Efficiently Mapping the AES Encryption Algorithm on GPUs

Wagner M. Nunan Zola Luis C. Erpen De Bona

Federal University of Paraná, Department of Informatics, Brazil

Abstract

Warped AES is a high performance heterogeneous GPU/CPU-SSE parallel method for encryption using GPUs. Considering the performance of encryption in GPU memory alone, our algorithm outperforms current published implementations on comparable hardware. In our ongoing research, we have also devised a speculative method for high throughput encryption on GPUs, while preserving low latency to client CPU applications. In this work we emphasize on techniques used to efficiently map the AES CTR encryption tasks for parallel execution on GPUs.

Keywords: Parallel AES CTR encryption, GPU Accelerated Internet, Advanced Encryption Standard.

Authors' contact:
{wagner,bona}@inf.ufpr.br

1. Introduction

Data encryption is an essential characteristic on computing systems. In the design of next generation distributed systems there is a growing trend in considering scalable encryption as an essential characteristic [McGrew and Viega; Kounavis at al. 2010]. Given these facts, additional challenges to the scalability of such systems take place, since the encryption algorithms have high computational costs.

Warped AES (WAES) is a high performance heterogeneous parallel algorithm for encryption using GPUs. Our methods enhance scalability of the Advanced Encryption Standard (AES) algorithm of symmetric encryption keys in heterogeneous computing environments with parallel SIMT (Single Instruction Multiple Thread) hardware. WAES heterogeneous GPU/CPU-SSE approach and speculative method [Zola and Bona] minimizes encryption or decryption latency even when working with plaintext on CPU memory. Acceleration is obtained for any number of flows and also in the case of a single "sequential" flow. Low latency of encryption is particularly important on high performance applications (e.g. secure MPI [Ruan at al. 2010]) and high speed networking. To the best of our knowledge WAES is the first work to demonstrate very low latency as well as high bandwidth, with small number of encryption contexts as well as with many contexts. WAES scales well with the number of GPU cores. This is due to the speculative method that computes many AES state blocks ahead of time and a caching scheme to buffer these blocks in either GPU memory or CPU memory, whichever is chosen for the application.

In this work we emphasize on techniques used to efficiently map the AES CTR encryption tasks for parallel execution on GPUs. Following we present basic concepts of AES encryption. In session 3 we present the parallel algorithm WAES. Related work is discussed in section 4. In section 5 we present the the experimental results of our study.

2. AES Encryption in CTR Mode

We resume a simple view of AES CTR mode encryption in Figure 1, due to space limitations. Also, in this section we only briefly discuss advantages of this encryption mode with respect to the commonly used CBC mode. The CBC encryption mode is commonly "perceived" as safer than CTR but, in fact, the work in [Bellare at al. 1997] shows that the concrete security bounds obtained with CTR, using a block cipher, are no worse than what is accomplished by using CBC-mode encryption. In fact, the current implementation of the SSH protocol, in OpenSSH, uses CTR as the default encryption mode. Besides, CTR allows direct addressing (random-access) and can execute in parallel. CBC is inherently sequential, encryption of a block depends on results of all previous blocks in one flow.

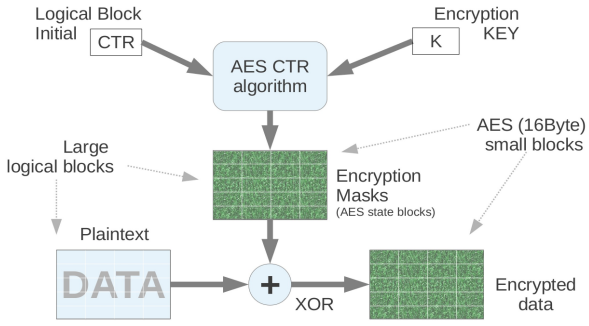

Figure 2: A simple view of AES CTR encryption

3. Mapping AES CTR to CUDA GPUs

The WAES algorithm is a parallel SIMT implementation for ciphering multiple AES contexts in CUDA architecture. It follows the reference implementation method for 32bit machines designed by the AES authors and formalized in [NIST 2001]. This implementation was also followed in OpenSSH.

We present in figure 2 a view of WAES working on contiguous AES blocks with no communication among threads. Working independently with thread

IDs, we can make CTR embarrassingly parallel. A counter block is a 16 byte number. Using thread IDs, adjacent AES blocks can be worked by separate threads with no use of atomic instructions or barriers. The algorithmic transformations necessary to make this scheme work is discussed in section 3.2.

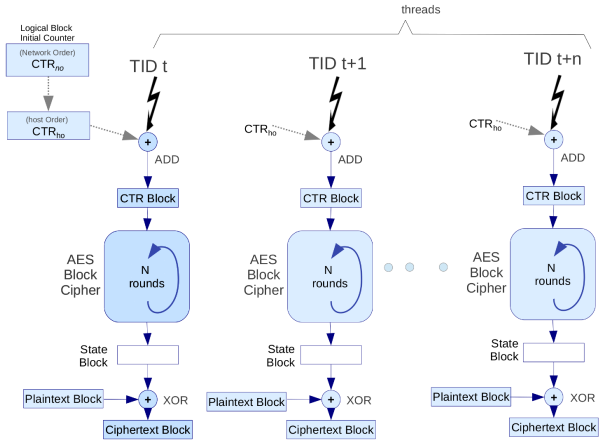

Figure 2: WAES parallel CTR Example of image

Dividing the encryption of only one AES block into the algorithm's steps or rounds results in too small granularity of work to be done in parallel by several processors, or presents difficulties related to the SIMT architecture. The main problems in this approach are the shuffling of bytes operations present in the algorithm, which implies non coherent (non coalesced) accesses to memory, and also the need for synchronizations among several threads to correctly obtain intermediary results. For this reason WAES does the encryption of many AES blocks in one thread, one AES block at a time, generating encryption results (state) in GPU memory or in CPU mapped memory. In fact, 32 threads operate the same instructions simultaneously in one warp scheduling unit of CUDA multiprocessors. These threads in a warp work different AES blocks in parallel in one multiprocessor, but WAES maintains a small amount of threads. The best performance was observed with the amount of threads slightly greater than the number of processors. With this mapping, we can generate more work to be done by each processing core, and very little synchronization is necessary.

With fewer threads we can maintain in registers all the necessary variables for ciphering one AES block and avoid global RAM access latencies, as CUDA automatically spills register values to slower memory on demand, when there is pressure over the maximum amount of registers. In the case of WAES the number of registers is sufficient for encrypting an entire block, if we tune the proper amount of threads. For this reason, we have conducted many experiments with the grouping of threads (thread grids), until the maximum performance is obtained.

One round of the AES 16 byte block cipher in WAES is depicted in figure 3. The block is maintained in four 32 bit registers w0 to w3, 4 bytes per word. The bytes are separated with bit masking operations. Afterwards these bytes are shuffled (AES ShiftRows) using bit shifting operations and used to index tables. The position of the byte in the block determines the proper table to use in indexing. Four precomputed tables (T0 to T3) maintained in shared memory efficiently store the coefficients needed for the rotation of bytes steps of AES (MixColumns). The results of indexing tables are 32bit words that need to be operated with XOR (exclusive-or). The last operation is done with the cypher key expanded in CPU for the specific round.

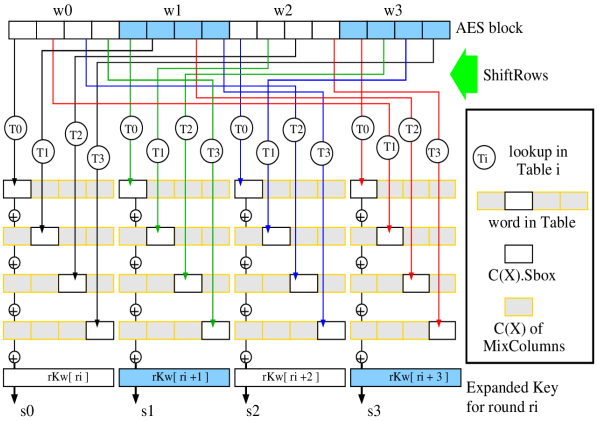

Figure 3: One round of AES (instruction level parallelism and no special purpose instructions)

The encoding of the AES block is symmetrical, being used both in encryption and in decryption. For this reason, encryption and decryption can occur in parallel on SIMT architecture with no divergence. This is an aside observation, for the fact that encryption and decryption can occur on the GPU for AES CTR without the need to run different GPU kernels. The result of encrypting one block are four words denoted s0 up to s3 . In the CTR mode these words should be operated via exclusive or with the corresponding block of plain text (on encryption) or with cipher text (in decryption mode). This last operation can be done on the GPU, or in CPU. In this version of the heterogeneous algorithm WAES, we do this operation efficiently using CPU SSE instructions.

The 32-bit algorithm maintains four small tables, which makes possible its allocation on shared memory. Each multiprocessor has a copy of the 4 necessary tables, stores by the CPU via the slower store called texture memory. On initialization, WAES makes parallel copies of these tables in the shared memory of each multiprocessor of the GPU. One of the key performance factors in WAES is on keeping the AES state block in registers making the shuffling operation faster. This is certainly a key factor for performance in 32bit implementations of AES. Executing on on parallel SIMT architecture, due to entanglement and access conflicts, this operation is particularly disastrous if done with global memory.

On the other hand, WAES outperforms other implementations [Iwai at al. 2010] of AES in GPUs

that maintain state in registers. We believe that this results from a lower computational cost of WAES in SIMT hardware by a reorganization in CTR update steps of the AES algorithm. As described in the next section, by maintaining CTR in network order and also in host (GPU) order, WAES introduces more register pressure to the SIMT hardware which implies in configuring smaller number of threads per GPU multiprocessor and working with persistent threads [Gupta at al. 2012] to execute AES rounds. Conversely this algorithmic reorganization incur in lower total instruction count for AES rounds and results in higher performance.

3.2 Communication-Avoiding CTR Counter Update

The CTR counter block is normally kept in ”network order” (”big endian”) and thus used to encrypt the AES block. The WAES kernel maintains CTR in registers, one copy per thread, in both host (”little endian”) and network order, as explained bellow.

The increment in CTR counter is done per thread for each AES block. We would have the need for synchronization between threads after encryption of each AES block if CTR was kept in a global way. With the encryption of multiple blocks in parallel, this counter is only updated on the host, adding the total amount of blocks encoded in the parallel processor. Each thread obtains an appropriate number to be added to the counter. This number is based on the thread identifier. Thus barriers are avoided between threads. Atomic fine-grain increments are also avoided, if only because such operations could not be done with ease, since the counter is of the same size of the AES block.

This increment of the CTR could be operated with the block kept in network order as is done in OpenSSH, byte by byte with a loop and other test-and-branch instructions. In WAES we chose to keep CTR in both formats, network and host order. The increment is then made directly in host order in the each GPU core using four 32-bit integers kept in registers. The counter is then converted to ”network order” before heading to the encryption block. In this format it was possible increment CTR without deviations or divergence in the execution of threads (details in [Zola and Bona]). In-deed, as we can review from fig. 3 the conversion back to network order has zero cost with a modified ShiftRows phase only in the first round, to seek the by-tes in host order instead. All subsequent rounds works with CTR in network order without modifications in ShiftRows. There is also no need to change the key scheduled for each round in network order format.

4. Related Work

There have been considerable interest in speeding up the encryption processes [Bernstein and Schwabe] in various architectures. Acceleration of AES encryption and decryption was first accomplished by Manavski [2007] with the initial version of the in the CUDA general purpose parallel programming environment. The work of Zola and Bona [2012], shows the performance of WAES in comparison to other published work [Harrison and Waldron] running on legacy (e.g. 8800 GTX) GPUs and recent work [Iwai at al. 2010].

The research by Osvik at al. [2010] compares a high throughput parallel AES method on GPU with implementations other architectures. This work focus on a trivially parallel method by encrypting one flow per thread, and is able to encrypt a huge number of simultaneous flows. This many flows method is specifically suited for cryptanalytic or key search applications. Working in parallel with only one thread taking care of each encryption flow achieves high aggregate throughput in case we have many thousand flows but presents high latency in any case, independently of the number of flows.

5. Experimental Results

For measuring efficiency of WAES methods we conducted experiments to enable comparisons of the different techniques. Our development/test machine was an AMD X2 CPU clocked at 2.2 Ghz and 800 Mhz DDR2 RAM. Measurements made at the final stage of the encryption done with SSE instructions on the CPU, in the case of CPU caching AES precomputed buffers, confirms that CPU RAM bandwidth is also a bottleneck for encryption, since AES block computing is now much faster in GPUs. We also have tested WAES in high-end Intel Xeon X5570 servers, DDR3 1333 Mhz RAM and PCI express 2.0 x16 using Fermi GPUs. PCI express on these machines achieve transfer peaks at 64 Gbps. In this setup, WAES encrypts with NVIDIA Tesla M2050 GPUs at peak throughput **of 48.1 Gbps in CPU memory.** This is in excess to fill 10 Gbps ethernet connection. On AMD Opteron 8370, Tesla C2060 GPU, WAES presents a maximum throughput of **71.8 Gbps on GPU memory mode.**

For most experiments, we varied the grid size of threads, which has considerable influence on the results. Once determined and set the size of the buffers we can adopt the grid activation that present the best performance in a particular hardware. For the comparison chart with earlier work, we use the highest flow obtained for each buffer size. In figure 5 we present a detailed study of the efficiency of the WAES algorithm on GTX260 GPU. In fact, best performance is achieved when the number of blocks equals the number of multiprocessors (MP) and the number of threads in a block in excess of the number of MP cores. We call this kernel configuration a *persistent block model*, as the threads in a block alternate in MP warps but the thread block remain fixed in one MP. The bottom chart presents the bandwidth when WAES is applied only to buffers in GPU memory. To measure

the transfer bandwidth of GPU kernels to mapped CPU memory (MM pinned) we implemented two methods. One, called WAES+MMstore, makes the parallel encryption of blocks generating directly into CPU mapped memory. In the test called COPY kernel, we coded a parallel kernel to just copy a GPU buffer to a mapped CPU memory. With this experiment we have found that the memory access time is the performance limiting factor for the generation of encrypted buffers in pinned CPU memory. WAES throughput for large buffers approaches the performance of the parallel COPY kernel.

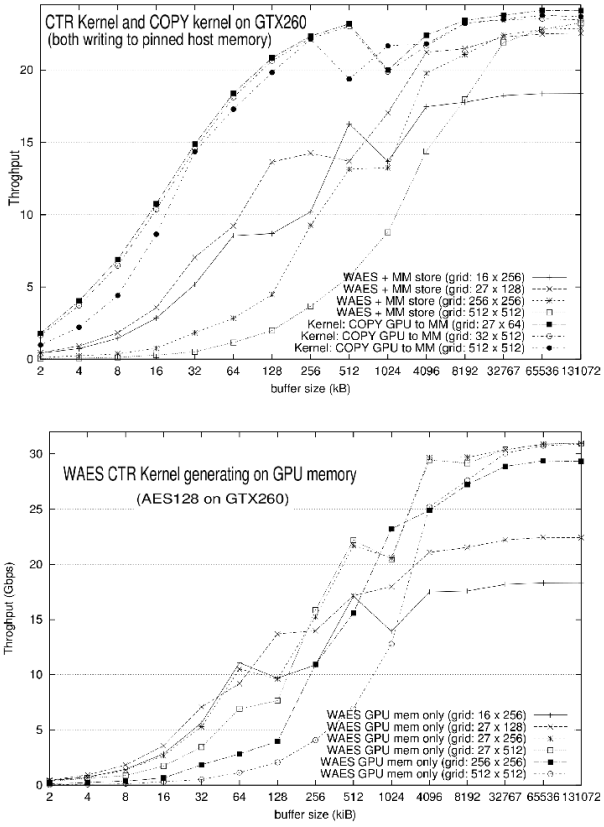

Figure 5: Performance tunning of WAES in GTX260 GPU.

6. Conclusion

In this work we have shown new techniques to efficiently map the AES CTR encryption algorithm for execution in parallel SIMT GPUs. Good performance has been achieved by algorithmic transformations to avoid inter-thread communications or synchronization. We have conducted extensive experimentation to tune for highest performance, which was achieved by mapping all algorithm state in registers and using shared memory for table lookups. In this configuration the *persistent blocks execution model* (defined in section 5, a.k.a *persistent threads*) is most appropriate.

WAES CTR kernel was included in WAESlib, a library that has good scalability features to work with multiple encryption contexts in GPUs, even considering the delays in transferring data. Currently we are using WAESlib to integrate WAES and other CTR based encryption algorithms in high performance applications and subsystems like secure MPI and encrypted file systems, where low latency and scalability are key characteristics.

References

M. BELLARE, A. DESAI, E. JOKIPII, AND P. ROGAWAY, 1997. *A concrete security treatment of symmetric encryption.* In Proceedings of the 38th Annual Symposium on Foundations of Computer Science, FOCS '97, pages 394–, Washington, DC, USA. IEEE Computer Society.

D. J. BERNSTEIN AND P. SCHWABE, 2008. *New AES software speed records.* In Proceedings of the 9th International Conference on Cryptology in India: Progress in Cryptology, INDOCRYPT '08, pages 322–336, Berlin, Heidelberg. Springer-Verlag.

K. GUPTA, J. STUART, AND J. D. OWENS, 2012. *A Study of Persistent Threads Style GPU Programming for GPGPU Workloads*. In Proceedings of Innovative Parallel Computing (InPar '12), May 2012.

O. HARRISON AND J. WALDRON, 2008. *Practical symmetric key cryptography on modern graphics hardware.* In In Proc. USENIX Security Symposium, pages 195–209.

K. IWAI, T. KUROKAWA, AND N. NISIKAWA, 2010. *AES encryption implementation on CUDA GPU and its analysis.* First International Conference on Networking and Computing, 0:209–214.

K. JANG, S. HAN, S. MOON, AND K. PARK, 2010. *Accelerating SSL with GPUs.* In ACM SIGCOMM'10, pages 437–438, August 2010.

M. KOUNAVIS, X. KANG, M. GREWAL, S. ESZENYI, 2010. *Encrypting the internet.* In SIGCOMM, pages 135–146. ACM, August 2010.

S. A. MANAVSKI, 2007. *CUDA compatible GPU as an efficient hardware accelerator for AES cryptography.* In Proc. IEEE International Conference on Signal Processing and Communication, pages 65–68, November 2007.

NIST, 2001. *Specification for the advanced encryption standard (AES).* Publication 197, NIST, November 2001.

NVIDIA-CORPORATION, 2010. *CUDA C programming guide.* version 3.2, September 2010.

D. A. OSVIK, J. W. BOS, D. STEFAN, AND D. CANRIGHT, 2010. *Fast software AES encryption.* In Proceedings of the 17th international conference on Fast software encryption, FSE'10, pages 75–93, Berlin, Heidelberg. Springer-Verlag.

X. J. RUAN, Q. YANG, M. I. ALGHAMDI, S. YIN, Z. Y. DING, J. XIE, J. LEWIS, AND X. QIN, 2010. *ES-MPICH2: A message passing interface with enhanced security.* 29th International Performance Computing and Communications Conference (IPCCC).

W. M. N. ZOLA AND L. C. E. BONA, 2012. P*arallel Speculative Encryption of Multiple AES Contexts on GPUs.* In Proceedings of Innovative Parallel Computing (InPar '12), May 2012.