce GTX 280*. Retrieved 09 01, 2008, from http://www.nvidia.com/object/geforce_gtx_280.html

Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., et al. (2007). A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum , 26* (1), 80-113.

Schneier, B. (2005, 1 18). *Microsoft RC4 Flaw*. Retrieved 10 23, 2008, from Schneier on Security: http://www.schneier.com/blog/archives/2005/01/microsoft_rc4_f.html

Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., et al. (2008). Larrabee: a many-core x86 architecture for visual computing. *ACM Transactions on Graphics , 27* (3).

TOP500.Org. (2008, 9 1). *Sandia National Laboratories ASCI Red*. Retrieved 9 1, 2008, from Top 500 Supercomputer Sites: http://www.top500.org/system/4428

Tsoi, K. H., Lee, K. H., & Leong, P. H. (2002). A Massively Parallel RC4 Key Search Engine. *Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* , p. 13.

Tzeng, S., & Wei, L.-Y. (2008). Parallel White Noise Generation on a GPU via Cryptographic Hash. *i3D* (pp. 79-88). ACM.

Vindasius, A. (2006). Security state of wireless networks. *The 10th International Conference ELECTRONICS, 7*, p. 4. Kaunas.

COPYRIGHT