# Analysis of Elliptic Curve Method & Block Wiedemann Algorithm: A Case Study of Big Integer Factorization on Multi-cores Platforms

PAS lab, CSIE, NTU

## I. Motivation

RSA is the most popular algorithm for public-key cryptography. RSA's security relies on the difficulty of big integer factorization on modern computers. In the past twenty years, there are lots of mechanisms to solve this problem. Most of them consume great computing power and need to be implemented on parallel computers.

Several years ago, the general trend in processor development has been shifted from high clockrate single processor to multi-cores. Many manufacturers joined the contest to develope multi-cores processor. Most of them design homogeneous multi-cores processor, i.e. all of the cores are the same. Examples are Intel core i7 and AMD PhenomII, etc. But there also exists some heterogenous multi-cores processors, e.g. STI's Cell processor.

Cell processor was developed by Sony, Toshiba and IBM, an alliance known as "STI" in 2005. One Cell processor is composed of nine heterogenuous cores. Each core can execute single instruction multiple data (SIMD) instruction independently. With cluster computers composed of Cell processors, three layer parallelism , including SIMD layer, heterogeneous multi-cores layer, and MPI layer, exist.

We hope to exploit Cell's three layer parallelisms to help the big integer factorization computation. And it can also be a case study of big integer factorization's performance on multi-cores platforms.

## II. General Number Field Sieve

Most encryption/decryption algorithms make use of one-way functions, which can be thought of as mappings that are difficult to invert. In RSA, the one-way function is multiplication of large prime integers, where "large" usually means over 1024 bits. The key is that while multiplication of such integers can be done nearly instantly, the inversion function of factoring back into primes is virtually impossible[1].

RSA's one way function's power depends on modern computers' inability to factorize big integers. In the past 20 years, many mathematicians developed lots of methods to solve this problem, like elliptic curve method invented by H. W. Lenstra in 1985. ECM is now the method of choice to find factors in the range $10^{10}$ to $10^{60}$ of general numbers. But it's not applicable to numbers with hundreds of digits. Another method, called number field sieve (NFS), is the most efficient algorithm known for factoring integers larger than 100 digits. Kazumaro Aoki, Jens Franke, Thorsten Kleinjung, Arjen K. Lenstra, and Dag Arne Osvik reached a new factoring milestone by completing the first special number field sieve factorization of a number having more than 1024 bits, namely the Mersenne number $2^{1039} - 1$ in 2007[2] [3].

General number field sieve composes of several steps, including

1) Select a polynomial
2) Build the factor bases
3) Sieve
4) Process relations
5) Find dependences
6) Compute the final factorization

There are thress steps related to performance. They are polynomial selection, sieving, and finding dependences. The polynomial selection is related to the number field generation. A good polynomial will eliminate the sieving procedure and increase the possibility to find non-trivial solutions. Sieving step is the most time-consumed step. But it can be parallelized via distributing differnent sieving intervals on different nodes. Step to find dependences is a matrix step. This step's objective is to find some instances of the matrix's kernel space so that we can use those instances to generate solutions. (It is not guaranteed that the solution is non-trivial so that we often need to find many instances rather than just one)

Take the number RSA100 for instance, the ratio of each steps is as the following Table I. (Use GGNFS version 0.77 with single core of Intel Q6600 processor)

TABLE I
GGNFS COMPUTATION RATIO ON RSA100

| | Time consumes in hours | Ratio |
|---|---|---|
| polynomial selection | 0.33 | 7.0% |
| make factor bases | 0.001 | 0.0% |
| Sieve | 4.11 | 86.7% |
| Process relations | 0.08 | 1.7% |
| Find dependences | 0.14 | 3.0% |
| Compute the final factorization | 0.08 | 1.7% |

## III. ELLIPTIC CURVE METHOD

Elliptive Curve Method (abbreviated as ECM) was invented by H. W. Lenstra in 1985. During the past twenty years, there are lots of modifications in order to improve the algorithm or implement it efficiently. Until now, the largest factor found using ECM so far was in 2006. And the number is 67 digits in length[4]. Even though ECM can't factorize large number more 200 bits efficiently, it can be a sub-step of General Nubmer Field Sieve's second step–the sieving step so that it's still worth to improve ECM's performance.

## IV. BLOCK WIEDEMANN ALGORITHM

Wiedemann Algorithm is invented by D. Wiedemann in 1986. Don Coppersmith modified it so that it can exploit computer architecture's characteristics and called Block Wiedemann Algorithm. Block Wiedemann algorithm is used to find kernel space of a squared sparse matrix over Gallois field. For General Number Field Sieving's usage, the Gallois field is $GF(2)$.

In addition to Block Wiedemann Algorithm, there is another well-known algorithm called Block Lanczos Algorithm. Block Lanczos Algorithm is built in GGNFS, a GPL implentation of the General Number Field Sieve.[5] But since the input for Block Lanczos Algorithm must be a symmetric matrix, it costs inneglect computing time during the conversion of original matrix while the dimension of matrix is large. In this thesis, I choose Block Wiedemann Algorithm to be implemented. Block Wiedemann Algorithm is as the following Algorithm 1

**Input**: A $N \times N$ matrix over $\mathbf{GF}(2)$ and the shift parameter $\Delta$, a non-negative integer

**Step 1.** Pick up random matrices $X, Y$. Let $Z = AY$;

**Step 2.** Let $\delta_l = \lceil N/m \rceil$ and $\delta_r = \lceil N/n \rceil$. Compute $H_i = XA^iZ, i = 0, ..., \delta_l + \delta_r + \Delta - 1$;

Then, solutions $w$ such that $Aw = 0$ are constructed from generating vector polynomials for that sequence;

**Step 3.** Compute a generating vector polynomial $g(\lambda) = g_0 + g_1\lambda + ... + g_d\lambda^d \in \mathbf{GF}(2)^n[\lambda]$ of degree at most $\delta_r$ for the sequence $\left\{XA^iZ\right\}_i$ i.e. such that: $XA^iZg_0 + XA^{i+1}Zg_1 + ... + XA^{i+d}Zg_d = 0$for $0 \leq i \leq \delta_l + \Delta - 1$;

**Step 4.** Let $g_l$ be the first non-zero vector coefficient of $g(\lambda)$. Compute $\widehat{w} = Yg_l + AYg_{l+1} + ... + A^{d-1}Yg_d$;

**Step 5.** Compute the first integer $\iota$ such that $A^\iota\widehat{w} = 0$;

**Output**: If $\iota \geq 1$, then $w = A^{\iota-1}\widehat{w}$ else $w = 0$

**Algorithm 1**: Block Wiedemann Algorithm[6]

## V. STI CELL BROADBAND ENGINE

STI's Cell processor is constructed from one power processing element (PPE) and eight synergistic processing elements (SPE) with a high-bandwidth element interconnection bus (EIB). All of nine cores work at 3.2GHz clockrate.

EIB is a ring-based topology, composed of four rings as shown in Figure 1. EIB is not only used for connecting PPE and SPE, but also connecting memory and IO controllers. Each ring's bandwidth is 16 bytes per cycle, and it supports multiple simultaneous transfers per ring. The peak bandwidth of EIB is 96 bytes/cycle.

Each SPE is composed of a synergistic processing unit (SPU) and a memory flow controller (MFC) (See Figure **??**). SPU, acting like a RISC processor, is used for general computation. Each SPU has SIMD unit and 256KB local storage without cache design. On SPE, data must be moved to the local storage via direct memory access (DMA) before computation. DMA ability is not provided by SPU, but the MFC. Each SPE has a DMA engine so that it can access main memory or other SPEs' local storage during its computation. With DMA's help, programmers can easily overlap between computation and communication on Cell processor.

In 2008, IBM introduces a variant of Cell processors called PowerXCell8i. The main feature of PowerXCell8i is the improvement of double-precision floating-point performance on the SPEs[8]. Compared with previous Cell processor, PowerXCell8i supports fully-pipelined double-precision floating-point operations. The double precision peak throughput of a PowerXCell8i processor is 102 GFLOPS using 8 SPEs. (14 GFLOPS with previous Cell processor) IBM's Blade Server QS22 has PowerXCell8i inside.

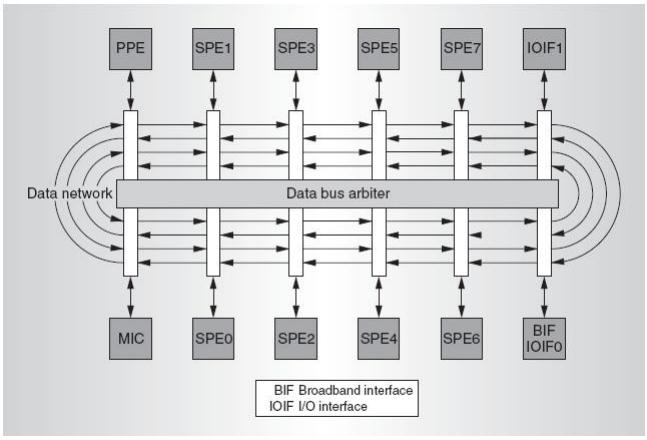


Fig. 1. Cell Element Interconnect Bus (EIB)[7]

## VI. Project Objective

This project provides an analysis of Block Wiedemann Algorithm's and Elliptic Curve Method's performance on the STI Cell Broadband Engine processor.

Block Wiedemann Algorithm is used for solving kernel space of a squared matrix. Find kernle space of a large matrix is an essential step of General Number Field Sieve, a popular algorithm to do big integer factorization.

Elliptic Curve Method is a factorization method. As now, the largest prime factor found using the ECM had 67 digits. Even though it is not enough to attack RSA algorithm, it can be a candidate to help General Number Field Sieve do the sieve step.

With above algorithms' performance result, we can understand STI Cell Broadband Engine processor's capability on big integer factorization.

## References

[1] M. E. Briggs, "An introduction to the general number field sieve," Master's thesis, Virginia Polytechnic Institute and State University, 1998.
[2] K. Aoki, J. Franke, T. Kleinjung, A. Lenstra, and D. Osvik, "A kilobit special number field sieve factorization," in *Advances in Cryptology-ASIACRYPT 2007*, Springer.
[3] D. J. Bernstein, T.-R. Chen, C.-M. Cheng, T. Lange, and B.-Y. Yang, "Ecm on graphics cards." Cryptology ePrint Archive, Report 2008/480, 2008. http://eprint.iacr.org/.
[4] "50 largest factors found by ecm." http://www.loria.fr/~zimmerma/records/top50.html/.
[5] "Ggnfs." http://www.math.ttu.edu/~cmonico/software/ggnfs/.
[6] G. Villard, "A study of coppersmith's block wiedemann algorithm using matrix polynomials," Tech. Rep. Research Report 975, LMC-IMAG, 1997.
[7] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessor communication network: Built for speed," *Micro, IEEE*, vol. 26, pp. 10–23, May-June 2006.
[8] "Powerxcell 8i processor product brief," 2008. http://www-03.ibm.com/technology/resources/technology_cell_pdf_PowerXCell_PB_7May2008_pub.pdf.