Academy of Romanian Scientists
University of Oradea, Faculty of Electrical Engineering and Information Technology

Vol. 4, Nr. 2, October 2011

Journal of Computer Science and Control Systems

University of Oradea Publisher

ISSN 1844 - 6043

This volume includes papers in the folowing topics: Artificial intelligence and robotics, Real-time systems, Software engineering and software systems, Advanced control of electrical drives, Dependable computing, data security and cryptology, Computer networks, Modern control systems, Process control and task scheduling, Web design, Databases and data mining, Computer graphics and virtual reality, Image processing.

CONTENTS

ABDELMALEK Ibtissem[1], GOLÉA Noureddine[2] - [1]University of Batna, Algeria, [2]University of Oum El Bouaghi, Algeria
LMI-Based Design of Fuzzy Controller and Fuzzy Observer for Takagi-Sugeno Fuzzy Systems: New Non-Quadratic Stability Approach.......5

DAXIAO Wang, YINGJIAN Xu, WEI Zhang, LEIJUN Xiang - Shanghai Jiao Tong University, China
Controller Designing and Parameter Tuning in the First Order plus Large Delay Time System.......13

DRĂGHICIU Nicolae, POPA Sorin - University of Oradea, Romania
Computer Analysis of Doppler Signals 19

FILIP Andrei-Toader, HANGIU Radu-Petru, MARȚIȘ Claudia, BIRO Karoly-Agoston - Technical University of Cluj-Napoca, Romania
Design of an Interior Permanent-Magnet Synchronous Machine for an Integrated Starter-Alternator System Used on an Hybrid-Electric Vehicle 23

GIRIRAJKUMAR S.M.[1], KUMAR Atal. A.[2], ANANTHARAMAN N.[3] - [1]M.A.M. College of Engineering, India, [2]SASTRA University, India, [3]National Institute of Technology, India
Application of Simulated Annealing for Tuning of a PID Controller for a Real Time Industrial Process 29

HARLIŞCA Ciprian, SZABÓ Loránd - Technical University of Cluj-Napoca, Romania
Wavelet Analysis and Park's Vector Based Condition Monitoring of Induction Machines 35

MANG Erica, MANG Ioan, ANDRONIC Bogdan, POPESCU Constantin - University of Oradea, Romania
Theoretical Implementation of the Rijndael Algorithm Using GPU Processing.......39

MANG Ioan, MANG Erica, POPESCU Constantin - University of Oradea, Romania
VHDL Implementation of an Error Detection and Correction Module Based on Hamming Code 43

NEAMTU Iosif Mircea - "Lucian Blaga" University of Sibiu, Romania
Advanced Methods for Testing Security 47

NEAMTU Iosif Mircea - "Lucian Blaga" University of Sibiu, Romania
Software Tools to Detect Suspicious Files 51

NOVAC Ovidiu, NOVAC Mihaela, VARI-KAKAS Stefan, VLADU Ecaterina - University of Oradea, Romania
Aspects of Cache Memory Simulation using Programs Under Windows and UNIX Operating Systems.......55

NOVAC Ovidiu[1], SZTRIK Janos[2], VARI-KAKAS Stefan[1], KIM Che-Soong[3] - [1]University of Oradea, Romania, [2]University of Debrecen, Hungary, [3]Sangji University, South Korea
Reliability Increasing Method Using a SEC-DED Hsiao Code to Cache Memories, Implemented with FPGA Circuits.......59

SUKUMAR Senthilkumar, MT PIAH Abd Rahni - University Sains Malaysia
A Study on Parallel RKAM for Raster FCNN Simulation 63

ȚEPELEA Laviniu, GAVRILUȚ Ioan, GACSÁDI Alexandru - University of Oradea, Romania
A CNN Based Algorithm for Medical Images Correlation 69

ZERROUKI Nadjib[1], GOLÉA Noureddine[2], BENOUDJIT Nabil[1] - [1]University of Batna, Algeria, [2]University of Oum El Bouaghi, Algeria
Genetic Algorithm Based High Performance Control for Rigid Robot Manipulators 73

Theoretical Implementation of the Rijndael Algorithm Using GPU Processing

MANG Erica, MANG Ioan, ANDRONIC Bogdan, POPESCU Constantin

University of Oradea, Romania
Department of Computer Science, Faculty of Electrical Engineering and Information Technology,
410087 Oradea, Bihor, Romania, 1, Universitatii street,
Tel/Fax:+40 259/408412, Tel:+40 259/408104; +40 259/408204
E-Mail: imang@uoradea.ro, emang@uoradea.ro, ppopescu@uoradea.ro

Abstract ***– In 1997, the National Institute of Standards and Technology (NIST) initiated a process to select a symmetric-key encryption algorithm to be used to protect sensitive Federal Information. In 1999 six of the fifteen candidate algorithms where selected, and in November 2000 Rijndael was announced to be the winner. This paper presents the study of AES Rijndael implemented using modern Graphics Processing Units, highlighting the benefits of parallel implementation over the standard Central Processing Unit implementation. It describes the approach based on Nvidia CUDA platform. AES is the standard encryption model adopted in current software applications.***

Keywords: ***Cipher, Rijndael, Encryption, Decryption, ShiftRow Transformation, Cuda, GPU, Byte Substitution, MixColumn Transformation.***

I. INTRODUCTION

The three criteria taken into account in the design of Rijndael are the following:

- Resistance against all known attacks;
- Speed and code compactness on a wide range of platforms;
- Design simplicity.

In most ciphers the round transformation has the Feistel Structure. In this structure typically part of the bits of the intermediate State are simply transposed unchanged to another position. The round transformation of Rijndael does *not* have a Feistel structure. Instead, the round transformation it is composed of three distinct invertible uniform transformations, called *layers*. By "uniform", we mean that every bit of the State is treated in a similar way.

The specific choices for the different layers are for a large part based on the application of the Wide Trail Strategy, a design method to provide resistance against linear and differential cryptanalysis. In the Wide Trail Strategy, every layer has its own function:

The linear mixing layer: guarantees high diffusion over multiple rounds.

The non-linear layer: parallel application of S-boxes that have optimum worst-case nonlinearity properties.

The key addition layer: a simple EXOR of the Round Key to the intermediate State.

Before the first round, a key addition layer is applied. The motivation for this initial key addition is the following. Any layer after the last key addition in the cipher (or before the first in the context of known-plaintext attacks) can be simply peeled off without knowledge of the key and therefore does not contribute to the security of the cipher (e.g., the initial and final permutation in the DES). Initial or terminal key addition is applied in several designs, e.g., IDEA, SAFER and Blowfish.

In order to make the cipher and its inverse more similar in structure, the linear mixing layer of the last round is different from the mixing layer in the other rounds. It can be shown that this does not improve or reduce the security of the cipher in any way. This is similar to the absence of the swap operation in the last round of the DES.

II. MAIN DIFFERENCES BETWEEN CPU AND GPU

The CPU (central processing unit) has often been called the brains of the PC. But increasingly, that brain is being enhanced by another part of the PC – the GPU (graphics processing unit), which is its soul.

All PCs have chips that render the display images to monitors. But not all these chips are created equal. Intel's integrated graphics controller provides basic graphics that can display only productivity applications like Microsoft PowerPoint, low-resolution video and basic games [4].

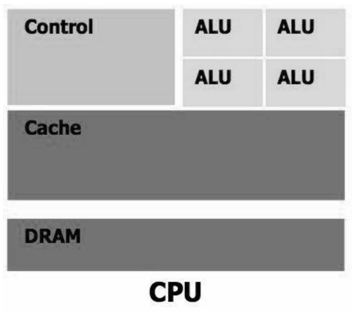

Figure 1. CPU architecture

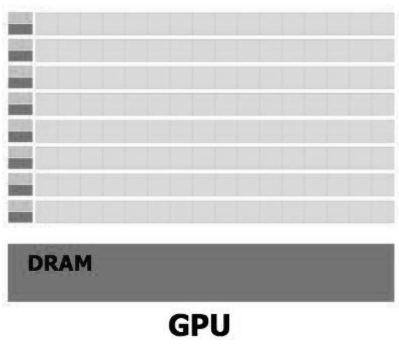

Figure 2. GPU architecture

The processor architecture differes considerably from that of the GPU because of the CPU main scope. The CPU has a purpoise of multiple tasks and multiple types of coputations. The GPU has a single purpoise of mathematica computation in order to display graphics.

III. SPECIFICATION

Rijndael is an iterated block cipher with variable block length and a variable key length. The bloc length and the key length can be independently specified to 128, 192 or 256 bits.

A. The State, the Cipher Key and the number of rounds

The different transformations operate on the intermediate result, called the State.

Definition: the intermediate cipher result is called the *State*. The State can be pictured as a rectangular array of bytes. This array has four rows, the number of columns is denoted Nb and is equal to the block length divided by 32.

The Cipher Key is similarly pictured as a rectangular array with four rows. The number of columns of the Cipher Key is denoted by Nk and is equal to the key length divided by 32. These representations are illustrated in Figure 3.

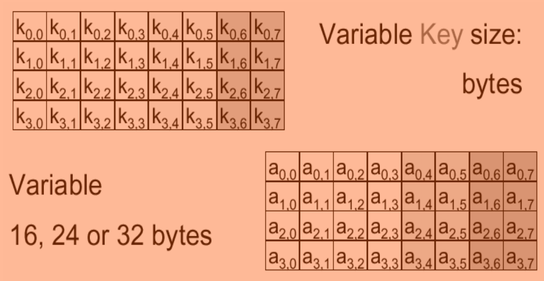

Figure 3. Example of State (with Nb = 6) and Cipher Key (with Nk = 4) layout.

In some instances, these blocks are also considered as one-dimensional arrays of 4-byte vectors, where each vector consists of the corresponding column in the rectangular array representation. These arrays hence have lengths of 4, 6 or 8 respectively and indices in the ranges 0..3, 0..5 or 0..7. 4-byte vectors will sometimes be referred to as words. Where it is necessary to specify the four individual bytes within a 4-byte vector or word the notation (a, b, c, d) will be used where a, b, c and d are the bytes at positions 0, 1, 2 and 3 respectively within the column, vector or word being considered.

The input and output used by Rijndael at its external interface are considered to be one-dimensional arrays of 8-bit bytes numbered upwards from 0 to the 4*Nb-l. These blocks hence have lengths of 16,24 or 32 bytes and array indices in the ranges 0..15, 0..23 or 0..31. The Cipher Key is considered to be a one-dimensional arrays of 8-bit bytes numbered upwards from 0 to the 4*Nk-l. These blocks hence have lengths of 16, 24 or 32 bytes and array indices in the ranges 0..15, 0..23 or 0..31. The cipher input bytes (the "plaintext" if the mode of use is ECB encryption) are mapped onto the state bytes in the order $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$, $a_{1,1}$, $a_{2,1}$, $a_{3,1}$, $a_{4,1}$ ... , and the bytes of the Cipher Key are mapped onto the array in the order $k_{0,0}$, $k_{1,0}$, $k_{2,0}$, $k_{3,0}$, $k_{0,1}$, $k_{1,1}$, $k_{2,1}$, $k_{3,1}$, $k_{4,1}$ ... At the end of the cipher operation, the cipher output is extracted from the state by taking the state bytes in the same order. Hence if the one-dimensional index of a byte within a block is *n* and the two dimensional index is *(i,j)*, we have:

$$i=n \bmod 4; \quad j=[n/4]; \quad n=i+4*j.$$

Moreover, the index *i* is also the byte number within a 4-byte vector or word and *j* is the index for the vector or word within the enclosing block. The number of rounds is denoted by Nr and depends on the values Nb and Nk. It is given in Table I.

TABLE I. Number of rounds (Nr) as a function of the block and key length

Nr	Nb = 4	Nb = 6	Nb = 8
Nk = 4	10	12	14
Nk = 6	12	12	14
Nk = 8	14	14	14

B. The round transformation

The round transformation is composed of four different transformations. In this notation the "functions" (Round, ByteSub, ShiftRow, ...) operate on arrays to which pointers (State, RoundKey) are provided.

We implemented a reduced version of the Rijndael algorithm with 6 round and 128 bits for cipher text and for cipher key.

It can be seen that the final round is equal to the round with the MixColumn step removed. We consider that the last round is similar with other rounds.

The component transformations are specified in the following subsections.

1) The ByteSub transformation

The ByteSub Transformation is non-linear byte substitution, operating on each of the State bytes independently. The substitution table (or S-box) is invertible and is constructed by the composition of two transformations:

1. First, taking the multiplicative inverse in GF(2^8). '00' is mapped onto itself.
2. Then, applying an affine (over F(2)) transformation.

2) *The ShiftRow transformation*

In ShiftRow, the rows of the State are cyclically shifted over different offsets. Row 0 is not shifted, row 1 is shifted over C1 bytes, row 2 over C2 bytes and row 3 over C3 bytes. The shift offsets Cl, C2 and C3 depend on the block length Nb.

3) The MixColumn transformation

In MixColumn, the columns of the state are considered as polynomials over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial c(x), given by:

$$c(x) = \text{'03'}\, x^3 + \text{'01'}\, x^2 + \text{'01'}\, x^1 + \text{'02'}.$$

4) *The Round Key addition*

In this operation, a Round Key is applied to the State by a simple bitwise EXOR. The Round Key is derived from the Cipher Key by means of the key schedule. The Round Key length is equal to the block length **Nb**.

IV. OVERWIEV ABOUT CUDA LANGUAGE OF PROGRAMATION

CUDA (originally an acronym for Compute Unified Device Architecture although this is no longer used) is a parallel computing architecture developed by NVIDIA. Simply put, CUDA is the computing engine in NVIDIA graphics processing units or GPUs, that is accessible to software developers through industry standard programming languages. Programmers use 'C for CUDA' (C with NVIDIA extensions), compiled through a PathScale Open64 C compiler, to code algorithms for execution on the GPU. CUDA architecture supports a range of computational interfaces including OpenCL and DirectX Compute . Third party wrappers are also available for Python, Fortran and Java.

The latest drivers all contain the necessary CUDA components. CUDA works with all NVIDIA GPUs from the G8X series onwards, including GeForce, Quadro and the Tesla line. NVIDIA states that programs developed for the GeForce 8 series will also work without modification on all future Nvidia video cards, due to binary compatibility. CUDA gives developers access to the native instruction set and memory of the parallel computational elements in CUDA GPUs. Using CUDA, the latest NVIDIA GPUs effectively become open architectures like CPUs. Unlike CPUs however, GPUs have a parallel "many-core" architecture, each core capable of running thousands of threads simultaneously - if an application is suited to this kind of an architecture, the GPU can offer large performance benefits.

In the computer gaming industry, in addition to graphics rendering, graphics cards are used in game physics calculations (physical effects like debris, smoke, fire, fluids), an example being PhysX and Bullet (software).

CUDA has also been used to accelerate non-graphical applications in computational biology, cryptography and other fields by an order of magnitude or more. An example of this is the BOINC distributed computing client.

CUDA provides both a low level API and a higher level API. The initial CUDA SDK was made public 15 February 2007. NVIDIA has released versions of the CUDA API for Microsoft Windows and Linux. Mac OS X was also added as a fully supported platform in version 2.0, which supersedes the beta released February 14, 2008.

CUDA has several advantages over traditional general purpose computation on GPUs (GPGPU) using graphics APIs.

Scattered reads – code can read from arbitrary addresses in memory.

Shared memory – CUDA exposes a fast shared memory region (16KB in size) that can be shared amongst threads. This can be used as a user-managed cache, enabling higher bandwidth than is possible using texture lookups.

Faster downloads and readbacks to and from the GPU.

Full support for integer and bitwise operations, including integer texture lookups.

Limitations

It uses a recursion-free, function-pointer-free subset of the C language, plus some simple extensions. However, a single process must run spread across multiple disjoint memory spaces, unlike other C language runtime environments.

Texture rendering is not supported.

For double precision there are no deviations from the IEEE 754 standard. In single precision, Denormals and signalling NaNs are not supported; only two IEEE rounding modes are supported (chop and round-to-nearest even), and those are specified on a per-instruction basis rather than in a control word (whether this is a limitation is arguable); and the precision of division/square root is slightly lower than single precision.

The bus bandwidth and latency between the CPU and the GPU may be a bottleneck.

Threads should be running in groups of at least 32 for best performance, with total number of threads numbering in the thousands. Branches in the program code do not impact performance significantly, provided that each of 32 threads takes the same execution path; the SIMD execution model becomes a significant limitation for any inherently divergent task (e.g., traversing a ray tracing acceleration data structure).

CUDA-enabled GPUs are only available from NVIDIA (GeForce 8 series and above, Quadro and Tesla).

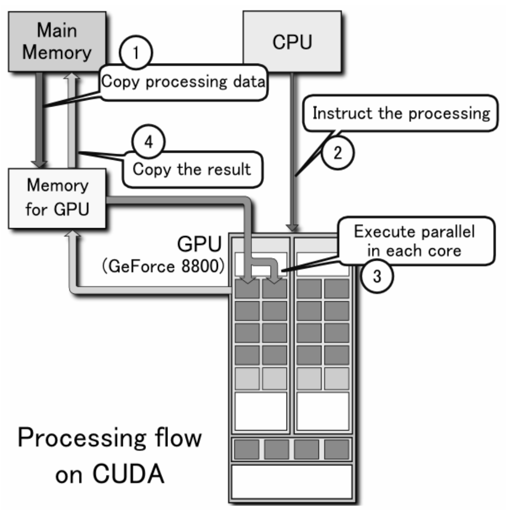

Figure 4. CUDA processing flow[5]

V. STUDY CASE

The tests were conducted using an AMD Athlon 300+, 2.2GHz, an NVIDIA GeForce GT240. As it can be seen, a comparison has been made between GPU and a CPU implementation. A peak throughput rate of 8.28 Gbit/s is achieved with an input size of 8MB. In that case the GPU is 19.60 times faster than the CPU.

For a input file of 2KB the GPU time was approximately 0.26 ms, the CPU time was approximately 1 ms. For a file of 512 KB GPU time was approximately 2.7ms versus 9ms for CPU, for files of 1 MB GPU time was 3.3 ms versus 19ms CPU, files of 4BM GPU time was 13.9ms and CPU time 74ms, for files of 8MB GPU time was 27.4ms and CPU time was 150ms.

VI. CONCLUSIONS

Encryption on GPU can improve speed of files of 8MB and larger up to 20 times.

- The current GPU's can provide good parallelism processing for intense threading programs, thus relieving the CPU.
- Current solutions of unified architectures can outperform CPU's in massive parallel programming.

Still there are efforts to introduce software that can rely solely on the GPU, the effort it greatly scaled because the GPU cannot act alone and retrieve the data without CPU aid, thus increasing the processing time in some cases by up to 50%, thus small programs do not actually present benefits by running on the GPU because of the data transfer time. If the data can be retrieve by bypassing the CPU then the processing time can be decreased significantly.

REFERENCES

[1] OpenGL Architecture Review Board, M. Woo, J. Neider, T. Davis, D. Shreiner, "The OpenGL Programming Guide: The Official guide to Learning OpenGL, Version 2", 5th edition. ISBN 0321335732, Addison-Wesley, New York, 2005
[2] General Purpose Computation Using Graphics Hardware, http://www.gpgpu.org.
[3] NVidia CUDA, http://developer.NVidia.com/object/CUDA.html.
[4] http://blogs.nvidia.com/2009/12/whats-the-difference-between-a-cpu-and-a-gpu/
[5] http://en.wikipedia.org/wiki/CUDA