IOSR Journal of Computer Science (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727
PP 64-68
www.iosrjournals.org

Implementation of Iterative Algorithm In Parallel Processing And Distributed Computing For Crypt Data.

Nasim kothiwale[1], Trupti patil[2], Snehal Madum[3],Wasim Mujavar[4]
[1,2,3]*(Student of Computer Engineering, Annasaheb dange College of Engineering, Ashta)*
[4] *(Student of Computer Engineering, Dhole Patil College of Engineering, Pune)*

ABSTRACT: *A parallel computer (or multiple processor system) is a collection of communicating processing elements (processors) that cooperate to solve large computational problems fast by dividing such problems into parallel tasks, exploiting Thread-Level Parallelism (TLP). In this paper we mainly focus on inImplementation of Iterative algorithm in parallel processing and distributed computing for crypt data. We propose a parallel processing crypto-processor for Elliptic Curve Cryptography (ECC) to speed up EC point multiplication. The processor consists of a controller that dynamically checks instruction-level parallelism (ILP) and multiple sets of modular arithmetic logic units accelerating modular operations. A case study of HW design with the proposed architecture shows that EC point multiplication over GF(p) and GF(2m) can be improved by a factor of 1.6 compared to thecase of using single processing element.*

Keywords *-Multiple modular arithmetic logic units (MALUs), Public-Key Cryptography (PKC),*

I. INTRODUCTION

The implementation of a fast Public-Key Cryptography (PKC) is a challenge in embedded systems. Among PKCs, the best well-known and most widely used PKCs are RSA and ECC. In embedded systems, ECC is regarded more suitable than RSA because ECC operates with higher performance, lower power consumption, and smaller area of hardware. Nevertheless, the performance is much slower than private-key cryptography such as AES. A lot of work has been reported on speeding up the performance of ECC.

The work can be classified by the following optimization levels: First of all, a mathematical optimization of ECC has been investigated in order to reduce the number of modular multiplications in a point multiplication. Secondly, a high speed modular multiplier has been researched for both hardware and software implementations. Among the proposed algorithms, Montgomery's algorithm is considered as one of the fastest algorithms especially in terms of hardware. Thirdly, an architecture-level improvement can be considered. Our interest in this paper lies in this level.

The originality of our design is that an EC point multiplication can still be accelerated by using multiple modular arithmetic logic units (MALUs) in parallel. Some previous work reported parallel use of two modular arithmetic units for accelerating the EC point multiplication. In that work, point doubling and point addition are reformulated so that they can be executed by two processing elements. On the contrary, our proposed architecture is flexible in the following sense: 1) it can deal with two or more MALUs. 2) parallelism can be found on the fly by dynamically checking the instructions. 3) therefore, we can deal with any type of algorithm of point multiplication.

Another research interest is to clarify the appropriate number of MALUs for ECC. The remainder of this paper is as follows. Sect. 2 gives a survey of relevant previous work for ECC implementations. In Sect. 3, our proposed system architecture is described. In Sect. 4 the architecture of the MALU is explained. The proposed MALU supports the finite field operation over GF(p) and GF(2m). Sect 5 explains how to check the ILP in our design. Sect. 6 shows the performance evaluation result of example cases of 160/256-bit ECC over GF(p) and GF(2m).

II. Previous Work

Here we look at two main research areas: rendering and simulation with parallel processing.

1. RENDERING LARGE CROWDS.

The number of virtual people in real-time applications has increased significantly since the late 1990s, owing to enormous improvements in graphics hardware, performance increasing graphics algorithms, and solution-oriented rendering techniques. Although conservative frustum- and occlusion-culling techniques and a low level of detail (LOD) help decrease a rendering system's load, they can't by themselves achieve interactive frame rates in massive crowd simulation applications. Image-based rendering techniques, which are a life jacket for crowded virtual environments, simply represent virtual characters with fewer polygons, mostly quads, instead of high-polygon 3D models. By using this image-based approach and Nvidia's 64-Mbyte GeForce GTS2 card, Franco Tecchia and his colleagues visualized a village of 10,000 people at approximately 20 frames per second.

2. PARALLEL COMPUTING AND VIRTUAL CROWDS.

The research we just described used simple models to address behavior- and navigation-related issues. However, the demand for realism involves not only lifelike graphics but also artificial intelligence, smooth navigation, and physical modeling. Lifelike real-time virtual environments with massive crowds require more processing power than a commodity PC can provide. Parallel computing might help meet the requirements of computing-intensive opera-massivetions in crowd simulations. The approaches we report in this section don't benefit from basic load-minimizing techniques such as frustum and occlusion culling or LOD. They also handle every agent the same way, independent of visibility or distance.

II. RELATED WORK

The idea of a unified multiplier for PKC was first introduced by Savas¸, Tenca, and Koc¸. The most relevant published work is the one of Satoh and Takano . They present a dual field multiplier with the best performance in both types of fields so-far. The throughput of an EC point ultiplication is maximized by use of a Montgomery multiplier and an onthe-fly redundant binary converter.

The biggest advantage of their design is scalability in operand size and also flexibility between speed and hardware area. The same idea is the basis of the work of Großsch¨adl in. The bit-serial multiplier which is introduced is performing multiplications in both types of fields. Batina, et. al., reported parallel use of two modular arithmetic units for accelerating the EC point multiplication . In general, there are two possible methods to find a parallelism; one is by recompilation of SW(or a controlling logic in an ASIC), another is by HW that can check the parallelism on the fly. The later method is known as a super-scalar architecture that is often used in current high-end RISC CPUs.

III. SYSTEM ARCHITECTURE

The proposed parallel computing ECC cryptosystem is composed of the main controller, several ALUs and RAM which is shared between the MALUs, *i.e.*, this RAM is the so-called single-port shared RAM. The configuration of MALUs and RAM is assumed flexible by setting interconnections among them. The block diagram for the system is illustrated in Fig. 1.

The main controller has three in- and outputs, one of themis a signal which tells the controller to stop sending instructions when the instruction buffer is full. A 32-bit output is used to send instructions and a 32-bit input/output passes data back and forward between the controller and the datapath. A dedicated controller is chosen instead of a normal CPU because of two reasons. Firstly, instructions can be sent at a constant timing interval. Secondly, it is more compact and faster, it is possible to send one instruction per cycle.

The datapath is made with a Harvard architecture, it has a separate data bus and instruction bus. The data transfer between the main controller and the MALUs is controlled by a Data Bus Controller (DBC). This path is only activated when an initial point and the parameters of an elliptic curve are sent to the RAM, or when the result is retrieved. During a point multiplication, the controller keeps on sending instructions to the instruction decoder; they are stored in an Instruction Queue Buffer (IQB) via the Instruction Bus Controller (IBC).

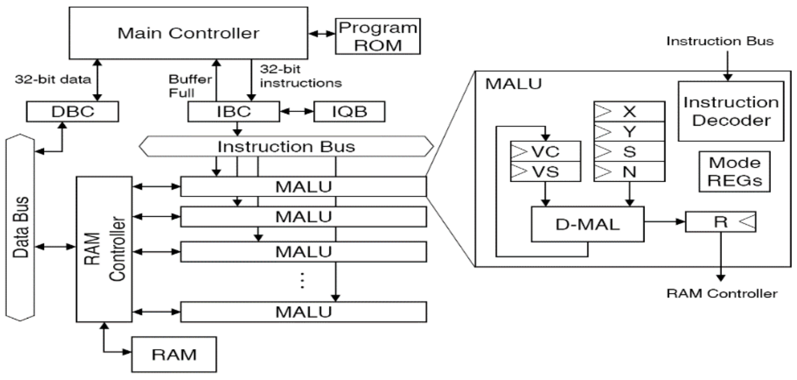

Fig1. System architecture of parallel processing processor.

The IQB has two main roles. First, the IBC checks if there is an instruction-level parallelism (ILP) by checking the data dependency in the IQB and forwards them to the different MALU(s). More information about the ILP can be found in Sect. 5, for the MALUs see Sect. 4. Another purpose of the IQB is buffering the difference of the speed of issuing instructions and the processing speed. The program ROM stores the instructions which are sent by the main controller. To be able to store all instructions for an ECC point multiplication of both GF(p) and GF(2m) (which can be chosen by setting the Mode register), only 596 bytes are necessary.

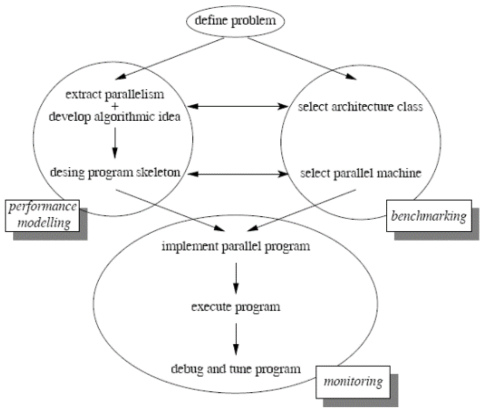

Fig 2 Performance Engineering in Parallel Program Development

The importance of incorporating performance modeling in software engineering activities has already been recognized in the literature and some results have already been presented on workload characterization for performance engineering In this work we will explain our ideas for a performance oriented software development process and discuss the impact on workload modeling Ferscha and Haring have proposed a performance oriented software development methodology for parallel processing systems where performance engineering activities range from performance prediction modeling in early design stages to monitoring measurements in the implementation and performance tuning phase.

IV. ITERATIVE ALGORITHM

ILP is checked for all instructions as long as two or more instructions are buffered in the IQB. Here, we introduce our strategy to find ILP. A MALU*N* instruction has three source operands and outputs the result to the RAM, *i.e.*, MALU*N* deals with four types of addresses, *X*, *Y*, *S*, and *R* that are expressed as follows:

MALUN :$R = X, Y, S$

Field Type	1MALU	2MALU	3MALU	4MALU
160-BIT GF(P)	165,326	111,046	103,594	103,594
256-BIT GF(P)	399,785	269,886	251,826	251,826
GF 2^{163}	135,689	86,277	83,901	83,901
GF 2^{257}	326,688	210,672	205,110	205,110

Table 1. Cycle counts [*cycle*] of EC point multiplication executed in different number of MALUs.

With out-of-order execution, the following dependencies are possible between two instructions, MALUi N and MALUj N (i and j are labels indicating issued order; $i < j$).

1 Read-After-Write (RAW) Dependency in order execution: $Ri = Xj$,$Ri = Y j$, or $Ri = Sj$. If the preceding result of $Ri$is necessary for following instructions, the instruction cannot be issued until the preceding instruction finishes.

2 RAW Dependency in out-of-order execution: $Rj= Xi$, $Rj = Y i$, or $Rj = Si$. In this case, Instruction MALUj N cannot be issued until the instruction MALUi N finishes. The proposed architecture does not need to check Write- After-Read andWrite-After-Write dependencies although they should be checked in a general super-scalar machine. This is because MALUN is a fixed-length instruction. Suppose we deal with D sets of instructions to search ILP, the number of conditions to check become $3(D - 1)2$. The hardware complexity for ILP expands with a large D.

V. PERFORMANCE EVALUATION

The system performance is evaluated with GEZEL which can make fast cycle-accurate simulations. The detailed hardware configuration is set as follows:

- Number of MALUs (#MALU) = 1 to 4
- MALU Configuration: $k = 160/256$, $d = 4$
- Depth of ILP exploration: $D = 4$

We use projective coordinates in EC point multiplication. Three sets of points, P, $2P$, and $3P$ are pre-computed and used for the left-to-right binary algorithm [10], *i.e.*, one point addition is always executed after two point doublings are executed. A modular inversion which is necessary for a coordinate conversion is executed with Fermat's Little Theorem. In our simulation, all necessary computations for EC point multiplication are included. The performance of a 160/256-bit EC point multiplication (ECC-160p / 256p and ECC-163b /257b) for different number of MALUs is summarized in Table 1. Considering allocation of two or three MALUs in the system, the performance improvement by a factor of 1.5 ˜ 1.6 is observed compared to the case of using one MALU. Any more speed-ups is not observed even when increasing the number of MALUs to more than three.

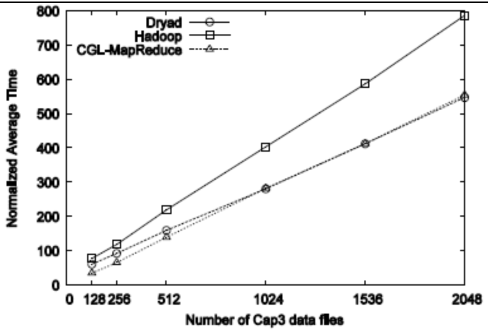

Figure 2. **Performance of the Cap3 application**

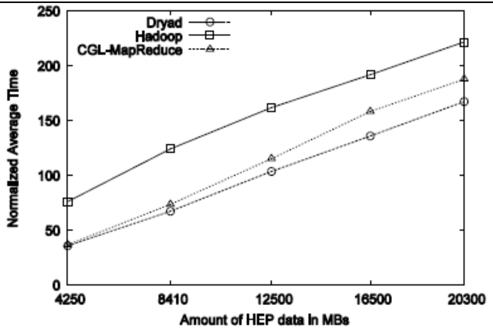

Figure 3. **Performance of HEP data analysis applications**

VI. IMPLEMENTATION RESULTS

We implemented iterative algorithm with two designs on a FPGA with different *k*. In case of *k* = 256, we allocated two MALUs because of resource limitation of our FPGA board. As shown in Table 2, our FPGA implementation shows an equivalent or better performance than other previous work for GF(*p*). However, regarding GF(2_m), the results of Satoh and Takano [6] and Orlando and Paar [12] are faster than our result for GF(2_m). This is obviously due to the faster clock frequency in case of the design of [6]. We note that the operation counts of their results (98 *Kcycle* and 230 *Kcycle* for GF(2_{160}) and GF(2_{256}), respectively) are more than ours. Regarding the design of [12], the architecture is intensively dedicated to GF(2_m). In this term, future work will deal with implementing a fast dual-field ECC which runs at the same clock frequency in both fields.

VII. CONCLUSION

We introduced parallel processing hardware architecture for crypt data by implement iterative algorithm . The proposed system is flexible in the MALU configuration and the number of processing elements. In conclusion, the performance of an EC point multiplication can be improved by using additional processing elements and an IQB that buffers the instructions and explores ILP dynamically. We could find ILP up to three instructions in our ECC program. The prototype implementation shows a significantly fast performance for both types of fields.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, ”A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,” Comm. ACM, vol. 21, pp. 120-126, 2011.
[2] N. Koblitz, ”Elliptic Curve Cryptosystems,” Math. Computation, vol. 48, pp. 203-209, 2010.
[3] V. S. Miller, ”Use of Elliptic Curve in Cryptography,” Advances in Cryptology: Proceedings of CRYPTO’85. Lecture Note in Computer Science, Springer-Verlag vol. 218, pp. 417-426, 1999.
[4] P. L. Montgomery, ”Modular multiplication without trial division,” Mathematics of Computation, vol. 44, pp. 519-21, 1985.
[5] E. Savas¸, A. F. Tenca and C¸ . K. Koc¸, ”A Scalable and Unified Multiplier Architecture for Finite Fields GF(p) and GF(2m),” Cryptographic Hardware and Embedded Systems: Proceedings of CHES’00. Lecture Note in Computer Science, Springer- erlag, vol. 1965, pp. 281-296, 2000.
[6] A. Satoh and K. Takano, ”A Scalable Dual-Field Elliptic Curve Cryptographic Processor,” IEEE Trans. Computers, vol. 52, pp. 449-460, 2003.