ISSN 2313-1527 (PRINT)
ISSN 2313-1535 (ONLINE)

IJEIE

International Journal of Electronics and Information Engineering

Vol. 2, No. 2 (June 2015)

INTERNATIONAL JOURNAL OF ELECTRONICS & INFORMATION ENGINEERING

PUBLISHING OFFICE

Min-Shiang Hwang
Department of Computer Science & Information Engineering, Asia University, Taichung 41354, Taiwan, R.O.C.
Email: mshwang@asia.edu.tw

International Journal of Electronics and Information Engineering is published both in traditional paper form (ISSN 2313-1527) and in Internet (ISSN 2313-1535) at http://ijeie.jalaxy.com.tw

PUBLISHER: Candy C. H. Lin

23-75, P.O. Box, Taichung, Taiwan 40199, R.O.C.

International Journal of Electronics and Information Engineering

Vol. 2, No. 2 (June 2015)

A NoC-based Sparse Matrix Multiplication System

Shao-Hua Chen[1,2], Yi-Sheng Lin[1], and Ming-Bo Lin[1]
(Corresponding author: Shao-Hua Chen)

Department of Electronic Engineering & National Taiwan University of Science and Technology[1]
No. 43, Section 4, Keelung Road, Taipei (106), Taiwan
Department of Electrical Engineering & Hwa Hsia Institute of Technology[2]
No.111, Gongzhuan Rd., Zhonghe Dist., New Taipei City (235), Taiwan
(Email: ritachen@cc.hwh.edu.tw)
(Received Mar. 12, 2014; revised and accepted Apr. 30, 2014)

Abstract

Sparse matrix multiplication (SMM) is widely used in many vital scientific and engineering computations, such as least-squares problems, eigenvalue problems, partial differential equations, and image reconstruction. However, it is a very time-consuming process and the irregular structure of sparse matrices usually causes general-purpose processors to perform poorly and to suffer cache misses severely. In this paper, we develop an SMM system based on network-on-a-chip (NoC) technology to parallelize the needed computations. To facilitate the load balancing and efficiency of packet distribution on the proposed SMM system, a method of mapping and partitioning a large matrix onto the system is also proposed. In addition, the proposed SMM system is fully parameterizable so that it has the maximum flexibility to meet the hardware resource available at hand. The proposed SMM system has been verified with a variety of network sizes, including 2×2, 2×4, 4×4, 4×8, and 8×8, on a Xilinx Virtex 5 device (XC5VLX110T) operating at 100 MHz. A number of random and real-application matrices are used to evaluate the performance of the proposed SMM system. In addition, the effects of network sizes, matrix sizes, and sparsity on the system performance are considered. The results show that the proposed SMM system can achieve up to 40× and 2× speedup over MicroBlaze and Intel processors, respectively. The proposed SMM system is also realized with a TSMC 0.18 μm cell library. The core area of the 4×4 system is 1,986.5 μm × 1,985.4 μm, equivalent to 259,026 gates. The average power consumption is 417 mW at the operating frequency of 166 MHz.

Keywords: Load balancing, mesh structure, NoC, parallel architecture, sparse matrix multiplication (SMM), SoC, wormhole router

1 Introduction

According to Moore's Law, the number of transistors that can be integrated on a chip would double every 18–24 months [11]. This makes it possible that the designer could put together a large number of IP blocks in a design. These IP blocks can be CPU cores, GPU cores, I/O controllers, and so on. However, the shrinking feature size of the deep-submicron technology makes interconnect delay and power consumption become the dominant factors of modern systems [5].

Most system-on-a-chip (SoC) designers adopt point-to-point interconnection or an on-chip bus to integrate components into a desired system nowadays. Although the point-to-point interconnection achieves high transmission performance, the cost of it grows exponentially with the number of IP blocks. In contrast, the bus-based architecture is widely used due to the advantages of lower complexity and small area. However, it suffers poor scalability, especially, when the system complexity is greatly increased. More precisely, the bus-based system has the following drawbacks which limit its widespread use in parallel computation. First, all components share the same bus to transmit data, which limits the transmission bandwidth. Second, the latency of the arbitration mechanism increases with the increasing number of connected components.

An efficient solution to the interconnection problem of a complex system is to embed the interconnection as a micronetwork called a *network-on-a-chip*, denoted as an NoC. A system architecture based on an NoC has much higher bandwidth than the architecture based on a bus since it has much larger number of concurrent transactions that can be carried out through the NoC. Besides, NoC achieves high scalability, reusability, and flexibility, and

hence, it is suitable for SoC applications. For these reasons, some research and commercial products of NoC have been announced recently, including Intel 80-core TeraFLOPS [17], TILE-Gx [15], TILE64 [3], and ARM AMBA Network Interconnect [1], showing a great development in this area.

In this paper, we demonstrate the design of an SMM system based on an NoC platform. SMM is widely used in many essential scientific and engineering computations, such as least-squares problems, eigenvalue problems, partial differential equations, and image reconstruction, even though it is a very time-consuming operation. Nevertheless, the irregular structure of sparse matrices usually causes general-purpose processors to perform poorly and suffer cache misses [16]. Fortunately, with the development of the NoC technology, we are able to accelerate multicore applications and maximize the performance of parallel computation. Therefore, this paper aims to design a suitable NoC platform for SMM.

The rest of this paper is organized as follows. In Section 2, we first give a brief review of related work. Following that, in Section 3, we introduce the basic concept of SMM. In Section 4, we describe the design of our NoC-based platform and the implementation of the SMM system. The experimental results are presented in Section 5. Finally, we conclude this paper in Section 6.

2 Related Work

Much research has been studied to improve the performance of SMM for general-purpose processors. In [10], the technique of register blocking is proposed to increase the memory locality of SMM. In [20], Zhang et al. build a specialized memory controller to decrease cache miss, but the performance of SMM is still closely related to the sparsity of matrices. With the improvement of parallel computation, Intel and AMD multicore processors are also used to improve the performance [18] of SMM, and some researchers even implement the SMM system on the CUDA platform [4]. In [12] and [21], a reconfigurable design and a tree-based design on FPGAs are also proposed, but the designs lack scalability severely. Even though a multiprocessor system-on-a-chip (MPSoC) architecture based on OpenRISC processors has been proposed [19], the poor packet distribution in the network deteriorates its performance significantly, and the area overhead of the system is very high.

In what follows, we first introduce how to effectively represent a sparse matrix with a compressed sparse row to store the nonzero elements in contiguous memory locations. Next, we describe the NoC design for the proposed SMM system, including routers, network interfaces, and processing elements. Then, we present an algorithm of matrix mapping and partitioning to achieve the load balancing and the transmission efficiency of the network used in the proposed SMM system.

3 Overview of Sparse Matrix Multiplication

The basic SMM operation can be expressed as follows:

$$y = A \cdot x \tag{1}$$

where A is an $n \times n$ sparse matrix with n_z nonzero elements; x and y are dense vectors of $n \times 1$. We use a_{ij} to denote the elements of A. Suppose that the operation of $y = A \cdot x$ is processed by a general-purpose processor, each vector y is generated by the equation of $y_i = \sum_{j=0}^{n} (a_{ij} \times x_j)$. Let R_i represent the number of nonzero elements in row i. Then, the sequential SMM involves $\sum_{i=0}^{i<n} (R_i - 1)$ addition operations and $\sum_{i=0}^{i<n} (R_i)$ multiplication operations. From the above equation, we know that the processing time of general-purpose processors is proportional to nonzero elements of matrices. Thus, the sequential SMM will consume a large amount of processing time for scientific and engineering matrices with millions of nonzero elements. Moreover, because the irregularity of memory accesses causes large numbers of cache misses in the hierarchical memory system of modern computers, the performance of SMM is rather poor in general-purpose systems.

In addition, the traditional data structure for a matrix is a two-dimensional array. Each element a_{ij} in the array can be accessed by two indices i and j. To store an $m \times n$ matrix, an enough memory space is needed. To save memory space, the storage format of sparse matrices should be designed with care. To achieve this, a data structure, called the *compressed sparse row* (CSR) [2] format, is proposed to store nonzero elements in contiguous memory locations. Nowadays, the CSR format has become the most common storage format for sparse matrices. In the CSR format, there are three vectors: *val* for nonzero elements, *col* for the column indices of the nonzero elements, and *ptr* for storing the locations in the *val* vector that start a row.

As an illustration, consider the following 5×5 sparse matrix:

$$A = \begin{pmatrix} 0 & 0 & 5 & 3 & 0 \\ 2 & 0 & 0 & 0 & 4 \\ 0 & 7 & 0 & 1 & 0 \\ 6 & 0 & 9 & 0 & 7 \\ 0 & 0 & 3 & 2 & 0 \end{pmatrix} \quad (2)$$

The data structure of CSR can be represented by Table 1.

Table 1: The CSR representation of the 5×5 sparse matrix of Equation (2)

val	5	3	2	4	7	1	6	9	7	3	2
col	2	3	0	4	1	3	0	2	4	2	3
ptr	0	2	4	6	9						

By subtracting consecutive elements of the *ptr* vector, a new vector *len* can be generated. *len* represents the number of nonzeros elements in each row, and this vector can be used to determine whether the accumulation operation is finished in practice. Note that the last element in the *len* vector is calculated using $n_z - ptr(n-1)$. For our example, the *len* vector is

len	2	2	2	3	2

In this paper, the CSR format is adopted in the entire system for storing matrices.

A basic CSR-based implementation of $y = A \cdot x$ is shown in Algorithm 1. From this algorithm, we can observe that the performance is dominated by the irregular and indirect memory access of vector x.

Algorithm 1 SMM with the CSR scheme

```
1: for i = 0 to m − 1 do
2:     y0 ← 0;
3:     for k = ptr[i] to ptr[i + 1] − 1 do
4:         j ← col[k];
5:         y0 ← y0 + val[k] × x[j];
6:     end for
7:     y[i] ← y0;
8: end for
```

4 NoC-Based System Design

4.1 Architecture

Since the mesh network may achieve high scalability and flexibility as compared with other network topologies, such as the ring network and butterfly network [6], it has become the most popular topology in MPSoC applications [14]. For this reason, we employ it to construct our proposed SMM system, as shown in Figure 1. In the proposed SMM system, a router is associated with each processing element (PE) directly so that they can be easily designed as a module of the mesh network.

The proposed SMM system consists of three parts: routers, network interfaces (NIs), and processing elements (PEs). Routers are responsible for delivering the packets in the mesh network and hence the transmission performance of the proposed SMM system is highly dependent on the router design. The NI between routers and PEs acts as a bridge with a function to compose packets into data or to decompose data into packets. PEs are the computational components of the proposed SMM system, with each consisting of an ALU (arithmetic and logic unit) and a register block. Because of a large amount of addition and accumulation computations required by SMM, all PEs in the proposed SMM system are designed in a way such that matrix elements are processed in parallel in order to reduce the processing time. All of matrix elements are distributed by a mapping algorithm so that the load of the mesh network and hence the system is balanced. For large sparse matrices, they can be realized by a partitioning algorithm. In addition, the architecture of the proposed SMM system, can be adjusted to any $M \times N$ network as long as $M = 2, 4 \ldots, 2^k$ $(k \in \mathbb{N})$, and $N = 2, 4 \ldots, 2^k$ $(k \in \mathbb{N})$. Based on this, the effects of the number of PEs on the performance can be easily analyzed.

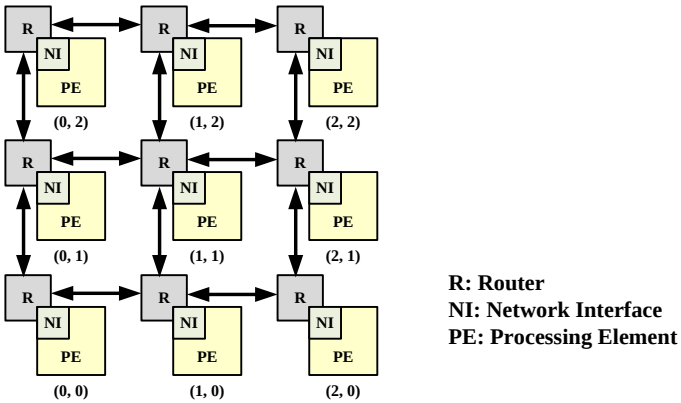

Figure 1: NoC-based architecture for SMM

4.2 Routers

Routers play an important role in the NoC infrastructure of the proposed SMM system. However, hardware overheads and power consumption are the two essential criteria needed to take into account in FPGA design. To shorten the latency of packet transmission and lower the cost of the proposed SMM system, the wormhole flow control is used to realize the router. Compared with other packet flow control methods, such as store-and-forward, virtual cut-through, and virtual channel [6, 13], wormhole flow control not only reduces the buffer size but also achieves high performance because of the inherent flit and pipeline technique associated with it. The architecture of each wormhole router in the proposed SMM system is shown in Figure 2, which has 5 input and output ports.

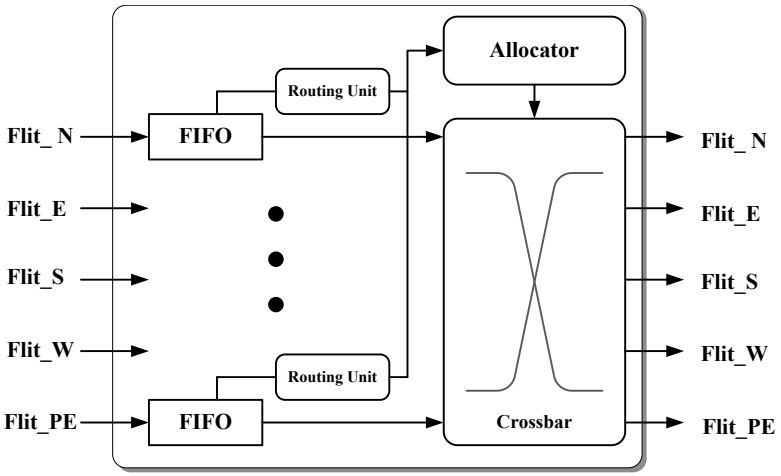

Figure 2: The architecture of each wormhole router

Each router can be divided into three parts: input units, allocators, and the crossbar. The input unit comprises a FIFO buffer and a routing unit. Once packets enter a router, they are stored in the FIFO buffer of the router and then delivered to the routing unit. The routing unit is implemented using the XY routing algorithm, because it is a deadlock-free, low-cost, low-latency, and minimal-path routing algorithm. The XY routing algorithm determines the path from a source node to a destination node. Although the non-adaptive XY routing may suffer from the network congestion in non-uniform traffic patterns [9], it cannot cause a big problem in our proposed SMM system because the packets are distributed uniformly. Besides, the turn model [8] can prove that the XY routing has no cyclic loops and hence it is deadlock-free. The details of the XY routing algorithm are shown in Algorithm 2. As an example, if a packet is sent from $(1, 1)$ to $(3, 3)$ in a 4×4 mesh, the selected path will be $(1, 1) \rightarrow (2, 1) \rightarrow (3, 1) \rightarrow (3, 2) \rightarrow (3, 3)$.

The arbiter is another crucial component in the router. An arbiter assigns a single resource to one of a group of requesters, while an allocator performs a match between a group of resources and a group of requesters. In this paper, several round-robin arbiters are used to build an allocator due to the low area overhead and strong fairness, as compared with other arbiters, such as fixed-priority arbiters, matrix arbiters, queueing arbiters [6], and so on.

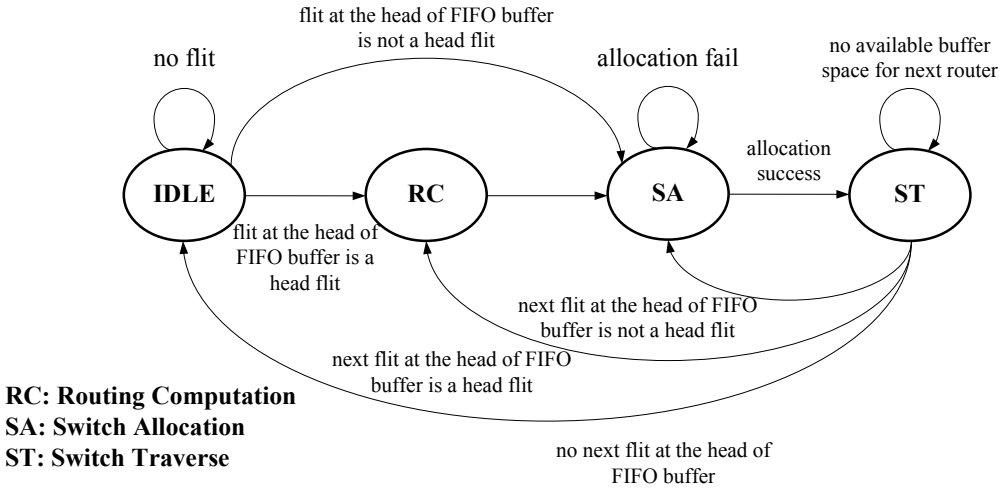

Figure 3: The state diagram of a 3-stage pipeline router

Algorithm 2 XY Routing Algorithm

```
(c_0, c_1): current node;
(d_0, d_1): destination node;
dest_port: output port of the router;
if d_0 − c_0 > 0 then
    dest_port ← EAST;
else if d_0 − c_0 < 0 then
    dest_port ← WEST;
else
    if d_1 − c_1 > 0 then
        dest_port ← NORTH;
    else if d_1 − c_1 < 0 then
        dest_port ← SOUTH;
    else
        dest_port ← PE;
    end if
end if
```

The round-robin arbiter's operation is that a request has the lowest priority on the next round of arbitration if it was just served. In the crossbar design, it is accomplished by multiple multiplexers and can receive the control signals from the allocator. The 5×5 crossbar is able to switch channels between routers and deliver packets.

To improve the performance of transmission, the router is implemented by a 3-stage pipeline, including routing computation (RC), switch allocation (SA), and switch traverse (ST). RC represents the routing computation of the input unit. SA represents the switch allocation of the allocator. Finally, ST represents the process that a packet traverses from the current router to the next-stage router. The state diagram of the 3-stage pipeline router is shown in Figure 3. Suppose that the 3-flit packet has no contention and only 5 clock cycles are needed to traverse the packet in the router.

4.3 Packet Formats

The packet of the XY routing algorithm, also a distributed routing algorithm, contains the information of a destination address and payload. However, the packet in our wormhole router is partitioned into a number of flits, with each consisting of a head flit, a number of body flits, and a tail flit as shown in Figure 4. The head flit contains the information of the destination address. The body and tail flits contain the payload part. Depending on the data width generated by the PE, the number of body flits may be different, generally, ranging from 2 to 4.

To let the PE function correctly, it is important to define the packet format. Let us take a 4×4 network as an example. For saving the space of FIFO buffers, each flit merely contains 10 bits. The two most significant bits of each flit are defined as the flit type, which determines the sequence of flits that follow. The head flit as shown in

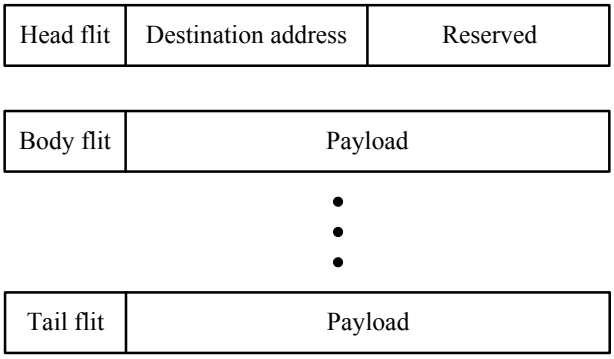

Figure 4: The packet format of the wormhole router

Table 2 contains the destination address, the register index of PEs, and the operation code (OPcode). The register index of PEs specifies the register to be used, and the OPcode defines the function of the packet to be performed by the PE. The detailed functions of packets will be discussed later.

Table 2: Definition of the head flit

Bit	Definition
9-8	flit type = 11
7-6	x-coordinate of destination address
5-4	y-coordinate of destination address
3-2	index of registers in PE
1-0	OPcode: INIT=01, MUL=10, ACC=11

Tables 3 and 4 define the payload of body and tail flits, according to the OPcode of the packet. The contents of both body and tail flits can be read from the CSR-based memory which stores matrices and vectors. Three OPcodes are defined to represent different functions: initialization (INIT), multiplication (MUL), and accumulation (ACC). The function of INIT is to initialize the registers in PEs for the operations of MUL and ACC. The other two functions, MUL and ACC, represent the operations of multiplication and accumulation, respectively.

Table 3: Definition of body flits

Bit	INIT	MUL	ACC
9-8	flit type = 10	flit type = 10	flit type = 10
7-0	value of vector	nonzero value of matrix	value of product

Table 4: Definition of tail flit

Bit	INIT	MUL	ACC
9-8	flit type = 01	flit type = 01	flit type = 01
7-0	*Len* value each row	row index of nonzero value	value of product

4.4 Network Interfaces

Routers perform packet exchanges whereas PEs are responsible for carrying out desired computations. The bridge between them is the network interface. In accordance with the direction of data flow, the NI can be further divided into the packet-to-data interface and the data-to-packet interface, as shown in Figure 5. In most cases, the data width in PEs is larger than the flit width in routers. As a consequence, there is a need to convert data into flits and vice versa. As shown in Figure 6, the conversion is very simple and can be accomplished by a shift register. The NI also checks the status of routers and PEs, such as the buffer space, before delivering data or packets. The design of NIs decouples the system complexity and achieves high reusability.

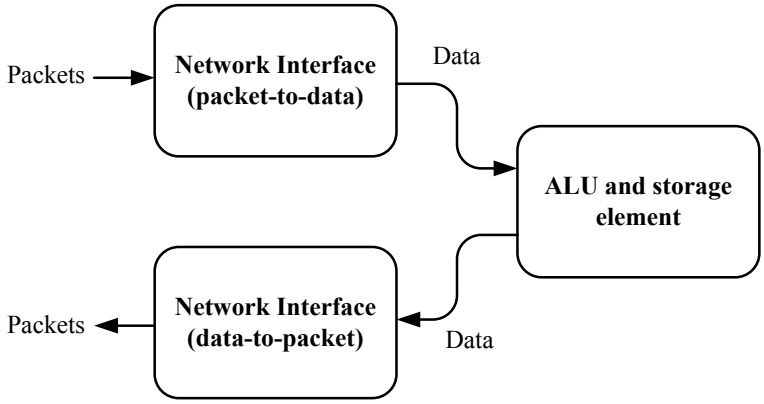

Figure 5: Network interfaces of packet-to-data and data-to-packet

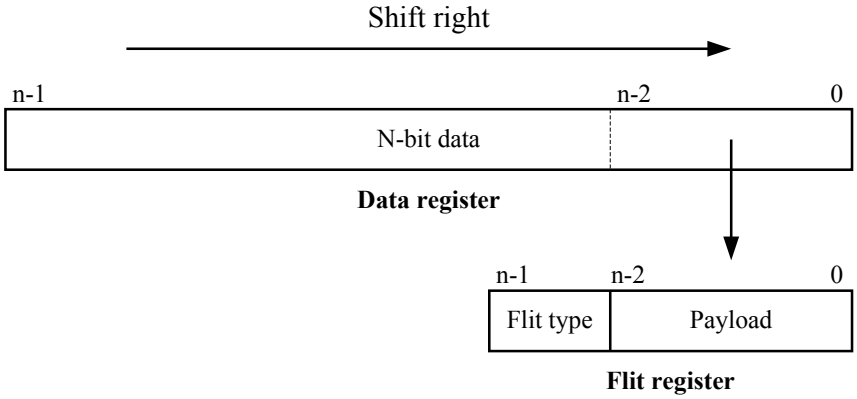

Figure 6: Conversion of data and flits

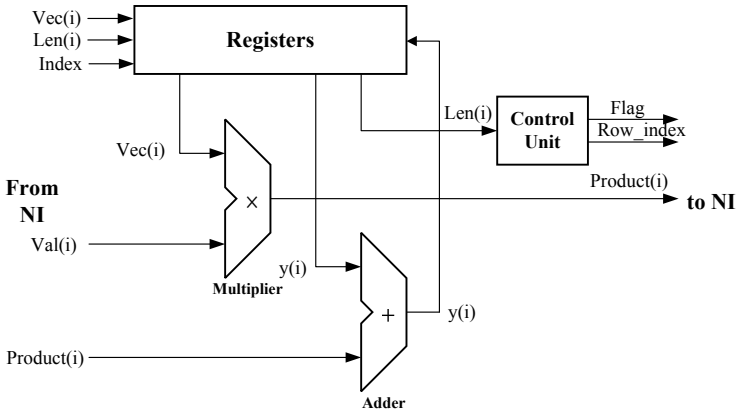

Figure 7: The processing element

4.5 Processing Elements

The architecture of each PE is shown in Figure 7. It consists of a multiplier, an adder, a number of registers, and a control unit. In our design, matrix A and vector x are stored initially in external memory based on the CSR format, and then are packeted and distributed to each node of the network by the mapping algorithm. After receiving the data generated by NIs, each PE is able to process the data in the corresponding mode.

Each PE supports three different modes: INIT, MUL, and ACC. In the INIT mode, PE(i) stores the elements of $vec(i)$ and $len(i)$ in its registers, where $vec(i)$ is the portion of the multiplicand vector (x) stored in the packet. In

our design, each PE in the network has the same number of registers for storing small parts of matrices and vectors. In the MUL mode, the multiplier reads a nonzero element *val* from the NI and finds the corresponding element of vector x in local registers using the column index of *val* as the address. It then multiplies these two elements together and delivers the result to the network. Besides, the width of each multiplier can be set to 16 bits, 32 bits, or 64 bits. In the ACC mode, PE(i) accumulates all nonzero elements of the matrix in a row after multiplication. Each time once an element is accumulated, the control unit decreases the value of *len* by 1. When *len* becomes zero, the final result of y is yielded and ready to be read. The detailed operations of each PE is illustrated in Algorithm 3.

Algorithm 3 Operations of Each Processing Element

```
1: read head flit;
2: i ← index of registers;
3: mode ← OPcode;
4: read body and tail flit;
5: if mode = INIT then
6:    store vec(i) to registers;
7:    store len(i) to registers;
8: else if mode = MUL then
9:    read val(i) and vec(i) from registers;
10:   read row_index;//row index of nonzero element val
11:   compute product(i) ← val(i) × vec(i);
12:   send a packet with product(i) to PE with the address of row_index;
13: else if mode = ACC then
14:   read product(i);
15:   compute y(i) ← y(i) + product(i);
16:   compute len(i) ← len(i) − 1;
17:   if len(i) = 0 then
18:      flag(i) ← 1;
19:   end if
20: end if
```

4.6 Partition and Mapping of Matrices

The total number of PEs determines the network size. However, a sparse matrix for scientific and engineering usually contains millions of nonzero elements. To tackle the multiplication of such matrices with a limited-size network, the matrix in question must be partitioned and its elements are distributed through the network using an efficient partitioning and mapping algorithm in order to maximize the performance of the system.

Since the registers in each PE can only store at most four $y(i)$ values due to the limitation of packet formats, the maximum matrix size can be directly processed by a fixed-size network is limited. For 2×2 to 8×8 networks, the maximum matrix sizes that can be processed directly are shown in Table 5. As a matrix with a size exceeds the maximum size of the underlying network, the matrix must be partitioned into blocks by the partitioning algorithm. These blocks are then processed one by one by the proposed SMM system.

Table 5: Maximum matrix sizes that can be processed in different network sizes

Network Size	2×2	2×4	4×4	4×8	8×8
Matrix Size	16×16	32×32	64×64	128×128	256×256

As described, in the proposed SMM system the computation of SMM is composed of three steps: initialization, multiplication, and accumulation. By loading the information of addresses, register indices, and the payload from external memory, packets can be forwarded in the network. The detailed operations of the mapping procedure of packets are illustrated in Algorithm 4. In the initialization step, the address and the index of registers are closely related to the *vec* index and the number of network nodes ($M \times N$). Since M and N are the integer multiples of 2, the values can be readily obtained from their binary representations. After storing the values of *vec* and *len* in the payload field, the packets are ready to be delivered.

In the multiplication and accumulation steps, the packets are generated in a similar way as in the initialization step except that the address and the index of registers are calculated by the column indices and row indices of *val*. The payload fields for the multiplication and accumulation steps contain both *val* and *product*. Once the values of *vec* are stored in the packets in the initialization step, these values can be reused to multiply and accumulate a large

Table 6: Synthesis results of Xilinx Virtex 5 (XC5VLX110T) with different network sizes

Network Size	Slice Logic Utilization	Used	Available	Utilization
2×2	Slice Registers	1,220	69,120	1%
	Slice LUTs	2,232	69,120	3%
	Occupied Slices	903	17,280	5%
2×4	Slice Registers	2,527	69,120	3%
	Slice LUTs	5,047	69,120	7%
	Occupied Slices	1,843	17,280	10%
4×4	Slice Registers	5,324	69,120	7%
	Slice LUTs	11,251	69,120	16%
	Occupied Slices	4,268	17,280	24%
4×8	Slice Registers	14,124	69,120	20%
	Slice LUTs	28,586	69,120	41%
	Occupied Slices	10,565	17,280	61%
8×8	Slice Registers	28,776	69,120	41%
	Slice LUTs	56,756	69,120	82%
	Occupied Slices	16,495	17,280	95%

number of nonzero elements of a matrix. Thus, the time needed in loading data from external memory is reduced dramatically.

Algorithm 4 Mapping of SMM

1: //Assume the network size is $M \times N$,
2: //and the matrix size is $k \times k$ with n_z nonzeros;
3: //Initialization;
4: **for** $i = 0$ to $i = k - 1$ **do**
5: $address \leftarrow (i \bmod (M \times N) \bmod M, i \bmod (M \times N)/M)$;
6: $register\_index \leftarrow i/(M \times N)$;
7: $payload \leftarrow vec(i)$ and $len(i)$;
8: send a packet with INIT function;
9: **end for**
10: //Multiplication;
11: **for** $i = 0$ to $n_z - 1$ **do**
12: $address \leftarrow (col(i) \bmod (M \times N) \bmod M, col(i) \bmod (M \times N)/M)$;
13: $register\_index \leftarrow col(i)/(M \times N)$;
14: $payload \leftarrow val(i)$ and row_index;
15: send a packet with MUL function;
16: **end for**
17: //Accumulation;
18: **for** each PE (in parallel) **do**
19: $address \leftarrow (row\_index \bmod (M \times N) \bmod M, row\_index \bmod (M \times N)/M)$;
20: $register\_index \leftarrow row\_index/(M \times N)$;
21: $payload \leftarrow product$;
22: send a packet with ACC function;
23: **end for**

The SMM of a large matrix can be accomplished by the partitioning algorithm. Suppose that if the maximum matrix size that can be processed by a particular network is $b \times b$, then an $n \times n$ matrix can be partitioned into $\lceil n/b \rceil$ blocks, as shown in Figure 8, with each block having a size of $b \times b$. Similarly, the $n \times 1$ vector is also partitioned into $\lceil n/b \rceil$ blocks, with each block having a size of $b \times 1$. Based on this partition, the multiplication is then performed on each matrix block with the corresponding vector block. Because most elements of the matrix are zeros, it is possible that some blocks contain no nonzero elements. In such a case, the computation time might be reduced significantly.

5 Experiments and Results

To verify the functionality and estimate the performance of the proposed SMM system, the proposed SMM system was designed and modeled in Verilog HDL, which is in turn implemented by a target device, Xilinx Virtex 5

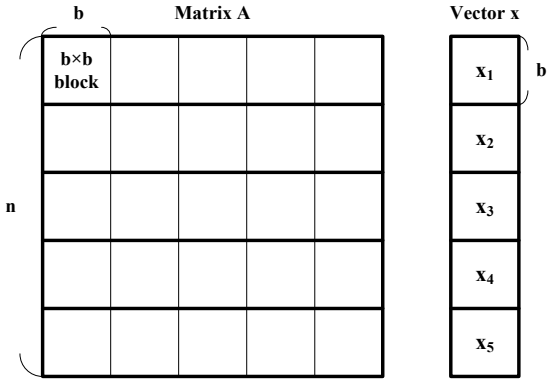

Figure 8: Partition of a large matrix

(XC5VLX110T, manufactured with 65-nm copper CMOS process technology), containing 69,120 slices. Within the FPGA device, an embedded memory module is used to store matrices and vectors. The system flow chart of the proposed SMM system is shown in Figure 9. At the beginning, the memory module keeps injecting packets with matrix and vector elements into the proposed SMM system if the buffer is not full yet. Meanwhile, the proposed SMM system starts performing the SMM. After the SMM process is done, the data in the registers of PE(i) can be verified and the performance of the proposed SMM system can be measured.

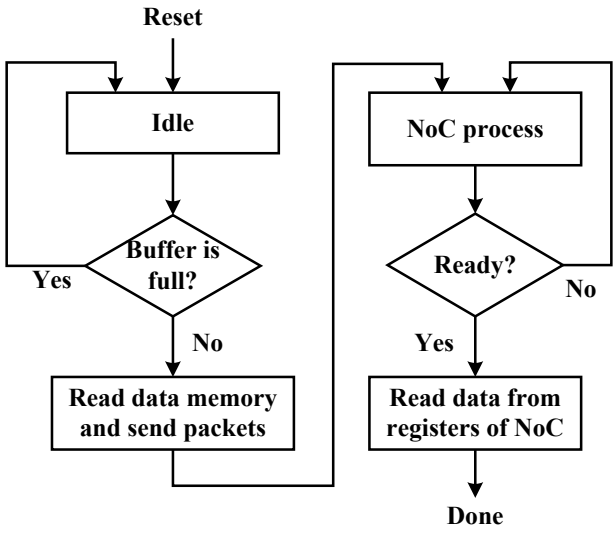

Figure 9: System flow chart of proposed SMM system

The synthesis results of Xilinx Virtex 5 (XC5VLX110T) with different network sizes are shown in Table 6. Because of the limitation of FPGA resources, the maximum network size of our design is limited to 8×8. As we can see, the slice utilization is 5% in the 2×2 network but increases to 95% in the 8×8 network. From the results, we can realize that the number of slices required in different network sizes grows in a factor linearly proportional to the number of network nodes approximately.

To estimate the performance of the proposed SMM system, two types of test profiles of sparse matrices are employed. The first type of test profiles is called random matrices that can be generated by the C++ standard rand() function. By changing the probability of the random function, different properties of random matrices can be created. The second type of test profiles called real-application matrices is available from the University of Florida Sparse Matrix Collection [7], including various engineering and scientific applications, such as 2D/3D problems, combinatorial problems, fluid mechanics, circuit simulations, and so on. The resulting performance of these matrices is more realistic and convincible since they usually contain a large number of nonzero elements.

In the FPGA implementation, our design runs at the frequency of 100 MHz. To measure the performance of the proposed SMM system, the runtime is used as the performance metric. Here, the runtime is defined as the time from

the injection of the first packet to the system till the completion of the computations of all the elements of y. In the experiment of random matrices, 5 matrices of different sizes and 5 matrices of different sparsity are employed. Note that the definition of sparsity is Sparsity = Nonzeros/Dimension^2. The resulting performance of random matrices are listed in Tables 7 and 8, and the corresponding line charts are shown in Figures 10 and 11, respectively.

Table 7: Runtime of random matrices with different sizes (μs)

Matrix size	Network size				
	2×2	2×4	4×4	4×8	8×8
20×20	2.17	1.05	0.97	0.89	1.01
40×40	7.45	3.89	2.15	1.99	1.62
60×60	13.80	6.49	3.58	2.92	2.64
80×80	24.40	12.09	7.03	4.61	4.52
100×100	40.23	19.57	10.22	7.19	6.80

Table 8: Runtime of random matrices with different sparsity (μs)

Matrix sparsity	Network size				
	2×2	2×4	4×4	4×8	8×8
20%	40.23	19.57	10.22	7.19	6.80
40%	60.78	31.05	15.75	12.21	11.80
60%	80.00	40.52	21.79	17.02	16.73
80%	100.08	51.76	27.20	22.66	21.93
100%	116.62	65.20	33.10	28.04	26.71

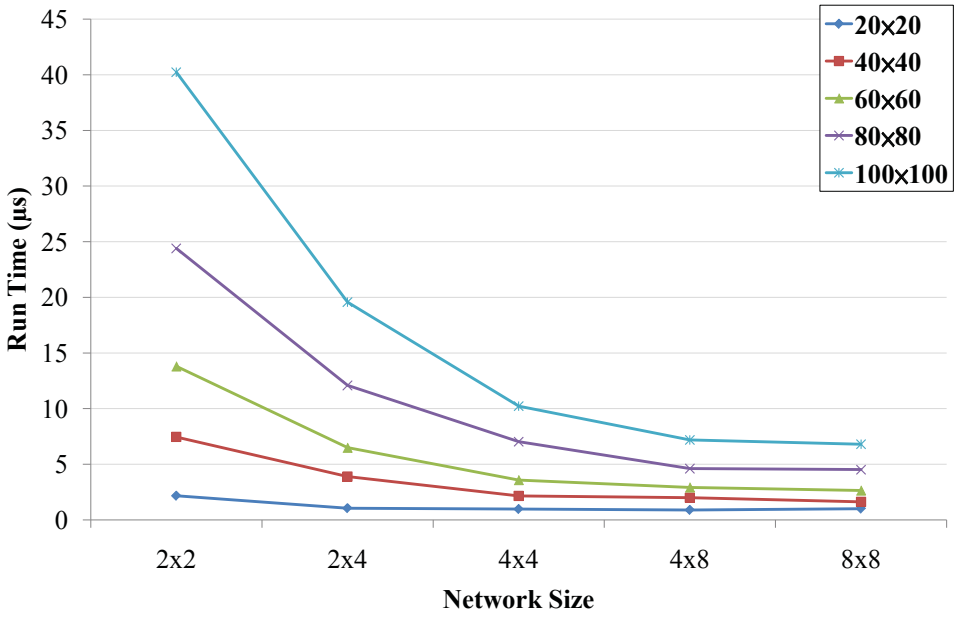

Figure 10: Performance of random matrices with different sizes

In the experiment of real-application matrices, several sparse matrices are chosen from the University of Florida Sparse Matrix Collection, as shown in Figure 9, along with the detailed specifications. The performance of real-application matrices is shown in Table 10, and the corresponding line charts are shown in Figure 12. Although the simulation results of real-application matrices are similar to random matrices, the performance improvement of real-application matrices is more significant, especially when the network size is large.

In addition to analyzing the impact of network sizes and sparsity on the system performance, we also measure the speedup of our proposed SMM system over general-purpose processors. To compare the performance with the

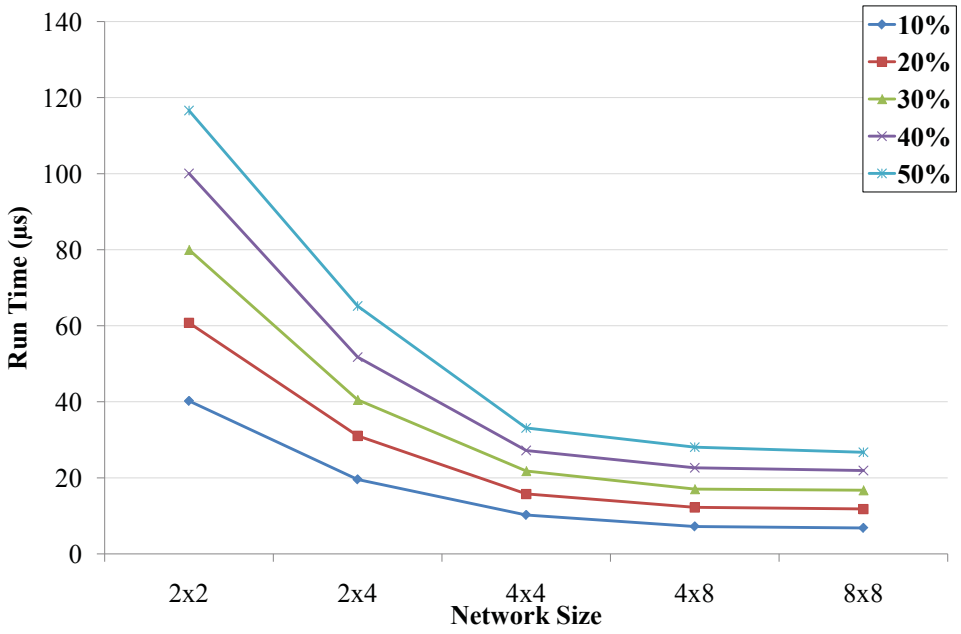

Figure 11: Performance of random matrices with different sparsity

Table 9: Detailed specification of real-application matrices

Matrix Name	Dimension	Nonzeros	Type	Sparsity
rdb5000	5,000	29,600	computational fluid dynamics problem	0.12%
sherman2	1,080	23,094	computational fluid dynamics problem	1.98%
Trefethen_700	700	12,654	combinatorial problem	2.58%
mcfe	765	24,382	2D/3D problem	4.17%
wang1	2,903	19,093	semiconductor device problem sequence	0.23%

Table 10: Runtime of real-application matrices (μs)

Name	Network size				
	2×2	2×4	4×4	4×8	8×8
rdb5000	1877.4	1226.85	448.7	317.03	273.54
sherman2	853.27	446.27	246.6	193.22	148.94
Trefethen	213.47	103.57	61.74	55.26	47.32
mcfe	454.59	252.28	207.03	162.13	141.58
wang1	881.62	512.19	279.14	215.09	166.46

same FPGA device, the MicroBlaze processor is used. The MicroBlaze is a soft RISC-based processor core designed for Xilinx FPGAs, which can be configured by Xilinx EDK. In our implementation, several peripheral devices are also added to the MicroBlaze processor, including timers, the debug module, and the UART module. The resource utilization of slice registers, slice LUTs, and occupied slices of the MicroBlaze processor is 21%, 21%, and 42% respectively. To measure the performance, the C++ code of SMM is compiled and programmed into the Xilinx Virtex 5 FPGA device, and the cycle-accurate runtime results of the MicroBlaze processor can be observed on the monitor through the RS-232 cable. The definition of speedup in our experiment is as follow:

$$\text{Speedup} = \frac{T_{CPU}}{T_{NoC}} \tag{3}$$

where T_{CPU} and T_{NoC} are the runtimes of SMM on the MicroBlaze processor and the proposed SMM system, respectively.

The speedup of different network sizes over the MicroBlaze processor in random matrices is shown in Figures 13 and 14 at the same operating frequency. Compared with the MicroBlaze processor, our proposed system shows

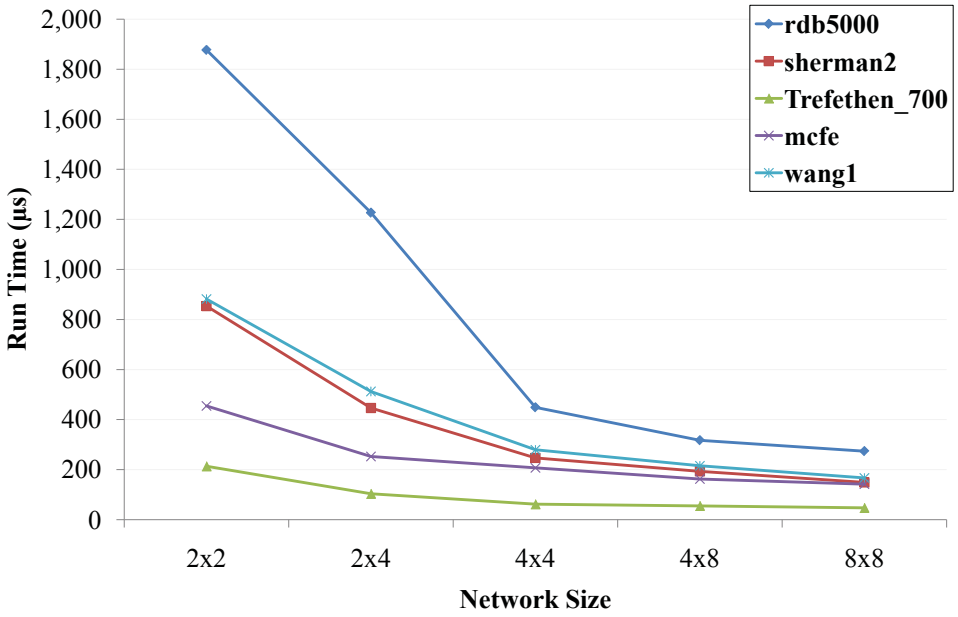

Figure 12: Performance of real-application matrices

a great performance improvement of 40x speedup in maximum. Due to the limitation of the MicroBlaze program memory, only random matrices are simulated.

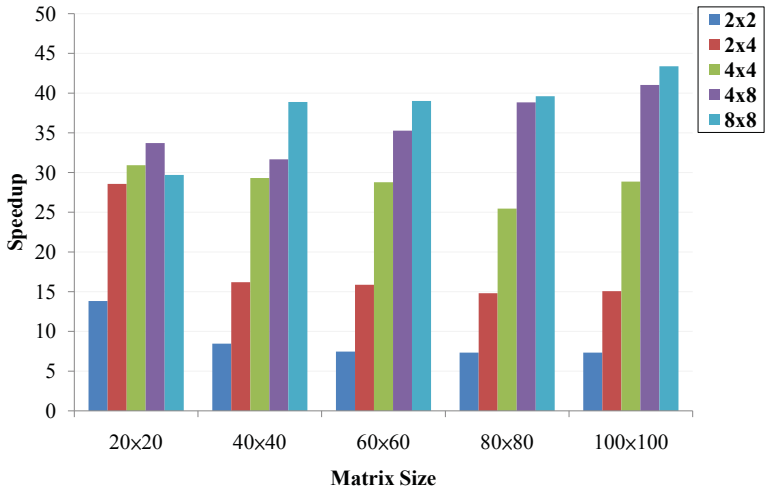

Figure 13: Speedup of NoC over MicroBlaze processor (1)

We also compare our design with an Intel Q6600 processor (with quad cores and an 8-MB cache) with 2-GB RAM operating at 2.4 GHz. The C++ program is compiled using Microsoft Visual Studio 2005. The speedup of different network sizes over the Intel processor in random matrices is shown in Figures 15, 16, and 17, respectively. The results show that our design can achieve an average of 2x speedup in maximum over the Intel processor, although the performance of our design under 4×4 networks is worse than that of the Intel processor. Nevertheless, this means that a great speedup can be achieved by a high-degree parallel computation system even though the operating frequency (100 MHz) of our proposed system is much lower than that (2.4 GHz) of the Intel processor. From the experiment, we can also realize that the performance improvement of real-application matrices in large networks is better than random matrices, because numerous PEs of large networks help maximize parallelism when processing large matrices with many nonzero elements.

After verifying the pre-synthesis codes on the FPGA device, we also synthesized our design based on a TSMC

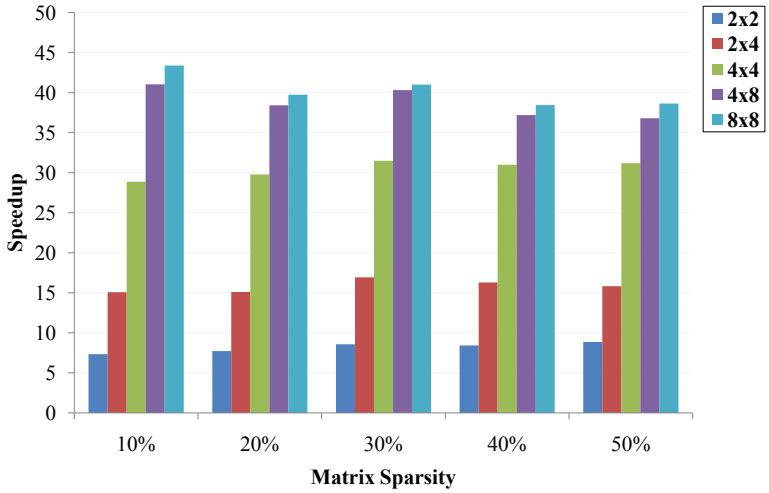

Figure 14: Speedup of NoC over MicroBlaze processor (2)

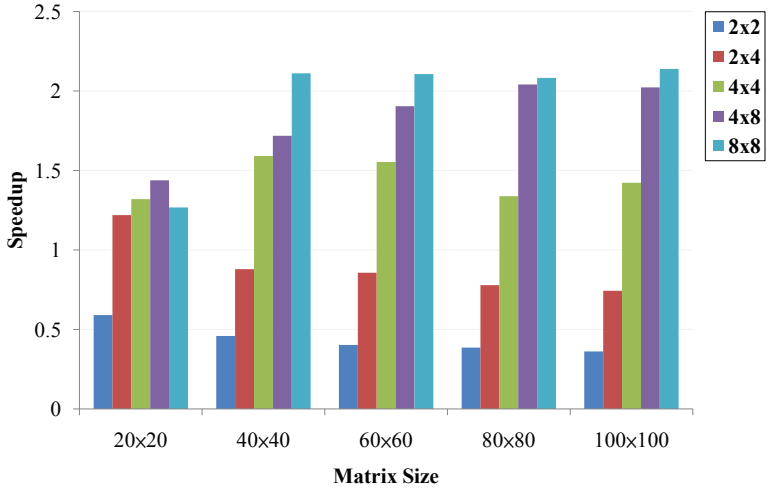

Figure 15: Speedup of NoC over Intel processor (1)

0.18 μm cell library by Synopsys Design Compiler and DFT Compiler. The gate count, maximum latency, power consumption, and test coverage after synthesis are shown in Table 11. Besides, we also accomplished the physical layout of the design by Synopsys IC Compiler as shown in Figure 18. The core area of the 4×4 network is 1986.5×1985.4 μm^2, and the power consumption is 480.55 mW at the frequency of 166 MHz.

Table 11: Synthesis results of Design Compiler and DFT Compiler

Network Size	2×2	2×4	4×4	4×8	8×8
Max. laterncy (ns)	5.4	5.39	5.4	5.4	5.4
Gate Count	72,581	134,380	259,026	506,143	1028,790
Power (mW)	91.07	189.83	417.40	842.64	1,741.30
Test Coverage	99.93%	99.94%	99.94%	99.94%	99.95%

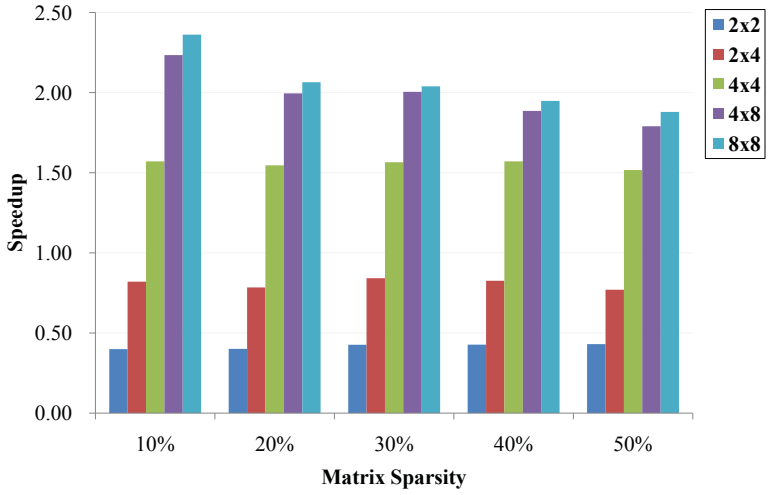

Figure 16: Speedup of NoC over Intel processor (2)

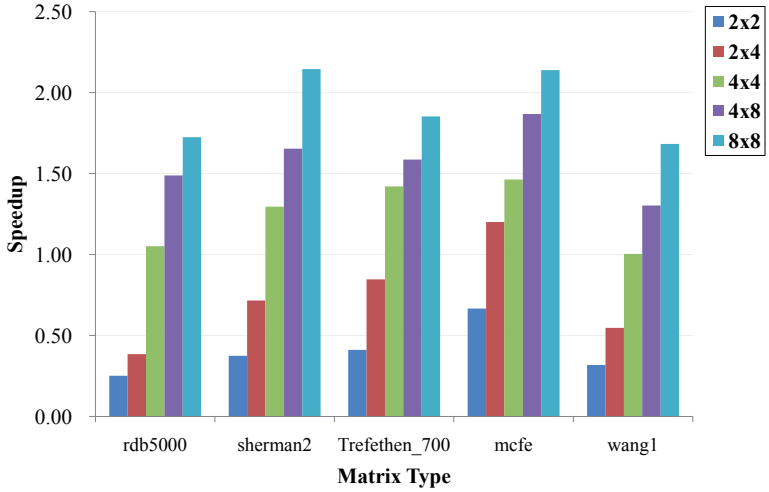

Figure 17: Speedup of NoC over Intel processor (3)

6 Conclusions and Future Work

In this paper, we presented the design of an NoC-based system for SMM to improve both the scalability and flexibility of parallel computation. The 2-D-mesh-based system consists of routers, network interfaces, and processing elements. In order to lower the area overhead and increase the transmission efficiency of the proposed SMM system, the wormhole router using the XY routing algorithm is introduced in the system. Besides, to achieve load balancing and effective packet distribution, the matrix data in the network are distributed by a partitioning and mapping algorithm.

From the experiment results, our design can achieve 40× speedup and 2× speedup in maximum over the MicroBlaze processor and the Intel Q6600 processor, respectively. Moreover, the proposed NoC-based architecture supports high-degree reusability, which allows us to construct complex SoC application systems. In the near future, we plan to design many other application systems, such as fast Fourier transform, multimedia encoders/decoders, and encryption/decryption algorithms.

Acknowledgment

This work was supported in part by the grants of NSC 100-2221-E-011-062 and NSC 101-2221-E-011-099 from National Science Council (NSC), Taiwan. In addition, the authors would like to thank National Chip Implementation

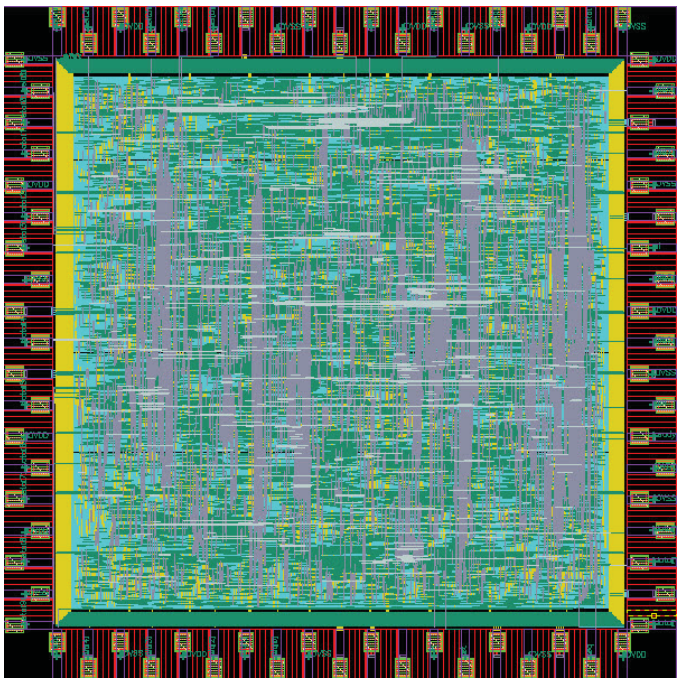

Figure 18: Physical layout of the 4×4 network SMM system

Center, Hsinchu, Taiwan, for their kind assistance, support, and chip fabrication.

References

[1] ARM, "CoreLink network interconnect for AMBA AXI." http://www.arm.com/products/system-ip/interconnect/axi/index.php.

[2] Z. Bai, J. Demmel, and J. Dongarra *et al.*, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Society for Industrial and Applied Mathematics, 2000.

[3] S. Bell et al., "TILE64 processor: a 64-core SoC with mesh interconnect," *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC 2008)*, pp. 88–598, Feb. 2008.

[4] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," *NVIDIA Technical Report NVR-2008-004,* NVIDIA Corporation, Dec. 2008.

[5] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[6] W. Dally and B. Towles, *Principles and Practices of Interconnections Networks*. Morgan Kaufmann Publishers, 2005.

[7] T. Davis and Y. Hu, "The university of florida sparse matrix collection", 2014. (`http://www.cise.ufl.edu/research/sparse/matrices`)

[8] C. Glass and L. Ni, "The turn model for adaptive routing," *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 278–287, May 1992.

[9] J. Hu and R. Marculescu, "DyAD - smart routing for networks-on-chip," *Proceedings of the 41st Annual Conference on Design Automation*, pp. 260–263, Jul. 2004.

[10] E. J. Im, K. Yelick, and R. Vuduc, "SPARSITY: optimization framework for sparse matrix kernels," *International Journal of High Performance Computing Applications*, vol. 18, no. 2, pp. 135–158, Feb. 2004.

[11] G. Moore, "Cramming more components onto integrated circuits," *Electronics,* Vol. 38, No. 8, pp. 114-117, April 1965.

[12] G. Morris and V. Prasanna, "Sparse matrix computations on reconfigurable hardware," *Computer*, vol. 40, no.3, pp. 58–64, Mar. 2007.

[13] C. Nicopoulos, D. Park, and J. Kim *et al.*, "ViChaR: a dynamic virtual channel regulator for network-on-chip routers," *Proceedings of Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 333–346, Dec. 2006.

[14] E. Salminen, A. Kulmala, and T. D. Hmlinen, "Survey of network-on-chip proposals." http://www.ocpip.org, Mar. 2008.

[15] Tilera, "TILE-GX precessor family." http://www.tilera.com/products/processors.
[16] S. Toledo, "Improving the memory system performance of sparse matrix-vector multiplication," *IBM Journal of Research and Development*, vol. 41, no. 11, pp. 711–725, Nov. 1997.
[17] S. Vangal et al., "An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS," *IEEE Journal of Solid State Circuits*, vol. 43, no.1, pp. 29–41, Jan. 2008.
[18] S. Williams et al., "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, pp. 1–12, Mar. 2007.
[19] J. Yang, C. Chun, and N. Bagherzadeh *et al.*, "Load balancing for data-parallel applications on network-on-chip enabled multi-processor platform," *Proceedings of the 19th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 439–446, Feb. 2011.
[20] L. Zhang et al., "The impulse memory controller," *IEEE Transactions on Computers*, vol. 50, no. 11, pp. 1117–1132, Nov. 2001.
[21] L. Zhuo and V. Prasanna, "Sparse matrix-vector multiplication on FPGAs," *Proceedings of the 13th International Symposium on Field-Programmable Gate Arrays (FPGA '05)*, pp. 63–74, Feb. 2005.

Shao-Hua Chen received the B.Sc. degree in electrical engineering from National Central University, Jhongli, Taiwan, and the M.S. degree in computer engineering from Syracuse University, Syracuse, NY. She is currently working toward the Ph.D. degree in electronic engineering at National Taiwan University of Science and Technology, Taipei, Taiwan. From 1995 to 1999, she was with Texas Instruments-Acer Inc., where she developed dynamic random access memory chips in CMOS technology. Since 2000 she has been with the Department of Electrical Engineering, Hwa Hsia Institute of Technology, New Taipei City, Taiwan, where she currently serves as a lecturer. Her main research interest is in the development of methods for precise time interval measurements implemented in application-specific integrated circuit technology and integrated in the field-programmable gate array technology.

Yi-Sheng Lin received the B.Sc degree in electronic engineering from Chang Gung University and the M.Sc degree in electronic engineering from National Taiwan University of Science and Technology in 2012. His major research interests is from algorithm to VLSI designing and verification.

Ming-Bo Lin received the B.Sc. degree in electronic engineering from National Taiwan Institute of Technology (now is National Taiwan University of Science and Technology), Taipei, the M.S. degree in electrical engineering from National Taiwan University, Taipei, and the Ph.D. degree in electrical engineering from University of Maryland, College Park. Since February 2001, he has been a Professor with Department of Electronic Engineering at National Taiwan University of Science and Technology, Taipei, Taiwan. His research interests include VLSI systems design, mixed-signal integrated circuit designs, parallel architectures and algorithms, and embedded computer systems. He has published over many journal and conference papers in these areas. In addition, he has directed the designs of over fifty ASICs and has consulted in industry extensively in the fields of ASIC, SoC, and embedded system designs. He was a recipient of the Distinguished Teaching Award from National Taiwan University of Science and Technology in 2007. He chaired the Workshop on Computer Architectures, Embedded Systems, and VLSI/EDA in National Computer Symposium (NCS) 2009. During the past decades, Professor Lin has translated two books and authored about thirty books (including revisions), especially including *Digital System Designs and Practices: Using Verilog HDL and FPGAs,* (ISBN: 9780470823231, John Wiley & Sons, 2008) and *Introduction to VLSI Systems: A Logic, Circuit, and System Perspective,* (ISBN: 9781439868591 CRC Press, 2012.)

A Policy-based De-duplication Mechanism for Securing Cloud Storage

Zhen-Yu Wang[1], Yang Lu[1], Guo-Zi Sun[1,2]
(Corresponding author: Guo-Zi Sun)

College of Computer, Nanjing University of Posts and Telecommunications[1]
Nanjing 210003, China
Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks[2]
Nanjing 210003, China
(Email: sun@njupt.edu.cn)
(Received Jan. 23, 2015; revised and accepted Feb. 25, 2015)

Abstract

With the rapid development of information technology, the amount of data of information systems increases at an unprecedented pace. Extension and development of clouding storage systems are becoming the first choice for businesses and individuals to store their data. However, the data is stored in the cloud and users lose the control over their data completely, so encryption is required before upload the data. But, general encryption methods have conflicts with data de-duplication. Some researchers use convergent encryption as a solution, but there are also some semantic security issues. Another threat comes from semi-trusted cloud storage providers (CSP), which can disclose users' privacy information. In this paper, we propose a policy-based de-duplication architecture, using the mechanism of security proxy (SP) and random storage, which separate storage services and security services to ensure the security of user data and improve the system efficiency at the same time.

Keywords: Cloud Storage; Convergent Encryption; De-duplication; Security Proxy

1 Introduction

In recent years, the amount of data which Enterprises and individuals create and store, have been increasing exponentially, which make cloud storage [13] popularly deployed. However, a critical issue needs to be addressed in cloud storage is data duplication. Research shows that 80% -90% of the data is redundant in backup and archival storage systems [8], and this rate is still increasing, which is a big waste. To mitigate this issue, data de-duplication technology has become a hot research topic [5].

Data de-duplication technology means that in cloud storage environment, only one copy of the same data can be stored instead of storing multiple copies. This technology can greatly reduce the storage space and communication bandwidth. According to the latest statistics, de-duplication is the most promising and effective storage technology, which will be applied to 75% of the on-line backup systems and cloud storage in the next few years [9]. However, there exists conflicts between data de-duplication and data encryption. In general encryption method, when the different encryption key is applied to encrypt the same plaintext, the cipher text generated is completely different, which makes it impossible to apply the data de-duplication. Though convergent encryption [3] can alleviate this conflict, it still suffers to the semantic security problem. The architecture in [2, 11] can realize the de-duplication in cloud storage, but it lacks the consideration of the security of user data.

Tan *et al.* [12] proposes a semantic-aware multi-tiered source de-duplication framework which improves system efficiency by eliminating redundancy detection of small files and compressed files. But it is not suitable for global de-duplication in cloud, because the data comes from large numbers of different users, which can result in massive redundant data. Liu *et al.* [7] proposes a policy-based de-duplication scheme, in which the de-duplication module is independent from cloud storage and improves protection of user privacy. However the user data must be transferred twice, which makes low system efficiency in today's limited bandwidth. Ref. [10] uses equality predicate encryption scheme and a hybrid approach for de-duplication to prevent information leakage when encrypting the outsourced data. Ref. [6] proposed an approach for secure authorized de-duplication which supports the duplicate check with differential privileges of users, but needs more user involvement. Harnik, *et al.* [4] describes a method to get some users' privacy information by detecting whether the client performed de-duplication operation or not. In addition,

the cloud storage provider is considered to be semi-trusted. Though it will not perform malicious attacks obviously, it may collect users' data and privacy information, such as Ref. [1], the data encryption is operated at cloud side, and sensitive information may be disclosed to the cloud provider easily. Even if the data is encrypted with Convergent encryption.

For the above security issues, we present the following solutions. First is the improved convergent encryption that encryption key is not only related to the data itself, but also related to the offset from the SP, which can reduce the correlation between the encryption key and the plaintext and can resist the content guessing attack well. For the illegal usage of user' information from CSP, we present the random storage mechanism. In this mechanism, we firstly use the source-based de-duplication to remove the repetitive data, and the remaining data uploaded to which storage server is according to the storage table generated by SP. In this way, we separate the storage service and security service and improve the transparency of the user data for CSP. For the metadata located in SP, sharing degree based access control can prevent the unauthorized access.

2 Security Issues in Cloud Storage

In cloud storage, user data security is entirely relied on data protection and access control mechanisms offered by the cloud service provider. However, data de-duplication will conflict with data encryption when conduct global de-duplication. When given two ciphertexts, it is difficult to determine whether they are encrypted from the same plaintext. The goal of de-duplication is that different users can share the identical files or different files can share the identical data blocks. If we want to apply the ciphertext-based de-duplication, cloud server has to identify all of the ciphertexts encrypted from the same plaintext, which is contrary to the purpose of encryption operation. Attacker can get information from the ciphertext without the decryption key.

Now some de-duplication schemes [1, 4] solve this problem using convergent encryption. Convergent encryption is similar to asymmetric encryption, except that the encryption key is generated from the hash value of the plaintext, and the encryption algorithm is based on XOR operation. The identical plaintext will be encrypted using the identical encryption key to generate the identical ciphertext, so this method is widely used in de-duplication system. However, convergent encryption also has some security issues the encryption key is directly related to the plaintext, so it cannot resist the content guessing attack [10]. Adversary can construct a specific file or data block to detect whether the file or data block has been already existed in the storage system or not, in order to obtain more information. Another threat comes from the cloud storage provider.

In this paper, we assume that providers are semi-trusted. That is to say, the clouds will work according to the predefined protocols and policies and will not attack user data obviously, but cannot protect the uploaded data from unauthorized viewing , scanning, and publicly display by the cloud storage providers (CSP), even the data is encrypted using the convergent encryption. Such as the scheme in [1], both the de-duplication and encryption of user data are done at the cloud side, which increases the amount of redundant data transmission. To make things even worse, it makes users' privacy much easier to be leaked to the cloud providers, even if convergent encryption is applied. Because the fingerprint database of blocks must be stored in the cloud server, which is equivalent to store the user data in the server in plain text. In this paper, we propose a policy-based de-duplication mechanism, using the mechanism of security proxy and random storage, which separate storage services and security services to eliminate the threat from CSP.

3 Policy-based De-duplication Mechanism

3.1 Architecture Design of Secure Cloud Storage

In our work, we proposed secure cloud storage architecture which consists of three main components: Client Security Proxy and cloud storage, as shown in Figure 1. We assume that CSPs and SP won't join up to attack user data. Multiple CSPs constitute the cloud storage, and provide storage services to the security proxy together, but are independent of each other. Our mechanism is based on source-side de-duplication, and do the chunk-level duplicate detection and convergent encryption at the client. By comparing the hash values of chunks with those in global fingerprint database, we will delete the duplicated chunks and generate the storage pointers. All the non-repetitive chunks will be convergent encrypted and uploaded to the specified cloud servers according to the random storage table generated by SP. Random storage is defined as bellows.

Definition 1. *Random Storage. Assuming that, the to-be uploaded file is f, and there are m storage servers. f is divided into n fixed size chunks first, that is, $f = \{c_1, c_2, ..., c_n\}$. n/m chunks is randomly selected to store in Server I, and another n/m chunks is stored in Server II, and so on. Then the random storage table will be return to client.*

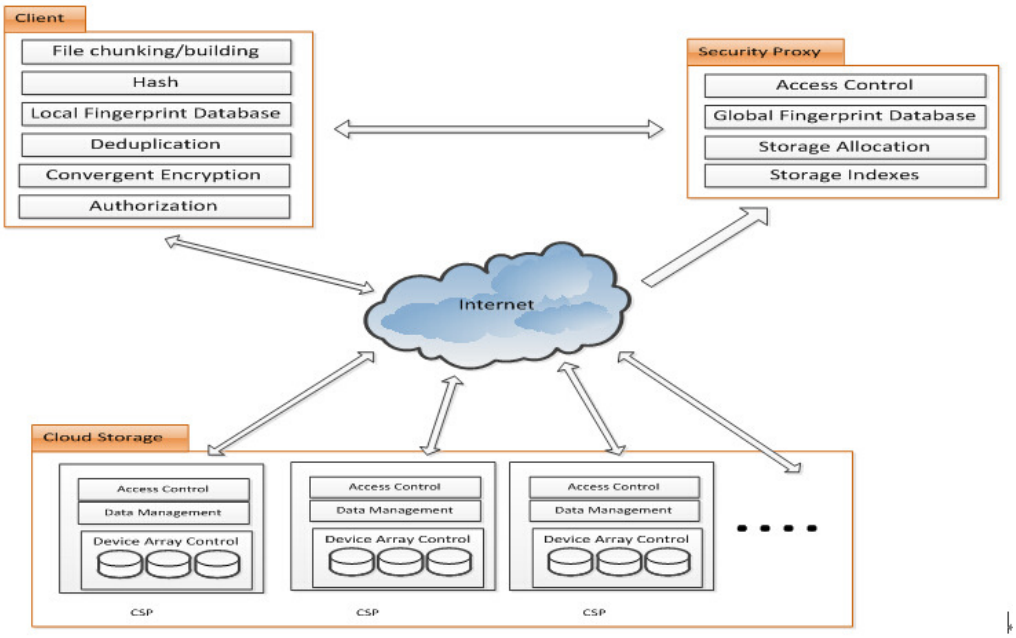

Figure 1: Securing cloud storage architecture diagram

- Client
  - File chunking/building: Conducting fixed-length chunking or content-based chunking on the file needed for block-level duplicate detection; And data blocks are assembled into files according to the metadata in the download process.
  - Hash: Calculating the Hash values of the files or data blocks as their fingerprints. Currently, the most commonly used two kinds of hash algorithms are MD5 and SHA-1. They have a very low collision probability, and we assume that there is no collision in our work.
  - Local Fingerprint Database: Storing the Hash values of files and data blocks users uploaded.
  - De-duplication: Comparing the Hash values of the files or data blocks with those in the fingerprint database on security proxy. Deciding whether perform de-duplication or not according to the comparison results returned.
  - Convergent Encryption: Encrypting the data needed to upload using $k = G(hash, offset)$ from security proxy as the encryption Key (defined in Section 3.2).
  - Authorization: Users need to match the access structure tree of SP when access the data (described in Section 3.3).
- Security Proxy
  - Access Control: Generating the access tree according to the blocks of the file. When the users access structure matches the access tree completely, he can access the cloud storage.
  - Global Fingerprint Database: Using to store the global Hash fingerprints and offsets of files and data blocks, category according to the file type and with a matching fingerprint features.
  - Load Monitor: Responsible for monitoring the load condition of every storage server in cloud.
  - Storage Allocation: Each data block will be randomly assigned to each storage server according to the fingerprint information the client uploaded and load condition of every storage server. At the same time, the server load is balanced. Then it generates storage table and return to the client and the usage of storage space is updated.
  - Storage Indexes: Recording the information of files and data blocks stored in the cloud server including file ID, file name, location of data, fingerprint and so on.
- Cloud Storage

– Access Control: Cloud servers require authentication of clients and security proxy that requesting storage services.
– Data Management: Cloud servers manage and optimize the data storage according to storage policy.

3.2 The Improved Convergent Encryption

Traditional convergent encryption uses the key which is generated with some attributes of the plaintext to encrypt the data itself, which makes the ciphertext identical after encryption of the identical plaintext. Though it achieves the objective of data encryption in a certain extent, it cannot ensure the semantic security, because of the high relativity of ciphertext and plaintext. It can only improve the security of unpredictable data, and cannot resist the content guessing attack. Especially in the enterprise environment many file are small variations of standard templates. And attacker can construct a same file and compare the ciphertext to detect the existence of it. In our paper, we improved the convergent encryption whose main improvement is the generation of the encryption key. The specific operation is as bellows and shown in Figure 2.

1) Client calculates the hash values of the data using hash function, and sends the values to security proxy.
2) After received the hash values, SP generates a random offset, and re-calculates the encryption key as $k = G(h, offset)$, the send back to client.
3) Client encrypts the data with k, and uploads the ciphertext to cloud. And the metadata (hash, offset, location) will be save in the SP servers.

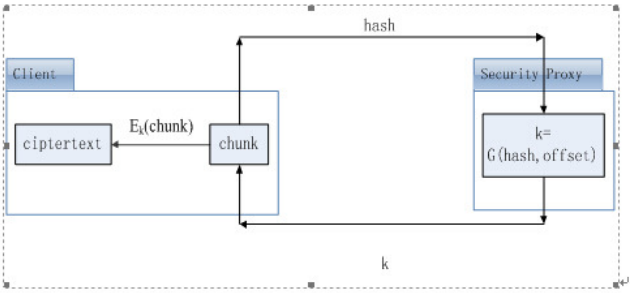

Figure 2: The Improved convergent encryption

3.3 Sharing Degree Based Access Control

Definition 2. *Sharing Degree (SD). In the de-duplication system, one data block could be shared with multiple files. The more of files the higher of the sharing Degree, and the more possibility to leak user privacy of the block.*

Definition 3. *Access Structure Tree. Each node of the tree is related to a data block, and it is represented by attribute of the block, logical operator and its child nodes. As for non-leaf node x, its logical operator is "OR", "AND" or "n/m", the threshold is k_x, and x has m_x child nodes, and 0 <k_x <m_x. As for the leaf node, threshold k_x=1, and its child nodes is none. In the access structure tree, the higher of SD of a data block, the closer it is to the root. So it needs more attribute sets when accessing the block.*

As show in Figure 3, process of matching a access structure tree: we represent the access structure tree as T, T_x is the child tree of T, and x is the root of T_x. Assuming that there is an attribute set, $T_x(v) = 1$ represents that v can match T_x. Hence, T_x(v) can be calculated with a recursive manner like this: If x is a leaf node and its attributes can match the block, then $T_x(v) = 1$; If x is a non-leaf node, the value $T_x(v)$ of x is calculated by its child nodes y_i. Iff there is more than k_x values of $T_{y_i}(v)$ is 1, the value of $T_x(v)$ is 1. When client applies to access a file f, SP will generate a access tree T according to the blocks of f, and will ask the client to return the access structure v. Iff v matches T, SP will return the encryption key k (k is defined in Section 3.2) to client.

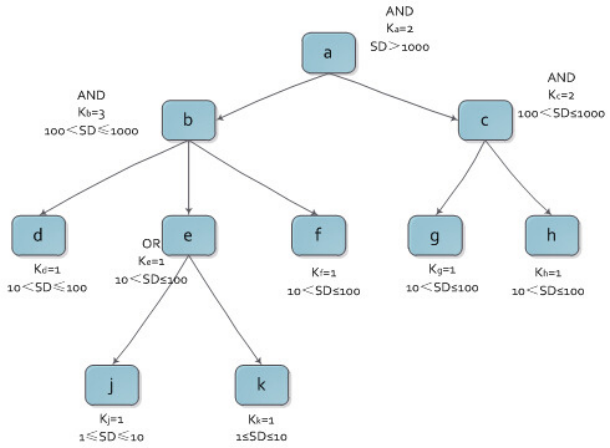

Figure 3: Schematic diagram of the SD based access structure tree

3.4 Process of Data Access

Data upload:

1) File-level redundancy detection:
   a. Calculate Hash value of file, and send them to the Security Proxy;
   b. Security Proxy compares them with the fingerprint database, and return the compare result to the client. If the file already exists, then delete the file and generate the storage pointers; else perform the block-level detection.
2) Block-level redundancy detection:
   a. Firstly, cut the file into blocks and calculate the Hash values, and then send the Hash values to the Security Proxy;
   b. Security Proxy compares them with the fingerprint database; If the file already exists, then delete the file, and generate storage pointers, otherwise encrypt them using convergent encryption.
3) Security proxy generates a random storage table and ensures the load balance of every storage server: each data block will be randomly assigned to storage servers. In other words, suppose there are N data blocks, it randomly selects N/M blocks of data stored to Server I, and another N/M blocks to Server II, and so on. Then send the scheme to the client and update both the global index fingerprint database and the file storage database.
4) Client calculates MD5 checksum C_i of each block D_i, and then package both blocks and corresponding checksum, and upload them to the assigned storage server, which is according to the random storage table.
5) Each cloud server decompresses the package, and calculates the checksum C_i' of each block D_i again, if $C_i = C_i'$, then stores the data, otherwise returns the check fails information to the client and asks to re-upload the data block.

The data upload process is shown in Figure 4.

Data download:

1) Client sends a request to the Security Proxy for downloading a file.
2) Security Proxy firstly verify the legality of the request, and then search storage index database according to the file ID to get the metadata (SI(fID, fname, k, Location, Chunks,...)) of each block of the file, and return to the client.
3) Client sends the download request to the corresponding cloud storage server according to the metadata.
4) Each cloud server returns the Chunks(chunk1, chunk2, chunk3, ...) to client after verifies the legality of the request.

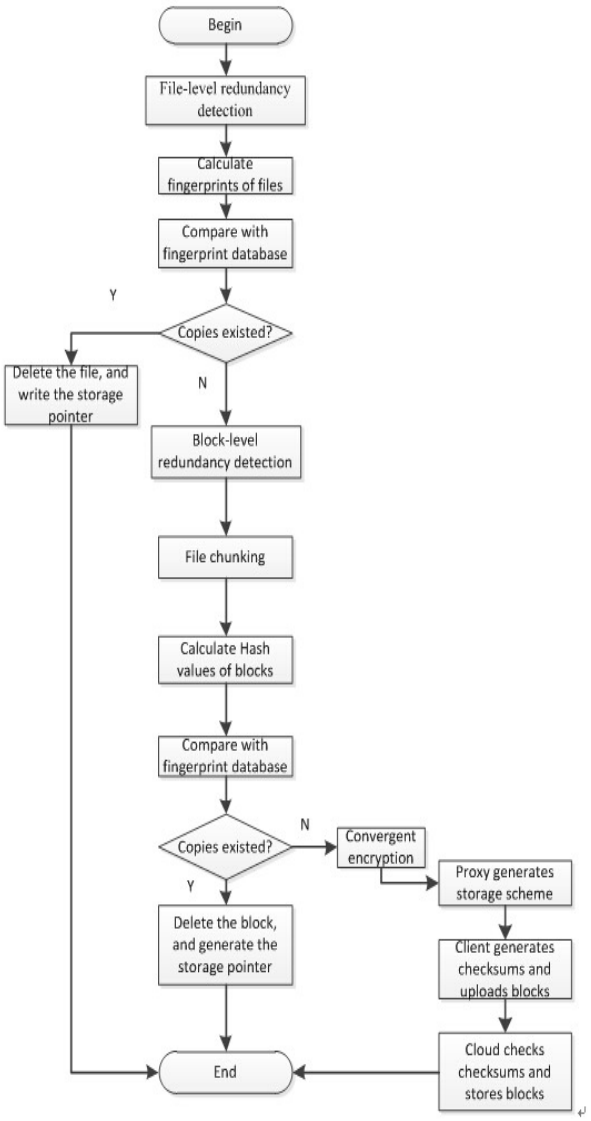

Figure 4: Flow chart of data upload

5) After receives Chunks, client searches in the fingerprint database to get the Fingerprints(fg1, fg2, fg3, ...), then decrypts ciphertext, and restore the original file.

It should be noted that, all interaction information between client and security proxy and between client and cloud server has a stamp and is encrypted. Client must pass authentication and timeliness certification of security proxy and cloud storage when to upload and to download files.

The data download process is shown in Figure 5.

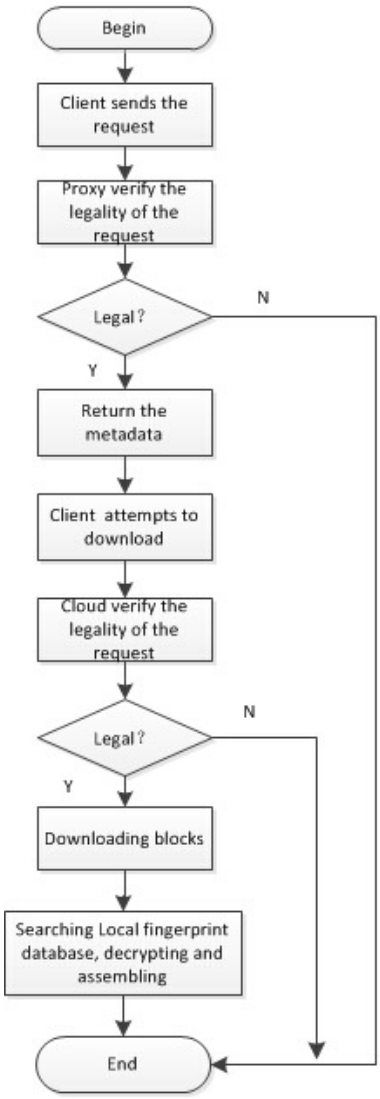

Figure 5: Flow chart of data download

4 Experiments and Analysis

In our experiments, the working platform configuration is show in Table 1. All the server nodes are virtual machines on the Langchao server. And the system is based on Ubuntu and the open source cloud OpenStack, mainly uses the Swift component of OpenStack to provide storage service. The structure of our cloud system is show as Figure 6. The master-1 is a control node used to run the Keystone and Swift service of Openstack, and it is as the SP node. The Node-1, Node-2, Node-3, Node-4 are computation and storage nodes, which are used to run the nova computing workstation. The IP-san is a RIAD used to provide physical storage space for control and compute nodes.

Table 1: Working platform

Server	Langchao NF8560M2
CPU	Intel(R) Xeon(R)E7-4807@1.87GHz
Number of cores	24
Memory	32G
Hard disk	10T
System platform	VMware vSphere 5.1.0
IP address	10.10.20.41

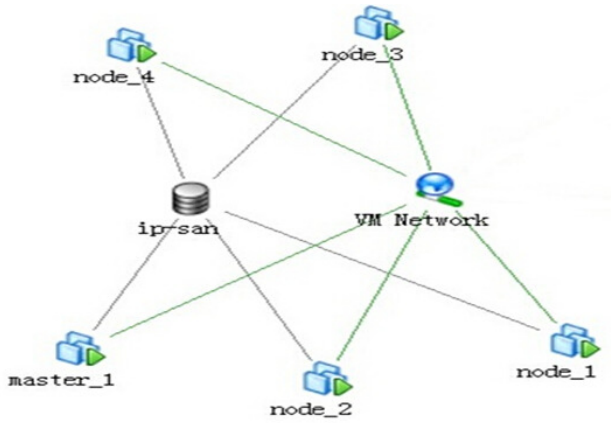

Figure 6: Structure of our cloud system

4.1 Security Analysis of Improved Convergent Encryption

Traditional convergent encryption can enhance the confidentiality of data, but cannot resist the content guessing attack. Assumed that a user will store the file $f = (a_1, a_2, \cdots, a_i, \cdots, a_n)$ to the cloud (a_i represents the i^{th} segment of the file), $h = hash(f)$ is the hash value calculated with a hash function. In traditional convergent encryption, h is the encryption key, and the ciphertext is $c = E_h(f) = E_{hash(f)}(f)$. In the condition mentioned in [9], if attacker knows all the segments of f except a_i, he can construct a file $f^* = \{a_1, a_2, ..., a_i^*, ..., a_n\}$, then calculate $h^* = hash(f^*)$ and $c^* = E_{hash(f*)}(f^*)$. In this condition, he can construct f^* continuously and compare c^* with c to reach his attack objective.

In our improved convergent encryption, the hash value of f is sent to security proxy first. Then, the security proxy will generate a new key k=G(h,offset) with h and the offset, and send k to client. The eventual ciphertext will be $c = E_{G(hash(f),offset)}(f)$ at client. In the way, the attacker cannot calculate the key according to the plaintext, because the key is decided by the hash value together with the offset generated by security proxy. Attack cannot be carried out when plaintext and key are unknown, that is to say, our improved convergent can resist the content guessing attack well.

4.2 Security and Performance Analysis of Random Storage

In this paper, we use the random storage to remove the threat from semi-trusted CSP. Secure proxy decides where to store the data blocks according to the random storage table, but the storage table is transparent to each cloud storage server. Cloud servers are only responsible for data storage and basic management, and each of them just stores some random blocks of a file. Assume that a cloud server has obtained some information via some method, which shows that some blocks are belong to a specific user. These data blocks are only some fragments of a file, and are completely messy and meaningless for the cloud server, so the cloud is impossible to dig out the information relevant to the user's privacy from these fragments. Metadata is stored in Secure Proxy. Based on the assumption referred before, Cloud Server cannot get the metadata and doesn't know where the other segments of the file are located in. So Cloud Server will not be able to obtain the complete file. User information and storage information of files are stored in Secure Proxy, while, data uploading and downloading are performed directly between the clients and cloud storages. Security Proxy cannot come into contact with the user data and cannot attack the user data,

too.

Paper [7] uses the re-encryption at proxy to improve security. That is, the user firstly uploads data to the proxy, and then user data is re-encrypted at the proxy server. While, the random storage used in this paper can greatly improve system performance, reduce bandwidth usage and eliminate the need for re-encryption which significantly improves system efficiency. As shown in Figure 7. We use 256-bit AES encryption and does encryption test on the datasets in which every blocks size is 4KB, and the data sets size are 1M, 10M,... ,and 1G. As can be seen from the figure, the encryption time is the exponential growth, which can be a heavy pressure for the proxy server in the application of cloud storage. In our scheme, we use client-end de-duplication, and the encryption is done at client. In this way, the computation pressure can be shared by all the clients.

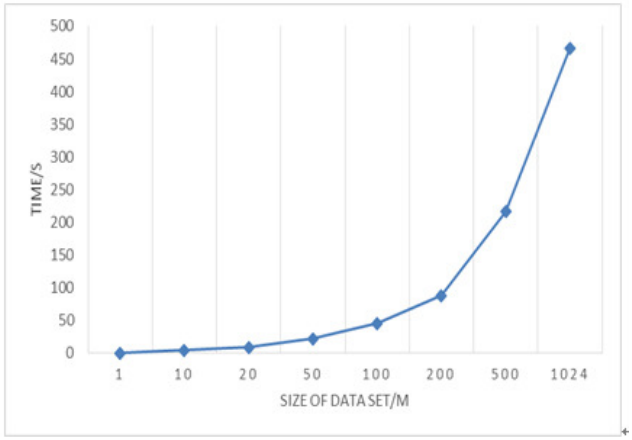

Figure 7: Time consuming of encrypting the dataset

In [7], Liu separated de-duplication module from the storage server, which makes user data need to be transmitted twice. First, the client uploads user data to De-duplication Proxy, and then the proxy stores the data to cloud storage servers. But the random storage used in this paper needs transferring user data only once. Security Proxy only generates storage table and Client decides which Cloud Server will each blocks be stored in according to the storage table, which can cut network load in half. In addition, with the load balance mechanism and multithread technology, the reduction of system efficiency caused by over load and empty load in some storage servers will not happen. In our experiments, as shown in Figure 8, the efficiency of data upload in our scheme is about 5.44 times than that in [7].

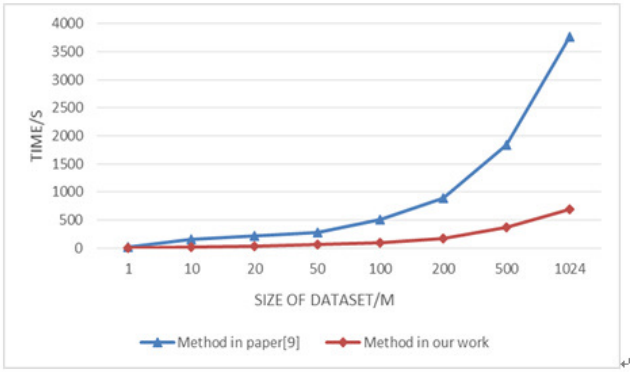

Figure 8: Uploading time comparison

5 Conclusion

The cloud storage can provide powerful computing performance and massive storage, and there is no doubt of its prospects for development. But the cloud storage technology has not been widespread so far, there are two main

reasons hinder its rapid development. First is the performance problem. When the users increase quickly but the CSPs do not improve their equipment timely, all the users will share the limited server resources, and the performance of data access will decline, which is one of the most important risk factors to be considered when enterprises choose cloud storage. Second is the data confidentiality and privacy security. Users store their data in the cloud which means lose the control of their own data and cannot ensure the confidentiality and privacy. It is more important to the business data.

In the secure cloud storage, de-duplication is the most effective lossless compression technology, and can greatly improve the utilization and performance of cloud storage. However, it brings some problems of data security and privacy. In this paper, based on the existing de-duplication technology, we propose the security proxy and random storage strategy, which separate the security service and storage service. In this way, we resolve the conflict between data encryption and de-duplication, resist the attack from outside, and prevent the illegal use of user data and privacy from CSP. In addition, the improved convergent encryption and SD based access control can resist the content guessing attack well. From the experiments and security analysis, the storage table we propose can improve the performance and security of cloud storage system.

Acknowledgments

This study was sponsored by the National Natural Science Foundation of China (Grant No. 61373006), the Foundation of Nanjing University of Posts and Telecommunications (Grant No. NY212059), and A Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD).

References

[1] P. Anderson and L. Zhang, "Fast and secure laptop backups with encrypted de-duplication," in *Proceedings of the 24th International Conference on Large Installation System Administration (LISA'10)*, pp. 1–12, 2010.
[2] C. G. Bi and X. L. Xu, "Data de-duplication mechanism for cloud storage systems," vol. 31, pp. 3052–3055, 2014.
[3] C. I. Fan, S. Y. Huang, and W. C. Hsu, "Hybrid data deduplication in cloud environment," in *2012 International Conference on Information Security and Intelligence Control (ISIC'12)*, pp. 174–177, 2012.
[4] D. Harnik, B. Pinkas, and S. P. Alexandra, "Side channels in cloud services, the case of deduplication in cloud storage," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 40–47, 2010.
[5] A. Li, J. W. SHU, and M. Q. Li, "Data de-duplication techniques," vol. 21, pp. 916–929, 2010.
[6] J. Li, Y. Li, X. Chen, P. Lee, and W. Lou, "A hybrid cloud approach for secure authorized deduplication," *IEEE Transactions on Parallel and Distributed Systems*, 2014.
[7] C. Liu, X. Liu, and L. Wan, "Policy-based de-duplication in secure cloud storage," *Communications in Computer and Information Science*, vol. 320, pp. 250–262, 2013.
[8] J. McKnight, T. Asaro, and B. Babineau, *Digital Archiving: End-User Survey and Market Forecast 2006–2010.* 2006.
[9] D. Russell, *Data deduplication will be even bigger in 2010.* Feb. 2010.
[10] Y. Shin and K. Kim, "Efficient and secure file deduplication in cloud storage," *IEICE TRransactions on Information and Systems*, vol. 97, no. 2, pp. 184–197, 2014.
[11] Y. Song, L. Yi, and F. X. Wang, "Research and design of data de-duplication architecture based on cloud storage," *Computer Systems and Applications*, vol. 22, no. 1, pp. 208–211, 2013.
[12] Y. Tan, H. Jiang, D. Feng, L. Tian, Z. Yan, and G. Zhou, "Sam: A semantic-aware multi-tiered source de-duplication framework for cloud backup," in *International Conference on Parallel Processing (ICPP'00)*, pp. 614–623, 2010.
[13] N. H. Yu, Z. Hao, J. J. Xu, W. M. Zhang, and C. Zhang, "Review of cloud computing security," *Acta Electronica Sinica*, vol. 41, no. 2, pp. 371–381, 2013.

Zhen-Yu Wang. He is a postgraduate of Nanjing University of Posts and Telecommunications, majoring in information security. His main research direction is security in cloud storage. Email:wzy9018@126.com.

Yang Lu. He is a postgraduate of Nanjing University of Posts and Telecommunications, majoring in computer technologies. His main research direction is data analysis. Email:guaijiangnan@163.com.

Guo-zi Sun. He received Ph. D. degree in mechanical engineering from the Nanjing University of Aeronautics and Astronautics in 2002. Currently, he is a professor at Nanjing University of Posts and Telecommunications. His research interests include network security and computer forensics. Email: sun@njupt.edu.cn.

Performance Modelling and Acceleration of Binary Edwards Curve Processor on FPGAs

Ayantika Chatterjee[1] and Indranil Sengupta[2]
(Corresponding author: Ayantika Chatterjee)
School of Information Technology, Indian Institute of Technology, Kharagpur, India[1]
(Email: cayantika@gmail.com)
Department of Computer Science and Engineering, Indian Institute of Technology, India[2]
(Email: isg@iitkgp.ac.in)
(Received May 28, 2014; revised and accepted Nov. 15, 2014)

Abstract

Binary Edwards Curve has evolved as an alternative to conventional elliptic curve cryptography which is prone to operational point attacks. However, comparatively slower unified scalar multiplication algorithm of this curve poses design challenges to hardware designers. FPGA, as opposed to ASICs due to their specific look-up-table based underlying architecture, provides unique challenges and opportunities for the design of such complex circuits. In this work, as opposed to an ad-hoc design methodology, we focus on developing an efficient architecture for scalar multiplication on binary Edwards curve in an analytical fashion. The method first identifies the tunable parameters of the architecture, followed by developing analytical estimates of the resources used and the critical path delay of the circuit in terms of the design parameters and the FPGA characteristics. Detailed analytical and experimental results have been provided to show that the model indeed helps to develop an architecture with improved efficiency with respect to other reported results on similar platform.
Keywords: Binary Edwards Curve (BEC); Elliptic Curve Cryptography (ECC); FPGA.

1 Introduction

Public key cryptography is of growing importance in the domain of key-generation, digital signature and data encryption. With the growing complexity of public key algorithms, design exploration is a challenging job. While a naive design approach may be correct in functionality, proper design decisions can help to improve significantly the performance of the designs. Such a modeling can help to reduce the design time and converge to ideal design point, thus improving the productivity of the crypto-hardware. However, developing such a design methodology requires capturing of the design parameters which in turn depends on the algorithms, and also analyzing their effects on the performance, which again depends on the platform. In this work we address the topic of designing Binary Edwards Curve (BEC) [1] based processor on LUT-based FPGA platform. BEC is an interesting development in elliptic curve cryptography, which is the next generation public- key cipher. BEC computations alleviates the ECC of its two weaknesses, owing to its lack of completeness and unifiedness. This makes BEC ideal for high security implementations, as they are resistant against simple power analysis and exceptional point attacks.

However, BEC computations are more complex. Each unified addition or doubling in its underlying scalar computation, which is the central operation of an Elliptic Curve processor, requires large computations for both addition and doubling compared to conventional ECC on $GF(2^m)$ fields. This makes design of BEC an interesting topic of research. In literature, Edwards first proposed that every elliptic curve over a non-binary field is birationally equivalent to Edwards form over an original field or the extension of the field [4]. However, this curve lacks its elliptic property in the domain of $GF(2^m)$. Further, Bernstein et al. in [1] extended the idea of this curve in binary domain in the name of Binary Edwards Curve (BEC). In [9] and [8], implementations based on this curve are explained in ASIC domain. The design scope of implementing processor in FPGA domain based on this curve is first explored in [2]. A few other contributions on this curve are present in [7] and [3].

In this paper, we have emphasized on modelling a processor based on BEC by minimizing the delay and required area as these two parameters play the most critical role in any hardware design. Further, simultaneous minimization of both the parameters is not always possible. Reduction of required time may increase the area. On the other hand, restrictions on resources and area can end up into higher clock cycle requirement. Hence, starting from the

algorithm, we have gradually developed the hardware and identified the critical parameters for each module which can play a major role in optimization of the overall design. Finally, we have targeted to design a modelled hardware using optimized parameter. The mathematical background of BEC demands that the design is expected to be side channel attack preventive. To validate this about the optimized design, simple side channel analysis is performed on the modified processor to prove that it is really simple power attack preventive.

The remaining part of this paper is organized as follows. In Section 2, algorithms related to BEC are discussed. The way of developing the architecture gradually from the unified addition algorithm is mentioned in Section 3. The critical tunable parameters are obtained and the delay computation is performed in order to obtain the optimized theoretical model of the design in this section too. At the end, the final optimized design is implemented in the hardware domain and the results are compared with some previous implementations in Section 4 and analysis in Section 5 shows that the processor is truly simple side channel attack preventive.

2 Preliminaries

Like other elliptic curve based cryptographic algorithms, Binary Edwards Curve (BEC) (a variant of elliptic curve) relies on the formation of an underlying arithmetic operation on the elements of the group, which are points on the curve. The central component of ECC algorithms are scalar multiplications, which involves in computing $Q = k.P$ (P is a point on prime curve). This point multiplication can easily be calculated once k and P are known, but it is a hard problem to find k when Q and P are known, provided k is sufficiently large. This hardness of discrete log problem is the basic assumption of security of any ECC [10].

Point multiplication is computed using repetition of two fundamental operations: addition and doubling. These operations are further obtained by arithmetic operations in the field on which the co-ordinates of the elliptic curve points are defined. Hence, the basic requirement of a ECC processor is the design of an optimized addition module. In the next subsections, we shall discuss subsequent algorithms for implementing point multiplication using unified addition of BEC.

2.1 Binary Edwards Curves

Let K be a field with $characteristic(K) = 2$. Let d_1, d_2 be elements of K with $d_1 \neq 0$ and $d_2 \neq d_1^2 + d_1$. The Binary Edwards Curve with coefficients d_1 and d_2 is the affine curve $E_{(B,d_1,d_2)}$: [1]

$$d_1(x + y) + d_2(x^2 + y^2) = xy + xy(x + y) + x^2y^2. \quad (1)$$

This curve is symmetric with x and y and so if (x_1, y_1) exists on the curve, then also will (y_1, x_1). The neutral element of the addition law is (0,0). Another important property of this curve is that negative of (x_1, y_1) is (y_1, x_1). This property is effectively used in our implementation. While implementing the ternary operation [2], both addition and subtraction are required. The subtraction property is implemented by one addition followed by swapping of co-ordinates.

Require: $P = (x.y)$, $k = (k_{l-1}k_{l-2}k_{l-3}...k_0)$.
Ensure: $Q = (x', y') = kP$.
1: $Q = 0$
2: **for** i from $l - 1$ downto 0 **do**
3: $Q \leftarrow 2Q$
4: **if** $k_i = 1$ **then**
5: $Q \leftarrow Q + P$
6: **end if**
7: **end for**
8: **return** Q

Figure 1: Double and add algorithm

The main target of our work is to implement point multiplication on BEC. The scalar point multiplication can be performed by Double and Add algorithm as shown in Figure 1 or its improvements. The improvements are either to improve the speed (using windowing method) or to reduce the side channel leakage (using Montgomerry's ladder method or point blinding technique) [5]. The basic double and add algorithm requires $(m - 1)$ point doubling (PD) and $(w - 1)$ point additions (PA), where m is the length and w is the Hamming weight of binary expansion of k. However, implementing PA or PD in affine co-ordinates requires costly inversion and hence addition in projective coordinates is beneficial [10]. $(x_1/z_1, y_1/z_1^2)$ in affine coordinate is equivalent to $(X_1 : Y_1 : Z_1)$ in Lopez-Dahab

projective coordinate [10]. During implementation, initially the affine coordinates are converted to projective and the addition is performed in projective coordinate to avoid the cost of inversion.

The projective closure of BEC curve shown in Equation (1) is:

$$\begin{aligned} & d_1(X+Y)Z^3 + d_2(X^2+Y^2)Z^2 \\ & = XYZ^2 + XY(X+Y)Z + X^2Y^2. \end{aligned} \tag{2}$$

Let, sum of two points (X_1, Y_1) and (X_2, Y_2) be (X_3, Y_3). The resultant point on the projective curve is defined according to the addition formulae explained in [1]:

Table 1: Unified addition law

W_1=$X_1 + Y_1$	W_2=$X_2 + Y_2$	A=$X_1(X_1 + Y_1)$
B=$Y_1(Y_1 + Z_1)$	C=$Z_1.Z_2$	D=$W_2.Z_2$
E=$d_1.C^2$	H=$(d_1.Z_2 + d_2.W_2).W_1.C$	I=$d_1.C.Z_1$
U=$E + A.D$	V=$E + B.D$	S=$U.V$
X_3=$S.Y_1 + (H + X_2.(I + A.(Y_2 + Z_2))).V.Z_1$		
Y_3=$S.X_1 + (H + Y_2.(I + B.(X_2 + Z_2))).U.Z_1$		
Z_3=$S.Z_1$		

In the subsequent sections, we shall concentrate to design a modelled BEC processor based on this unified addition. The modelled hardware should exhibit optimum performance with a trade-off between area and delay.

3 The BEC Processor Architecture

The BEC processor offers several design choices. In this section, we attempt to design a compact processor based on BEC. In general, a processor consists of two main modules: Register file for temporary storage of data and Arithmetic logic unit (ALU). Several arithmetic and logical operations are performed in ALU and temporary data are stored in the register file. Proper scheduling of these operations is crucial for overall performance of the processor. To achieve enhanced performance, we first develop an abstract overview of the architecture and identify the tunable parameters. Tunable parameters include all the variable features of the architecture, which has an influence on the circuit performance, namely area and speed. In our approach, we analytically model those performance metrics using the identified tunable parameters, which are subsequently optimized to develop an efficient architecture.

Algorithm 1: Algorithm for designing optimized processor

Input: Input algorithm to be realized by hardware
Output: Optimum Delay and LUT product for overall hardware

```
initialization;
Register_Scheduling() ;
ALU_Scheduling() ;
Total_Delay = Computation of delay for the multiplier ;
Total_LUT = Computation of LUT requirement for the multiplier ;
LUT_DelayProduct = Total_Delay * Total_LUT ;
for all possible critical paths do
    Estimate the optimum LUT_DelayProduct ;
```

Algorithm 1 explains the steps for optimizing the BEC based processor. Next, we explain the scheduling of register and ALU module before discussing the design choices for optimization of the processor.

3.1 Register Scheduling

Scheduling is the task of determining the instants at which execution of operation will start and finally the functional units are mapped. To obtain an optimized scheduling for the operations of Table 1 a simple strategy for scheduling is applied as explained in Algorithm 2.

Table 2: Minimum register requirement

Operations	New input variable	New Output variable	Register used	Register free	Total register
$(X_1 + Y_1)$	2	0	2	0	2
$(X_2 + Y_2)$	2	$1(W_1)$	3	0	5
$X_1(X_1 + y_1)$	0	1 (W_2)	1	0	6
$(Z_1.Z_2)$	2	$1(A)$	3	0	9
$(W_2.Z_2)$	1	1	2	0	11
$(d_2.W_2)$	1	1	2	2	11
H	1	1	2	1	12
E	0	1	1	1	12
$d_1.C.Z_1$	0	1	1	2	10
U	0	1	1	0	11
$I + A(Y_2 + Z_2)$	0	1	1	0	12
V	0	1	1	1	12
S	0	1	1	2	11
$V.Z_1$	0	1	1	1	11
X_3	0	1	1	1	11

Algorithm 2: Register_Scheduling

Input: Input algorithm to be realized by hardware
Output: Optimum number of used registers after scheduling

initialization;
$nw\_inputvar$ = number of new input variables for each scheduling step ;
$nw\_outputvar$ = number of new output variables for each scheduling step ;
Reg_free = number of registers can be reused ;
$Reg\_used = nw\_inputvar + nw\_outputvar - Reg\_free$;
for *each operation* **do**
  if *Reg_free ! = 0* **then**
    $Reg\_used = Reg\_used + nw\_inputvar + nw\_outputvar$;
  else
    $Reg\_used = Reg\_used + nw\_inputvar + nw\_outputvar$ - Reg_free;

For each step of Table 1, requirement of each register is computed based on the number of new input and output variables and the available free resources. In Table 2, first and second columns keep the count of new input and output variables in each stage, the third column counts the total register and the fourth one counts the number of registers becoming free at each stage. Finally, from the table it is evident that at least 12 registers are required for maintaining the scheduling constraints.

Using this count, overall scheduling with the registers are shown in Table 3 for the operations of unified addition shown in Table 1.

In the next subsection, we shall explain the steps for scheduling ALU module.

3.2 ALU Scheduling

In case of scheduling ALU, we use the knowledge from register scheduling explained in Table 3. In this section, we identify the arithmetic logic components to implement the operations in Table 1. The main components are field adder-subtractor and multiplier for field elements. However, the addition algorithm which is used to realize the BEC point addition steps is explained in Table 1.

As the point addition is performed in projective co-ordinates, another field inversion is required finally to reduce the result to affine co-ordinates. Hence, in our case, the identified modules are field multiplier, multiplexers and power block for inversion. Initially, we have identified the different required modules. Further, number of multipliers have been identified from the parallelization of scheduling table. Next, from the possible inputs of multiplier numbers and sizes of the multiplexers are determined. The overall ALU scheduling technique is explained in Algorithm 3.

Table 3: Scheduling for projective addition

Clock	State	Operation(C_1)	Operation(C_0)
1	Initial	$RA_2 \leftarrow X_2$	$RB_2 \leftarrow Y_2$
2	Initial	$RC_1 \leftarrow Z_1$	$RC_2 \leftarrow Z_2$
3	Initial	$RA_1 \leftarrow X_1$	$RB_1 \leftarrow Y_1$
4	S_1	$RE_1 \leftarrow A_2 + B_2$	$RC_2 \leftarrow (A_2 + B_2).C_2$
5	S_2	$RD_1 \leftarrow d_1$	$RC_2 \leftarrow C_1.C_2$
6	S_3		$RA_2 \leftarrow D_1.C_2$
7	S_4		$RB_2 \leftarrow A_2.C_2$
8	S_5	$RD_2 \leftarrow d_2$	$RA_2 \leftarrow A_2.C_1$
9	S_6		$RD_2 \leftarrow D_2.E_1$
10	S_7	$RE_2 \leftarrow Z_2$	$RC_2 \leftarrow (A_1 + B_1).C_2$
11	S_8		$RE_1 \leftarrow D_1.E_1$
12	S_9		$RA_2 \leftarrow (E_1 + D_2).C_2$
13	S_{10}		$RA_2 \leftarrow (A_1 + C_1).C_1$
14	S_{11}	$RC_2 \leftarrow Z_2$	$RD_2 \leftarrow D_1.E_2$
15	S_{12}	$RD_2 \leftarrow B_2 + D_2$	$RF_1 \leftarrow B_1.(B_1 + C_1)$
16	S_{13}		$RF_2 \leftarrow F_1.E_2$
17	S_{14}		$RE_2 \leftarrow (B_2 + F_2).C_1$
18	S_{15}	$RB_2 \leftarrow Y_2$	$RF_2 \leftarrow (B_2 + F_2).D_2$
19	S_{16}	$RB_2 \leftarrow X_2$	$RD_1 \leftarrow D_1(B_2 + C_2)$
20	S_{17}		$RD_1 \leftarrow (D_1 + A_2).B_2$
21	S_{18}		$RE_2 \leftarrow (E_1 + D_1).E_2$
22	S_{19}		$RB_1 \leftarrow F_2.B_1$
23	S_{20}		$RD_1 \leftarrow F_2.A_1$
24	S_{21}	$RA_1 \leftarrow (B_1 + E_2)$	$RD_2 \leftarrow D_2.C_1$
25	S_{22}	$RB_1 \leftarrow Y_2$	$RB_2 \leftarrow (B_2 + C_2).F_1$
26	S_{23}	$RA_2 \leftarrow X_2$	$RB_2 \leftarrow (A_2 + B_2).B_1$
27	S_{24}	$RB_2 \leftarrow Y_2$	$RD_2 \leftarrow (E_1 + B_2).D_2$
28	S_{25}	$RB_1 \leftarrow (D_2 + D_1)$	$RC_1 \leftarrow F_2.C_1$

Algorithm 3: ALU_Scheduling

Input: Input algorithm and register scheduling table
Output: Required ALU submodules and their optimum sizes

initialization;
Identify the number of multipliers from scheduling table ;
for *operation* C_0 **do**
  n_1 = number of possible inputs to one input of the multiplier ;
  n_2 = number of possible inputs to other input of the multiplier ;
  size of input multiplexer to the first input of multiplier = $n_1 : 1$;
  size of input multiplexer to the second input of multiplier = $n_2 : 1$;
for *operation* C_1 **do**
  Estimate the number of possible inputs to the output multiplexer ;
Estimate the size of the power block for inversion ;

3.3 Overall Architecture Analysis

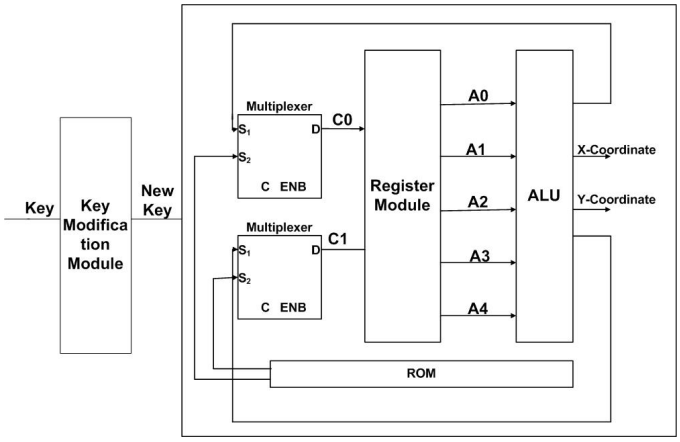

Figure 2: Final architecture of BEC processor

In previous sections, we have analyzed the modules require to compute scalar multiplication. Figure 2 depicts the BEC processor capable of performing the multiplication kP. The processor mainly consists of Register module and the ALU connected with proper bus. ROM is used to store initial coordinates of the point P, for calculating kP and the required curve parameters of complete BEC for performing the multiplication. However, in the diagram another Keymodification module is present. As explained in [2], this module is required for modification of key to increase the overall performance. However, since this module is directly not connected with the kP multiplication, we are not considering this module in the analysis of optimization.

Figure 3 describes the register details of the implemented architecture. Due to the use of projective coordinates, extra registers are required for storing Z_1 and Z_2. As a whole, 12 registers are required for implementing addition including 4 intermediate registers as decided in Section 3.1. Multiplexers MC0 and MC1 are used for selecting the input to choose from ALU output or ROM (which holds the value of initial point P_1). Another three multiplexers M01, M02, M03 are used to select input for Arithmetic Logic Unit from registers.

Figure 4 describes the architecture of the ALU of this design based on the identified modules in Section 3.2. It is having five inputs from the register file and two outputs corresponding to two parallel operation sequences as mentioned. One multiplication is done in each step for the efficient use of single ALU and the addition operation is done in parallel in order to save clock-cycle. Outputs of $MUXA$ and $MUXB$ are the inputs to the multiplier block. Depending of the selection of $MUXD$, output of the multiplier block (data of bus $ALUC_1$) is either fetched as the input of the power block and the output of power block directly goes to the bus C_0, else output of $MUXC$ is stored as data of bus $ALUC_0$. Then with the change of clock cycle, data of bus $ALUC_0$ and $ALUC_1$ are processed by the register module. From this figure, again the possible data paths can be identified as shown in Table 4.

Since, delay of $DPath3$ is much lesser than the other two paths due to the absence of multiplier, the critical path of the design must consists of MuxA or MuxB, the multiplier and the power block along with the associated

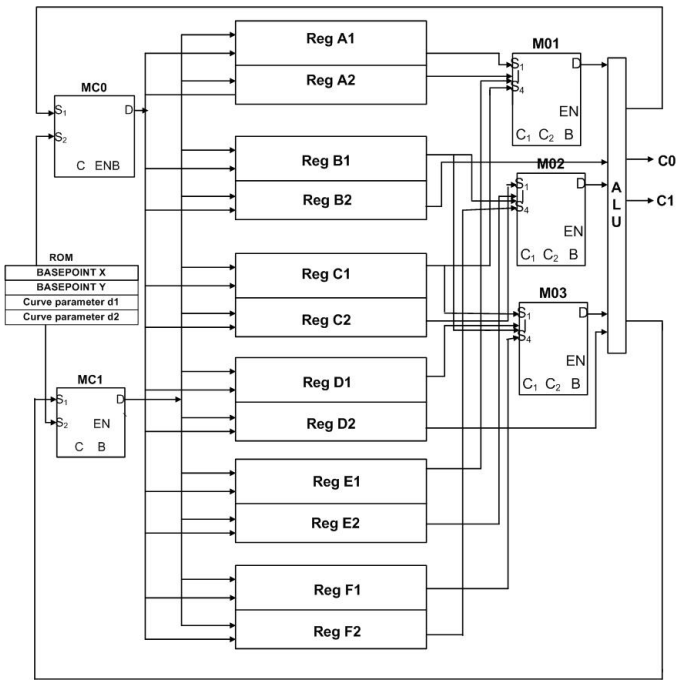

Figure 3: Register module of the processor

Table 4: Different possible functional paths

Path	Submodules
DPath1	$MuxA \rightarrow Multiplier \rightarrow MuxD \rightarrow powerblock \rightarrow Muxout$
DPath2	$MuxB \rightarrow Multiplier \rightarrow MuxD \rightarrow powerblock \rightarrow Muxout$
DPath3	$MuxC \rightarrow Muxout$

multiplexer. We ensure that $DPath1 \equiv DPath2$ in terms of delay. In the next subsections, we shall compute the overall delay and LUT requirement for the processor.

3.4 Delay Optimization of the ALU Unit

The overall ALU module is scheduled according to the algorithm 3 and finally the overall processor is scheduled following the algorithm 1. In next subsections, we shall observe the detailed optimization of all submodules like multiplication module and inverse module.

3.4.1 Optimization of the Multiplier

For field multiplication in this processor, hybrid Karatsuba multiplier with sub-quadratic complexity is used. The multiplier which is the most critical block, can be modelled based on the parameters of the underlying algorithm. Hybrid Karatsuba multiplier is the mingle of both general and simple Karatsuba multiplier. General Karatsuba multiplier [6] is better for maximum utilization of LUT for smaller bits. On the other hand, Simple Karatsuba is beneficial for minimizing gate counts for higher bits. To ensure both the advantages, General Karatsuba is used when $m \leq 29$, else Simple Karatsuba is used. In General Karatsuba multiplier, the m-bit multiplicands $A(x)$ and $B(x)$ are divided into A_h, A_l and B_h and B_l respectively:

$$A(x) = A_h.x^{m/2} + A_l \quad (3)$$

$$B(x) = B_h.x^{m/2} + B_l \quad (4)$$

$$\begin{aligned} C(x) &= (A_h.x^{m/2} + A_l)(B_h.x^{m/2} + B_l) \\ &= A_h.B_h.x^m + (A_h.B_l + A_l.B_h).x^{m/2} \\ &\quad + A_l.B_l. \end{aligned} \quad (5)$$

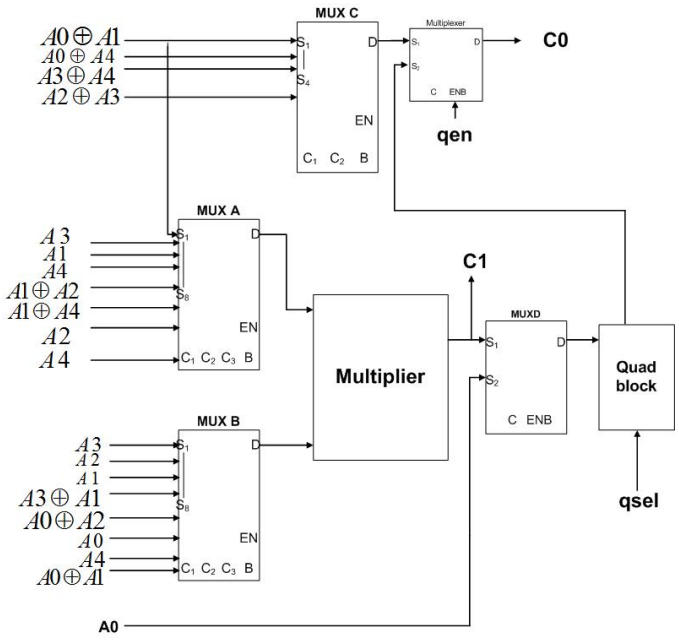

Figure 4: ALU of BEC processor

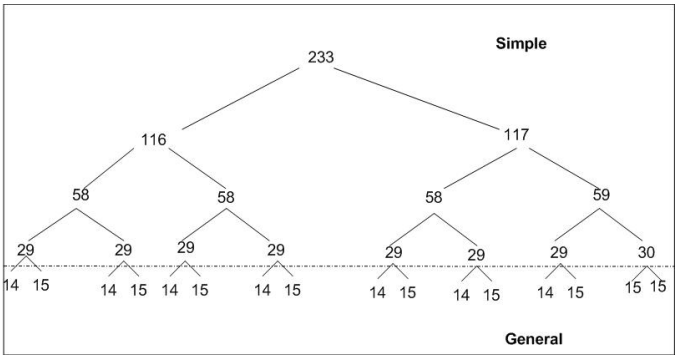

Figure 5: Hybrid Karatsuba multiplier tree

Delay calculation for multiplier: An m-bit Hybrid Karatsuba multiplier is used for multiplication in this design. The Karatsuba multiplier in turn calls school-book multiplier [11]. The delay associated with τ-bit schoolbook multiplier is $D_{sbmultiplier}$ and it is expressed as $log_k 2\tau$ [13] for a k-input LUT based device.

Total delay associated with the multiplier is equal to height of the tree plus $log_k 2\tau$, i.e. $log(m/\tau) + log_k 2\tau$.

From the above relation, it is clear that with fixed field(m) and LUT device(k), τ is the main tuner of the delay equation. According to the FPGA property and previous experimental results [11], the threshold for Karatsuba multiplier has been chosen at $\tau = 29$ for this $m = 233$-bit multiplier, as this threshold gives the best performance.

After performing analysis on multiplier, in the next subsection delay analysis of inverse module will be performed.

3.4.2 Optimization of the Inverse Module

Projective to affine conversion is necessary to generate the final result after the scalar point multiplication. Itoh-Tsujii algorithm is one of the known algorithms for calculation of multiplicative inverse and it works on the basis of Fermat's little theorem as shown in the following equation:

$$a^{-1} = a^{2^m - 2}. \tag{6}$$

The recursive relation for calculating multiplicative inverse in [12] is:

$$\beta_{k+j}(a) = {\beta_j}^{2^k} \beta_k = {\beta_k}^{2^j} \beta_j \tag{7}$$

where $\beta_{k+j}(a) \in GF(2^m)$ and $\beta_k(a) = \beta_k$. In [11] it is clearly shown that Multiplicative inverse of $a \in GF(2^{233})$ can be expressed as, $a^{-1} = (\alpha_{232}(a))^2$. Hence, it is clear that a field multiplier block is necessary for the implementation

of the inversion algorithm. From Figure 6 again it is observed that the multiplication and inversion operations are not executed in parallel, hence for the optimized design, same field multiplier is reused in inversion.

Finally, input of a quad block (qin) raises the input to its power of n, for powerblock [11] of size n. Cascaded blocks can raise the input to the power of $n^2, n^3, ...n^{14}$ etc. Then, the quad selection line ($qsel$) controls which of the raised inputs will pass to the output (qin^{qsel}) with the formation of addition chain. Selecting the proper length of addition chain, number of clock pulses can be minimized [2].

Delay calculation for the inversion block: The inversion block typically consists of multiplexer and power block. The power block is basically the collection of u_s number of cascaded 2^n circuits. Let the total delay of the quad block is D_{Invblk}, which is dependent on power block delay(D_{2^n}) and multiplexer delay (D_{Mux}).

$$D_{Invblk} = u_s D_{2^n} + D_{Mux}. \tag{8}$$

Further, delay of the hardware largely depends on LUTs present in the design. Hence, in the next subsection we shall discuss about how the number of LUTs present can be theoretically estimated from the design parameters and how the LUTs are attributing to the delay of the design. As explained in [13], the delay D_{2^n} is computed based on the LUT requirement of the 2^n circuit.

LUT Requirement Analysis

Performance of any design on an FPGA domain largely depends on the physical device delays of the FPGA components. Look up table (LUT) based FPGA is a popular architecture in which the basic programmable logic block is a k-input LUT, which can implement any Boolean function of up to k-variables and is largely proportional to the delay.

The total number of LUTs required to implement an equation depends on the number of variables present in the equation, i.e a k-input LUT can handle a function of k variable. The total number of k input LUTs for a function with x variables can be approximated as [13],

$$lut(x) = \begin{cases} 0 & \text{if } x \leq 1; \\ 1 & \text{if } 1 \leq x \leq k; \\ \lfloor \frac{x-k}{k-1} \rfloor + 2 & \text{if } x > k \text{ and } (k-1) \nmid (x-k)\ ; \\ \frac{x-k}{k-1+2} & \text{if } x > k \text{ and } (k-1) | (x-k). \end{cases}$$

As explained in [13], the delay based on LUT requirement is $\lceil log_k x \rceil$. In subsequent sections, we shall discuss how this delay affects overall processor performance.

LUT Analysis of Inverse Module

The output of this power circuit is $d = A^n.a$ and A is $m * m$ square matrix representation in $GF(2^m)$ and a is the m-bit input to the power circuit. Any bit of d, d_i can be computed as the XOR of elements present in a and the LUT requirement is computed as:

$$LUT_{2^n} = \sum_{0}^{n-1} lut(d_i) \tag{9}$$

and the delay is computed as:

$$D_{2^n} = Max(LUT\ delay\ of\ d_i)\ \ (0 \leq i \leq (m-1)). \tag{10}$$

So the entire delay of the power block is:

$$D_{Invblk} = u_s D_{2^n} + \lfloor log_k(u_s + log_2 u_s) \rfloor. \tag{11}$$

From Equation (11), it is clear that critical parameters for this equation are the cascade number (u_s) and the power of the power block (n), hence the design is tunable on the basis of these two parameters.

3.5 Final Delay Computation of the Overall Processor

As explained in Section 3.4.2, LUT requirement and LUT based delay are dependent on the variables handled.

The total LUT requirement of the total design is the summation of the LUT requirements for the submodules. Specifically, for this BEC processor,

$$\begin{aligned} \text{Total LUT} &= LUT_{multiplier} + LUT_{Invblock} \\ &\quad + LUT_{multiplexers}. \end{aligned} \tag{12}$$

For a m-bit hybrid Karatsuba multiplier, the total LUT requirement is [13]:

$$\begin{aligned} LUT_{hkmul}(m) &= 2LUT_{hkmul}(\lfloor m/2 \rfloor + LUT_{hkmul} \\ &\quad (\lceil m/2 + (2m-1) \rceil). \end{aligned}$$

From [11], the threshold $\tau = 29$ is experimentally determined. Below this threshold, genaral Karatsuba multipliers and above this threshold, school book multipliers give the better performance, as shown in the Karatsuba multiplier tree in Figure 5.

The LUT requirement for the schoolbook multiplier is:

$$LUT_{sbmul} = 2\sum_{i=1}^{\tau-1} lut(2i) + lut(2\tau).$$

For any multiplexer with s-selection lines, LUT requirement for handling $(2^s + s)$ variable is $lut(2^s + s)$. Further, for m-bit variable, the total number LUTs are:

$$LUT_{MUX} = m * lut(2^s + s).$$

Finally, the total LUTs for multiplexers in the ALU are:

$$\begin{aligned} LUT_{AluMux} &= LUT_{MuxA} + LUT_{MuxB} + LUT_{MuxC} \\ &\quad + LUT_{MuxD} + LUT_{MuxOut}. \end{aligned} \tag{13}$$

For the rest of the processor:

$$\begin{aligned} LUT_{OtherMUX} &= LUT_{MC0} + LUT_{MC1} + LUT_{M01} \\ &\quad + LUT_{M02} + LUT_{M03}. \end{aligned} \tag{14}$$

From Equation (13), *MUXA* and *MUXB* are with 3 selection lines, *MUXC* is with 2 selection lines and 4-inputs and *MUXD* and *MUXOut* are with 2-inputs and single selection line. In Equation (14), *MC0* and *MC1* are of 2 inputs and the rest are of 4 input lines. The effective LUT requirement is computed with the above equations.

The final delay computation is associated with the delay of multiplexers, ALU and power block. Any $2^s * 1$ MUX can be represented as a function of $2^s + 1$ variables.

$$D_{MUX} = maxlutpath(2^s + s).$$

Now, considering $k = 4$ for 4-input LUT design (as the platform of the design is $Virtex4$ with 4-input LUT),

$$\begin{aligned} D_{MUXA} &= log_4(2^3 + 3) \\ D_{Reduction} &= log_4(2) = 0.5 \\ D_{MUXC} &= log_4(2^1 + 1). \end{aligned}$$

All the above delays are summed up with the delays of multiplier and inversion block. Final delay can be computed as,

$$D = 10.895 + (D_{2^n} * u_s + log_4(u_s + log_2 u_s)). \tag{15}$$

3.5.1 Clock Cycle Requirement Analysis

The main challenge of modelling such architecture is to consider both the clock-cycle and the area requirement as the design parameters. An ideal design should have minimum clock-cycle requirement and also lower area. But, both the requirements are contradictory, as the clock pulse minimization requires higher hardware support which eventually increases the area. For this design, fast multiplication is prioritized in this design and area is optimized

as far as possible. If the total length of the key is l and the Hamming distance is h, then the total number of clock pulses required is:

$$\begin{aligned} \text{Number of clock pulses} &= 3 + 25(l-1) + 25(h-1) \\ &\quad + \text{inversion clocks.} \end{aligned} \tag{16}$$

From the scheduling criteria mentioned in Table 3, 3 clock pulses are required for initialization and rest for the steps of unified addition algorithm. The inversion clocks are mainly dependent on the structure of inversion module, the power(n) and the number of cascades(u_s). With increase of n and u_s, it is expected that the clock cycle requirement will decrease, but the LUT requirement will increase simultaneously. Hence, for the ideal performance there should be a balance between these two parameters.

Table 5: Comparison of area-delay product for various cascades

Powerblock	Cascade Number	LUT	LUT Delay	Clock-cycle	Performance [Normalized]
2	17	34728	18.39	7268	0.025
3	1	36885	14.99	7342	0.028

Another important issue in case of considering the total clock-cycle is that, we have considered half of the bits in 233-bit key as 1 and no consecutive ones are considered to get the advantage of ternary representation as mentioned in [2]. But, the initial 233 clock pulses required for converting the 233-bit key to its ternary form are taken into consideration. Hence, in actual scenario as the number of consecutive ones increase in the key, there will be always a probability of decreasing total number of required clock pulses and the design will be faster depending on the choice of key.

4 Analysis of the Result

So far we have discussed about the tunable parameters and how they are affecting the delay and the timing requirements. In this section, we analyze how this tuning can improve the overall performance. By performance, we mean a well defined parameter and it is defined as the following relation:

$$\text{Performance} = \frac{1}{LUT * Delay * ClockCycles}.$$

Next, we have obtained the LUT requirement, delay and the clock cycle for different combinations of n and u_s based on the Equations (12), (15). Further, the parameter Performance is computed with these parameters for different n and u_s. Among these values, the design with quad circuit of 17 cascades requires minimum LUT but octet circuit with single cascade requires least number of clock cycles. These two critical observations are listed in Table 5. Further, design with octet gives the better performance compared to other designs. Hence, the design is actually implemented in hardware in Virtex4 FPGA platform. Total area requirement of this design in terms of gate count is $255,523$ and 4 input LUT requirement is 37034. The design is synthesized at 73MHZ frequency and total time requirement is $.12ms$.

4.1 Comparison with Previous Works

In this section, we compare the optimized design with some existing BEC processors. According to present literature, the known implementations are in [2, 8, 9]. But, the first two designs are ASIC implementations of BEC and both are optimized for low area consumption. On the contrary, our design is for FPGA platform and comparatively larger number of registers are used to make the design faster. In the previous two implementations, the cost of sequential multiplication and addition is digit size dependent and it is $163/d$, where d is the digit size. In FPGA designs, parallel implementation of addition and multiplication in every clock cycle is the main reason behind the increase of speed.

In the Table 6, a comparative study is given with the design explained in [9] and [8]. But, the most important comparison is in between the design explained in [2] and the work explained in this paper as both are in the same FPGA platform.

For better comparison, a scaling factor is multiplied with the slice and required time as mentioned in [11], The number of slices hold a quadratic relationship, hence the factor is considered as $(233/m)^2$ and the scaling factor for time is $233/m$. Since, the implementations mentioned in [9] and [8], both are in ASIC(.13μ technology) and the working frequencies are also largely different, we have considered the area time product as a measure of comparison. Table 6 shows this optimized design gives a better area-time product compared to the FPGA implementation in paper [2] also.

Table 6: Comparison of BEC processor implementations in ASIC and FPGA platform

Work	Platform	Field [m]	Area [Gate count]	Scaled area	Frequency [HZ]	Time [ms]	Scaled Time	Area-Time Product [GigaUnit]
Design in [9]	ASIC	163	13,427	27254	400K	149.50	213.79	5.8
Design in [8]	ASIC	163	14,992	30229	5M	101.00	144.37	4.3
Work in [2]	FPGA	233	240,064	240,064	47.4M	.19	.19	0.04
This work	FPGA	233	255523	255523	73.0M	.12	.12	0.03

5 Power Profile Analysis

As mentioned earlier, the BEC processor is based on unified addition and doubling laws. In general, irregular, key bit dependent, faster scalar multiplication algorithms like *Double and Add algorithm* are prone to side-channel attacks (SCA) as the addition and doubling operations should be handled separately. But, the scalar multiplication operation of BEC processor is expected to be simple side channel attack preventive due to this unified nature. In this section, we first explain the FSM of the design to describe the key bit dependant operations and then the detailed analysis of the power profiles.

5.1 FSM of the Design

The functionality of the processor is explained based on the FSM as shown in Figure 6. Initial 3 states (from $Init1$ to $Init3$) are for initialization. In these states, values of X-coordinates and Y-coordinates of P are loaded from ROM to register. Next, state $Add1$ to state $Add25$ are for the unified addition (or doubling)according to projective addition formula, which requires 21 general multiplications and 4 multiplications with field parameters d_1 and d_2. Since BEC is strongly unified, same states are used for both addition and doubling. The addition and doubling is done on the basis of most significant bit(MSB) and $(MSB-1)$ of the key. If key $[MSB:MSB-1]$ is 00, then only doubling is required. If key $[MSB:MSB-1]=01$ or 11, addition is required along with doubling. In case of key $[MSB:MSB-1]=11$, as it stands for -1, (x,y) is required to be replaced by $-(x,y)$. According to the property of the curve, $-(x,y)$ equivalent to (y,x). This change is also incorporated in FSM as well as the design. From the FSM, it is clear that same steps of operations are executed for both addition and doubling. This will reduce the chance of power leakage from key-bit dependant operations.

5.2 Analysis on Obtained Power Traces

In [3], we have explained the detailed method of power profile analysis for BEC processor. Here, we apply the same method on the modelled BEC processor to check if it is truly side channel attack preventive. As mentioned in [3], we add the modified key modification module, which has already proven to be simple side channel attack preventive. Further, we applied different keys like key with all one bits, key with all zero bits, key with alternative all zero and all one and finally different random keys. For each of these cases, the obtained power profiles are almost identical and do not reveal any information regarding the key bits.

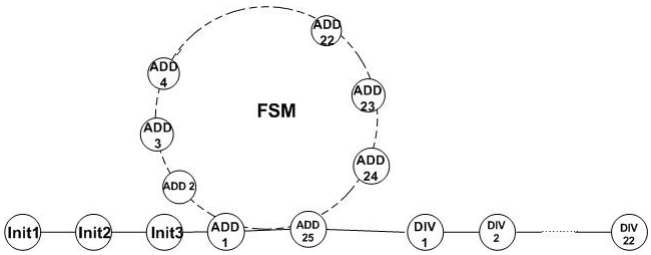

Figure 6: FSM of the design

Figure 7: Power profile for the chosen key

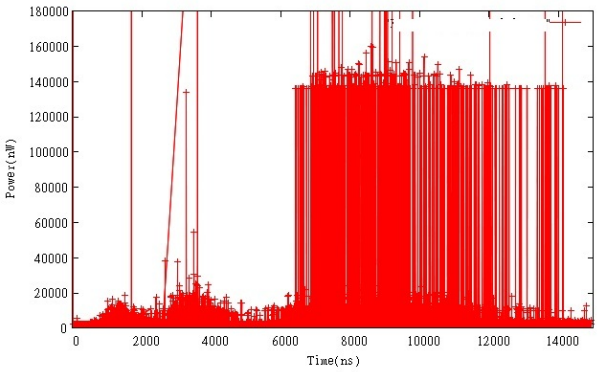

Figure 8: Power profile for the all one bit key

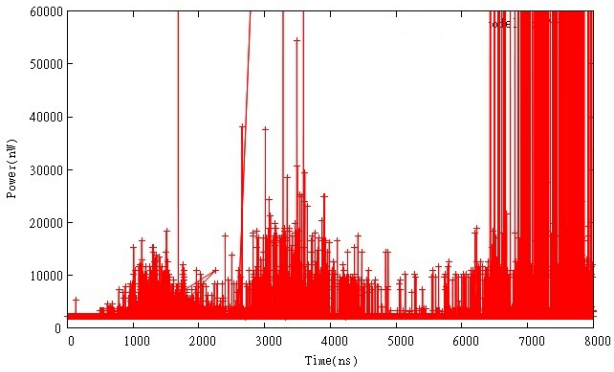

Figure 7 shows the power profile with a random key resembles with Figure 8 of power profile with key of all one bits. The similarity proves that the modified processor is truly simple side channel attack preventive, since there is no leakage of key related information from the power profiles.

6 Conclusion

In this paper, we have made an effort to model the FPGA implementation of BEC processor to obtain optimized parameters. Initially, the tunable parameters are identified and optimized to obtain the final design. With proper power analysis, we have shown that the modelled processor is truly simple side channel attack preventive. According to our synthesis result, this design is faster and more generalized approach compared to the previous ASIC and FPGA implementations of BEC. Further, such optimization approaches help the designer to directly obtain the optimized design in terms of area and delay, rather than working with any arbitrary parameters.

References

[1] D. J. Bernstein, T. Lange, and R. R. Farashahi, "Binary edwards curves," Cryptology ePrint Archive, Report 2008/171, 2008. (http://eprint.iacr.org/).

[2] A. Chatterjee and I. Sengupta, "Fpga implementation of binary edwards curve using ternary representation," in *Proceedings of the 21st ACM Great Lakes symposium on VLSI (GLSVLSI'11)*, Lausanne, Switzerland, 2011.

[3] A. Chatterjee and I. Sengupta, "Design of a high performance binary edwards curve based processor secured against side channel analysis," *Integration*, vol. 45, no. 3, pp. 331–340, 2012.

[4] H. M. Edwards, "A normal form for elliptic curves," in *Bulletin of the American Mathematical Society*, pp. 393–422, 2007.

[5] J. Hoffstein, *An Introduction to Mathematical Cryptography*, Springer, 2009.

[6] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Physics Doklady*, vol. 7, pp. 595, Jan. 1963.

[7] K. H. Kim, C. O. Lee, and C. Nègre, "Binary edwards curves revisited," in *Proceedings of 15th International Conference on Cryptology in India (INDOCRYPT'14)*, pp. 393–408, New Delhi, Dec. 2014.

[8] U. Kocabas, L. Batina, and I. Verbauwhede, *Hardware Implementations of ECC over a Bonary Edwards Curve*. Taiwan: Master Thesis of Katholieke Universiteit, Leuven, 2009.

[9] U. Kocabas, J. Fan, and I. Verbauwhede, "Implementation of binary edwards curves for very-constrained devices," in *21st IEEE International Conference on Application-specific Systems Architectures and Processors (ASAP'10)*, pp. 185 –191, Rennes, France, July 2010.

[10] A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone, and R. L. Rivest, "Handbook of Applied Cryptography," 1997.

[11] C. Rebeiro and D. Mukhopadhyay, "High speed compact elliptic curve cryptoprocessor for fpga platforms," in *Proceedings of the 9th International Conference on Cryptology in India: Progress in Cryptology (INDOCRYPT'08)*, pp. 376–388, Berlin, Heidelberg, 2008.

[12] F. Rodríguez-Henríquez, G. Morales-Luna, N. A. Saqib, and N. Cruz-Cortés, "Parallel ITOH–TSUJII multiplicative inversion algorithm for a special class of trinomials," *Design Codes and Cryptography*, vol. 45, pp. 19–37, Oct. 2007.

[13] S. S. Roy, C. Rebeiro, and D. Mukhopadhyay, "Theoretical modeling of the ITOH–TSUJII inversion algorithm for enhanced performance on k-lut based fpgas," in *DATE*, pp. 1231–1236, 2011.

Ayantika Chatterjee is an ongoing PhD student in School of Information Technology, IIT Kharagpur. Her research interest is Cryptography and VLSI.

Indranil SenGupta presently working as Professor in the Department of Computer Science and Engineering, and the Managing Director of Science and Technology Entrepreneurs Park (STEP, IIT Kharagpur, India. Backed by teaching and research experience of 26 years, research interests include cryptography and network security, VLSI design and testing, and reversible/quantum computing. Previously the Heads of the Department of Computer Science and Engineering and also the School of Information Technology, IIT Kharagpur, Prof. Sengupta has several research contributions in different international jounals ans conferences.

A Survey of Mobility Models of Vehicular Adhoc Networks and Simulators

Thangakumar Jeyaprakash[1], Rajeswari Mukesh[2]
(Corresponding author: Thangakumar Jeyaprakash)

Department of Computer Science and Engineering, Hindustan University[1]
No.1, RajivGandhi Salai, Padur, Chennai, Tamilnadu, India
Department of Computer Science and Engineering, Hindustan University[2]
No.1, RajivGandhi Salai, Padur, Chennai, Tamilnadu, India
(tkumar@hindustanuniv.ac.in)
(Received July. 28, 2014; revised and accepted Dec. 13, 2014)

Abstract

The survey and analysis of Mobility Models and Network Simulators for Vehicular Adhoc Networks (VANET) is still hard to endure for the researcher's community over decades. This paper will provide an in-depth survey for the researchers doing research in Mobility Models and Simulators of Vehicular Adhoc Networks. The purpose of the paper is to describe the various Mobility Models suitable for VANET and the Network simulators to simulate the Vehicular Adhoc Networks which will be worthwhile to propose a new mobility model of Vehicular Adhoc Networks. In this paper, we have proposed a new Mobility Model suitable for the developing countries with the features of existing mobility Model of VANET.

Keywords: Mobility Models; Network Simulators; Vehicular Adhoc Networks (VANET).

1 Introduction

The main theme of this research survey is to generate a structure of a vehicular mobility model based on the accessible research survey by the government to provide a comfort to passengers. For example, the Ministry of Road Transport and highways, India, had been published a survey on road accidents and causes of Road accidents. The causes of road accidents are pedestrian fault, Defect in the condition of motor Cycle, Road damage, Weather condition, Cyclist fault, Drivers fault etc. In Figure 1, Based upon the survey taken by Ministry of Road Transports and Highways, The road accidents caused due to drivers fault is 79 percent approximately. A VANET is a wireless network which is a particular form of MANET. It does not have a strict restriction in power, processing and storage capacities like MANET [4]. It can access to the fixed infrastructure is always possible as Roadside station (RSS). Each vehicle moves according to a Road Network Pattern [3] and not at random like MANET. Several Routing protocols originally defined for MANETs have been adapted to VANETs such as AODV, DSR, DSDV, TORA etc.

In future, we can expect that vehicles will be equipped with the maximum number of wireless devices. It affords safety and comfort to passengers. It does not rely on any central administration for providing communication. The practical applications of VANET are commercial, access to Internet, notification, etc. Communication will be provided by On-Board Division (OBD) in built in the Vehicle and Road-Side Station (RSS) Infrastructure. These Components [15] of On-Board Unit are Event Data Recorder (EDR), Unique Identification Number (UIN), Global Positioning System (GPS), Sensors to detect obstacles and Special Devices used to provide Communication.

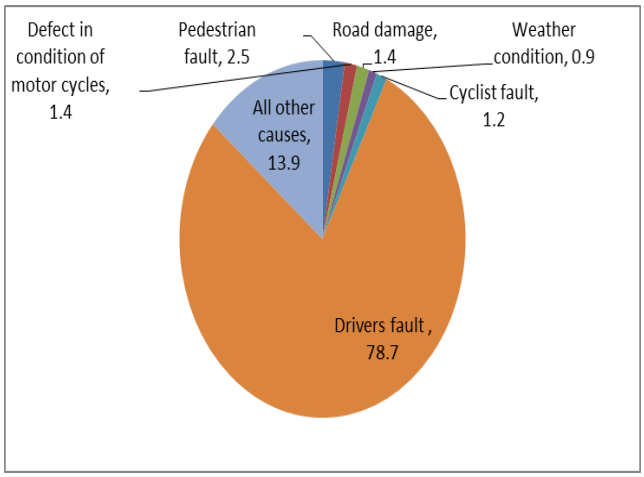

Figure 1 : Causes of Road accidents 2012, published by Ministry of Road Transport and Highways, India

2 Related Works

Most of the researchers simulate the VANET Mobility Models in simulators with some desirable parameters of the vehicle like speed, time, distance etc.

Stefan Krau et al proposed a Krauss model [11] for car tracking represent the variations of the velocity, the minimum stopping distance has to be maintained by the drivers regard to the vehicle that precedes them to avoid shocks or collision.

Peter Wagner proposed a Wagner Model [17] with the purpose of introducing the two characteristics of the human driving. 1) The driver usually plans the future event while they drive. 2) The people use the type of control exercise over their vehicle while they drive, i.e. discrete in a time, act in certain times according to driver situation.

Kerner defines the three phase traffic theory [10] which divides the vehicular traffic in three phases. 1. Free flowing: In this state, the vehicles will be moving without any congestion traffic. 2. Synchronized flow: Vehicles will be circulating with the synchronized speed. 3. Wide congestion: In this state, the vehicles will be moved with almost wide congestion and the velocity is almost zero.

Treiber et al. proposed an Intelligent Driving Model [16] (IDM) in which the acceleration of the vehicle depends upon the factors like own acceleration, the acceleration of the surrounding vehicles, the distance towards the precedent vehicles apart from its own acceleration.

Maldonado et al. compared the routing protocols [12] for vehicular ad hoc networks determined by the Network Routing Load. Abbas and Khader [1] proposed a Dynamic Transition Mobility Model for VANET to describe the movement pattern of vehicles and data transmission between vehicles using NCTUns5.0 tool.

Davis proposed a City Section Mobility Model [5] in which the roads are considered to be bi-directional and single-lane roads. The security distance will be maintained between the vehicles. This model introduces the principles of Random Way-Point Model [14] that the vehicles select the random destination over the grid and move towards it with the constant speed to reach the destination. Once its reach the destination, it repeats the process.

Harri et al. [7] described a Framework for Realistic Vehicular Mobility Models with certain building blocks. He states the framework of VANET had been macroscopically and microscopically [8]. The inter-distance between the car, overtaking, braking due to an obstacles, indication of braking etc. revealed the microscopic level. The traffic densities and the speed flows in the network stated the macroscopic level. He states the framework with four main blocks. They are Motion constraints, Traffic Generator, Time patterns and External Influences. The motion constraint defines the parameters of the vehicles in mobility. The motion constraints of the vehicles are neighbor vehicles, pedestrians, obstacles, speed- breakers, streets, buildings etc. The traffic generator discussed the inter-distance between the car, overtaking, braking due to an obstacle, indication of braking, breaking due to a sudden shocks or collision etc. The time pattern defines a various mobility of the vehicles at various times. The last block explains the external influences such as the communication between the vehicles, communication protocols etc.

To generate the Vehicular Mobility Models, these four ultimate blocks is necessary for the researchers. Certainly, the parameters of the blocks can be rehabilitated affording to different places i.e. where the suggested mobility model works. The objective of the paper is to find the traffic solution for the developing countries. Hence, to achieve this, the

parameters of the blocks must be chosen accordingly.

Harri et al. framework [7] had been extended with additional external influences which should be realistic for developing countries. A mobility model should be constructed with the building blocks like motion constraints, Time patterns, External influences, and the traffic generator.

The following specifics had been considered in generation of Mobility Model [7].

Topological Maps: The traces of streets, Roads or several real data are acquired from the topological maps.

Obstacles: The obstacles should be involved in the motion constraints as parameters. The neighbor vehicles, streets, buildings, speed breakers, trees, traffic signals are considered to be an obstacle.

Attraction points: Every vehicle has a source and the destination points. The destination points are called attraction points if the destination point is same always for a particular vehicle. The office is an attraction point.

Repulsion Points: Every vehicle has a source and the destination points. The Source points are called repulsion points. The home is a repulsion point.

Multilane: The road can be a multilane.

Speed Constraints: The speed constraints are the vehicles speed, behavior or speed of the neighbor vehicle etc.

3 Comparisons of Mobility Models

In Table 1, the assessment of Mobility Models based on the references evidently revealed to choose the traffic simulator based mobility model with the less complexity. Researchers can able to generate the Mobility Models with Traffic-based simulator models. The simulation of these Mobility Models is assumed to be valid. Researchers can compare the movement pattern generated by a traffic simulator is same as like the other one. For e.g., the traces created by a CORSIM traffic simulator should be identical as like CORSIM simulator. These trace file can be generated from the graphical softwares like NSG2 and can be used by the Network simulator. The Mobility Models has been categorized into 4 types:

1) Synthetic Models [6];
2) Survey–based Models;
3) Real Time Trace–based;
4) Traffic – Simulator Based.

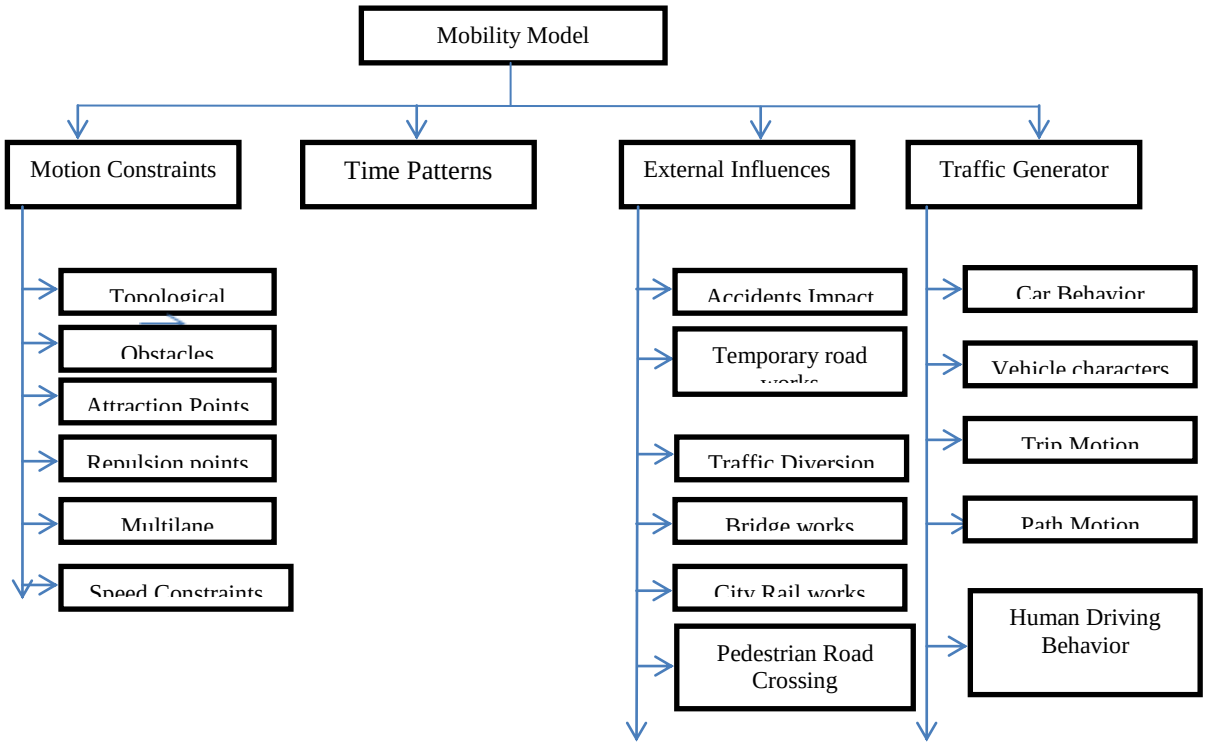

Figure 2: Structure of mobility model with additional external Influences [7]

Table 1: Comparison of mobility models [7, 8, and 3]

Model / Characteristics	Types	Examples
Synthetic Models [15]/ Mathematical based	Stochastic Mobility Models – random flow of vehicles [7] Traffic stream models – It is a hydrodynamic phenomenon Car following Model – movement patterns based on vehicles ahead Queue Models – First in First out pattern. Behavior model –Based on Social influences	Random Waypoint Mobility Model [13] Weighted Waypoint Mobility Model [9]
Survey –based Models/ Survey – based	UDeL Mobility Model [18] – movement pattern generation based on the survey provided by U.S department of Labor. Agenda –based Mobility Model [18] – Based on vehicular traffic statistics	UDel Mobility Model Agenda-based Mobility Model
Real Time Trace – based	Wheels Project – for Highway section Dieselnet – for bus system Cabspotting project – for call taxi movement patterns	CrawDAD UMASSDieselnet Cabspotting MIT Reality Mining Trace –based Models/ USC MobiLib
Traffic – Simulator Based	PARAMICS CORSIM PARAMICS Traffic simulator models delivers a realistic demonstration of the "friction" to traffic flow instigated by pedestrians CORSIM used to simulate larger traffic networks	VISSIM TRANSSIMS SUMO [2] NS2 **Advantages;** Validated Traces Not more complex The purchase of license is available for the use of Traffic simulators

4 Proposed Mobility Model with Additional External Influences for Developing Countries

In Figure 2, the road traces or the street traces should be constructed with the external influences which could be works in developing countries, the traffic pattern generator should be created in the mobility model with the parameters like car behavior and human driving patterns [7].

5 Mobility Model And Network Simulator

The generated Mobility Models should be implemented in three approaches. 1) Isolated Mobility model; 2) Embedded Mobility Model; 3) Federated Mobility Model.

5.1 Isolated Mobility Model

In Figure 4, The Vehicular traces generated by a traffic simulator should be feed into the Network simulator for the communication implementation is known as Isolated Mobility Model. In Figure 3, the vehicular traces generated by the vehicular traffic generator are called Open Street Maps (OSM). These OSM will be embedded into the network simulator for the communication.

Figure 3: Open street map of Chennai city, India generated by OSM Traffic simulator software [7]

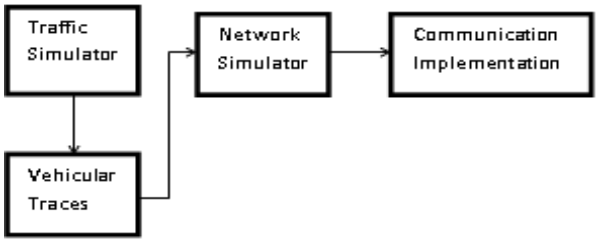

Figure 4: Isolated mobility model and network simulator [7, 13]

5.2 Embedded Mobility Model

This mobility model is differs from isolated, that it is never separated from Network Simulator. Some of the traffic generator embedded with network simulator too. In Figure 5, the traffic simulator is always interacting with the network simulator as an interface.

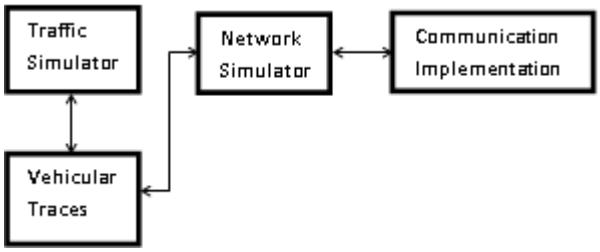

Figure 5: Embedded mobility model and network simulator [7, 13]

Mobility model interacts more with network simulator can use this mobility model. For example, TRANSLite traffic generator deals with SUMO configuration file to generate traces and to perform simulation. MoVes are also an embedded Mobility model generates vehicular traces and also contains the network simulator. Simulation for Urban Mobility is a network traffic simulator suitable for Vehicular Adhoc Networks. It is an Open source designed to handle large road networks. It is licensed under General public License (GPL). It allows to simulate how a given traffic flow consists of vehicles move through a given road network with effective simulation.

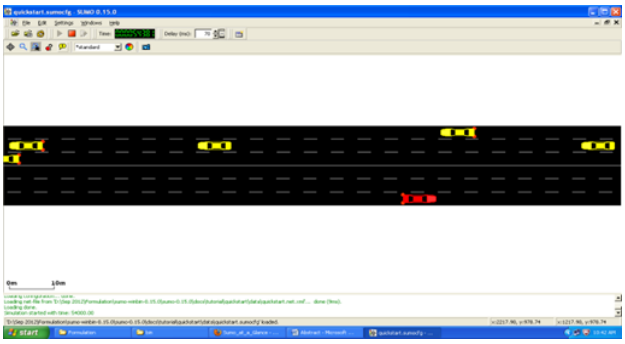

Figure 6: Simulation of urban mobility model [15]

In Figure 6, the SUMO binaries are required to generate and visualize the route. In Figure 7, TRANSLITE traffic simulator is used to generate network files, cropping an existing network, Mobility file generation, Route generation with edges and visualizing the vehicular traces.

5.3 Federated Mobility Model:

The interaction between the two embedded Mobility Models is known as Federated Mobility Model. The CORSIM simulator or VANETMOBISIM interacts with the QUALNET simulator, also known as the Federated Simulations Development Kit (FDK), with the feedback

messages of both network simulators and Vehicular Mobility Models. The traces of Simulator SUMO can be extracted using interpreter and the same will be feed into Network simulator NS2.

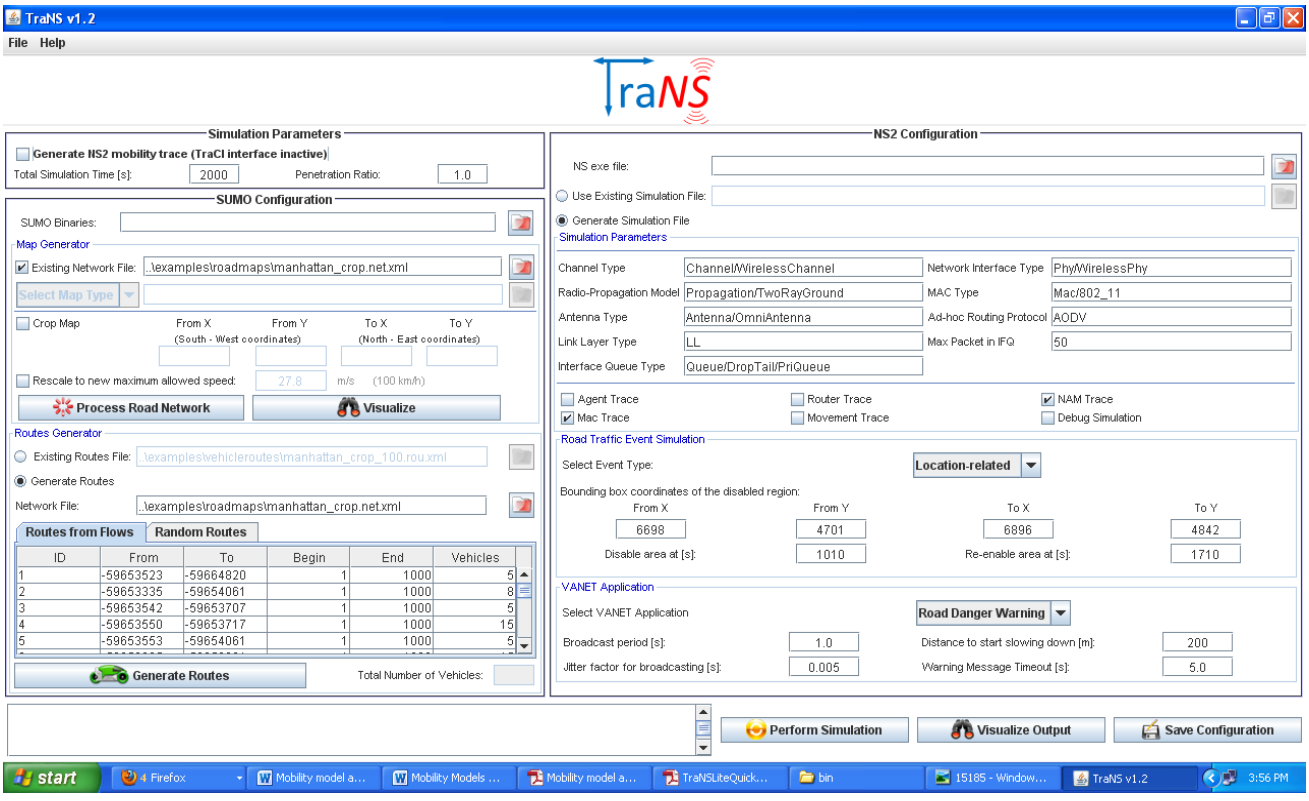

Figure 7: TRANSLite simulator implementation [15, 16]

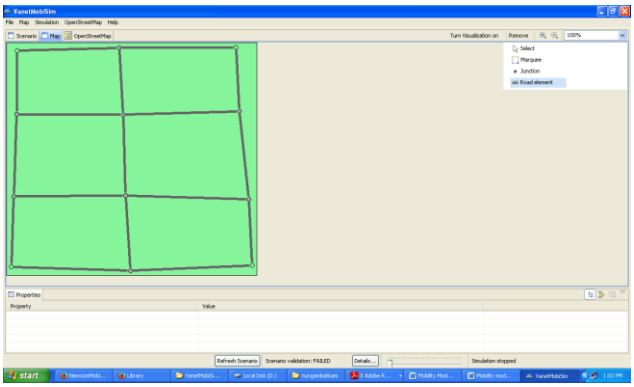

Figure 8: Generation of grid map using VANETMOBISIM simulator Implementation

In Figure 8, the vehicular road network traces of grid map have been generated using VANETMobiSim Network simulator. The generated traces can be loaded for the simulation purpose. VanetMobiSim is an open source, well-organized, large scale, interoperable and configurable traffic simulator for accurate vehicular simulations. It was developed at the Institute of Telematics at the University of Karlsruhe.

6 Conclusion

In this paper, the various Mobility Models such as Synthetic Mobility Models, Survey-based Mobility Models, Trace–based Mobility Models and Traffic simulator based Mobility Models has been compared in different views. The different Network Simulators such as SUMO Simulation of Urban Mobility, VANETMOBISIM, and TRANSLite has been analyzed and implemented. Thus the survey helps us to propose and simulate a mobility model of VANET with an enhanced Routing protocol using NS2.

References

[1] M. Abbas, Sheik Abdul Khader, ”Comparative analysis of packet delivery in vehicular adhoc networks”, *Journal of Computational Information Systems*, vol. 8, no. 1, pp 343 – 353, 2012.

[2] L. Bononi, M. Di Felice, M. Bertini, and E. Croci, ”Parallel and distributed simulation of wireless vehicular ad hoc networks”, *in Proc. ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, Torresmolinos, Spain, 2006.

[3] M. Brackstone and M. McDonald, ”Car-following: A historical review”, *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 2, no. 4, pp. 181-196, 1999.

[4] P. Caballero, *Security Issues in Vehicular Ad Hoc Networks(Ed.)*, University of La Laguna, Spain: Intech, 2011.

[5] V. Davies, ”Evaluating mobility models with in an adhoc networks”, *Colorado School of Mines, Colorado, USA, Tech, Rep, Master's Thesis*, 2000.

[6] M. Fiore, ”Vehicular mobility and network simulation”, *Handbook on Vehicular Networks, S. Olariu and M. C. Weigle Eds.*, Taylor and Francis, 2008.

[7] J. Harri, J. Filali, F. Bonnet, C, "Mobility models for vehicular ad hoc networks: A survey and taxonomy", *IEEE Communications Surveys & Tutorials*, vol. 11, no. 4, pp. 19-41, 2009.

[8] D. Helbing, ”Traffic and related self-driven many-particles systems”, *Rev. Modern Physics*, vol. 73, pp. 1067-1141, 2001.

[9] W. Hsu, K. Merchant, H. Shu, C. Hsu, and A. Helmy, ”Weighted waypoint mobility model and its impact on ad hoc networks*”, in Proc. ACM Mobile Computer Communications Review (MC2R),* pp. 59-63, Jan. 2005.

[10] B. S. Kerner, “The physics of traffic: empirical freeway pattern features, engineering applications and theory", *Three-phase traffic theory* , 2002.

[11] S. Krau, P. Wagner & C. Gawron,”Metastable states in a microscopic model of traffic flow”, *Physical Review E*, vol. 55, pp. 5597, May 1997.

[12] V. Maldonado, Manuel Quinones, Katty Rohoden and Rommel Torres, "Comparison of routing protocols and mobility models for vehicular ad-hoc networks using real maps", *International Journal of Computer Applications*, vol. 15, pp.8-15, Dec. 2012.

[13] A. Rojas, P. Branch, and G. Armitage, "Experimental validation of the random waypoint mobility model through a real world mobility trace for large geographical areas", *in Proc. 8th ACM international symposiumon Modeling, analysis and simulation of wireless and mobile systems,* Montreal, Canada, pp. 174-177, Oct. 2005.

[14] A. K. Saha, David B. Johnson, "Modeling mobility for vehicular ad hoc networks", *1st ACM workshop on Vehicular Adhoc Networks (VANET 2004),* pp. 91-92, Oct. 2004.

[15] J. Thangakumar, Rajeswari Mukesh, "A tactical information management system for unmanned vehicles using vehicular adhoc networks," in *Fourth International Conference of Intelligent, Modeling and Simulation,* pp.472-474, Bangkok, Jan. 2013.

[16] M. Treiber, A. Hennecke and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations", *Physical Review E*, vol. 62, no. 2, pp. 1805, Aug. 2000.

[17] P. Wagner, "How human drivers control their vehicle", *The European Physical Journal B - Condensed Matter and Complex Systems*, vol. 52, no. 3, pp. 427-431, Aug. 2006.

[18] Q. Zheng, X. Hong, and J. Liu, "An agenda-based mobility model" in *Proc. 39th IEEE Annual Simulation Symposium (ANSS-39-2006),* pp. 188-195, Huntsville AL, Apr. 2006.

Thangakumar Jeyaprakash has received his B.E Degree in Electrical & Electronics Engineering from Dr.Sivanthi Aditanar College of Engineering, Tamilnadu in 2003; He obtained his M.Tech in Computer Science & Engineering, SRM University, Chennai. He is presently pursuing his Ph.D. at Hindustan Institute of Technology & Science, Chennai, and Tamilnadu, India. He has Eight years of industrial, academic and research Experience. He is a member of IEEE, IET. His area of Interests is Mobile Ad hoc Networks, Vehicular Ad hoc Networks, Cryptography & Network Security, Data mining & software Engineering.

Rajeswari Mukesh has received her Ph.D. degree in Computer Science and Engineering from Jawaharlal Nehru University. Hyderabad. At present, she is a Professor and Head of Computer science and Engineering department at Hindustan University. She is guiding 8 PhD candidates. She has published more than 10 international journals and attended more than 15 international and National Conferences. Her area of specialization is Big Data, Biometrics, Adhoc Networks, and Cyber Security. She is a Member of IEEE & IET. She has won the "Women Engineer award" recently from IET.

Online Scheduling of Moldable Jobs with Deadline

Kuo-Chan Huang, Wei Hsieh, Chun-Hao Hung
(Corresponding author: Kuo-Chan Huang)

Department of Computer Science, National Taichung University of Education
No. 140, Min-Shen Road, Taichung, Taiwan
(Email: kchuang@mail.ntcu.edu.tw)
(Received Apr. 21, 2014; revised and accepted July 5, 2014)

Abstract

Parallel job scheduling has long been an important research issue for high-performance computing (HPC). Most modern parallel application programs can be written with a good property called *moldable* which allows parallel programs to have the flexibility of exploiting different parallelisms for execution. Some previous research has developed adaptive processor allocation methods for moldable jobs to improve the overall performance of HPC systems. Recently, the concept of HPC as a Service (HPCaaS) was proposed to bring the traditional HPC field into the era of cloud computing. For HPCaaS systems, scheduling jobs with deadline efficiently becomes a crucial issue. This paper explores the issues of scheduling moldable jobs with deadline, which has not received enough research attention yet. We propose and evaluate three possible dynamic scheduling approaches for online moldable jobs with deadline.

Keywords: High-performance computing; Moldable; Online scheduling.

1. Introduction

High performance computing (HPC) has long been a very important field for solving large-scale and complex scientific and engineering problems. However, accessing and running applications on HPC systems remains tedious, limiting wider adoption and user population [1]. As cloud computing emerges, which emphasizes easier and efficient access to IT infrastructure, recently the concept of *HPC as a Service* [1] was proposed to transform HPC facilities and applications into a more convenient and accessible service model.

To achieve that goal, one of the necessary works is to develop advanced parallel job scheduling methods for improving system efficiency and user satisfaction. Efficient scheduling of *moldable* jobs is a promising direction. *Moldable* [14] is a good property of parallel jobs which allows the jobs to decide the number of processors to use right before their execution. This flexibility provides a great potential for HPC systems to improve their job execution efficiency through appropriate jobs scheduling approaches. Most modern parallel application programs could be elaborately designed to have the *moldable* property. Therefore, moldable job scheduling becomes an important research issue for HPC systems. Some previous research has proposed several adaptive processor allocation methods for dealing with this issue [8, 9, 10, 11].

Scheduling jobs with deadline is an important research topic in many fields. For HPCaaS, submitting jobs with deadline would be an inevitable and desirable user requirement. Job scheduling with deadline has received much research attention in the literature [7, 15, 16]. However, most of the research work deals with rigid jobs [14]. Few attentions have been paid on moldable jobs with deadline. Since moldable jobs would account for an important proportion in modern HPC systems, we explore the issues of efficient online scheduling for moldable jobs with deadline in this paper.

We propose three possible dynamic scheduling approaches for online moldable jobs with deadline. The proposed approaches were evaluated with a series of simulation experiments. The experimental results indicate that careful selection of scheduling approaches is important and different moldable job scheduling approaches can lead to large performance difference.

2. Related works

Parallel job scheduling and allocation has long been an important research topic [3, 4, 13]. For rigid jobs [14], backfilling job scheduling approaches have been proposed to improve system performance [2, 5]. For moldable jobs [14], previous research [11] has shown potential performance improvement achieved by adaptive processor allocation. The proposed adaptive processor allocation methods in [11] dynamically determine the number of processors to allocate just before job execution according to the amount of current available resources and job queue information.

In [8, 9], Srinivasan *et al.* proposed a schedule-time aggressive fair-share strategy for moldable jobs, which adopts a profile-based allocation scheme. This strategy thus needs to have the knowledge of job execution time. Sun *et al.* proposed an adaptive scheduling approach for malleable jobs with periodic processor reallocations based on parallelism feedback of the jobs and allocation policy of the system in [10].

In [1], AbdelBaky *et al.* proposed the concept of HPC as a Service, aiming to transform traditional HPC resources into a more convenient and accessible service. They focused on the issues related to elastic provisioning and dynamic scalability, which are concerned in malleable jobs [14]. In this paper, we focus on the moldable property [14] in most modern parallel applications.

There are many research works in the literature discussing the issues of scheduling jobs with deadline [7, 15, 16]. However, most of the research work deals with rigid jobs. Few attentions have been paid on moldable jobs with deadline. In [6], Saule *et al*. proposed a moldable EDF method to schedule parallel jobs with deadline using the well-known Earliest Deadline First (EDF) heuristic. However, deadline is not the focus in [6] and the moldable EDF method was actually proposed to deal with optimizing the stretch of moldable jobs without deadline. In this paper, we explore the issues of efficient online scheduling for moldable jobs with deadline.

3. Dynamic Scheduling Approaches

In this section we propose three dynamic scheduling approaches for moldable jobs with deadline. The approaches are based on the Earliest Deadline First (EDF) heuristic [6], but differ in how they handle the jobs in queue if the first job cannot meet the deadline with currently available processors. For all the three approaches, when jobs are submitted to the queue, they are sorted in a non-decreasing order of their deadlines. The job scheduler then repeatedly tries to allocate each job according to the sequence in queue. Each time the scheduler gets the first job in queue, which has the earliest deadline at that moment. The scheduler then checks whether the first job can meet its deadline with currently available processors or not. If not, the scheduler can take different actions to deal with the situation, leading to different scheduling approaches. In this section, we investigate three possible approaches for handling such situation.

In the first approach (Algorithm 1), if the first job cannot meet the deadline with currently available processors, it is simply discarded and marked as missed before the scheduler proceeds to the next job in queue. The approach is described in detailed in the following Algorithm 1.

Algorithm 1

```
1: i = the index of 1st job in queue
2: np= number of free processors
3: While(queue is not empty and np>0)
4: {
5:    for(int k=1;k<=np;k++)
6:    {
7:       Calculate the expected finish time of job i with k processors
8:       if(job i meets the deadline)
9:          break;  //break to execute the job
10:       else
11:          Delete job i from the queue
12:    }
13:    i = next job in queue
14: }
```

It is possible that a job cannot meet the deadline under currently available processors, but would become able to meet deadline later because of more available processors due to resource release by currently running jobs. Taking this into consideration, we propose the second dynamic scheduling approach (Algorithm 2) which keeps the first job in queue for later re-checking when it cannot meet the deadline at the moment and proceeds to handle other jobs in queue. The first job will be re-considered in the future scheduling activities when some running jobs finish and release occupied resources. Algorithm 2 in the following describes the approach in details.

Algorithm 2

```
1: i = the index of 1st job in queue
2: np= number free processors
3: While(queue is not empty and np>0)
4: {
5:    for(int k=1;k<=np;k++)
6:    {
7:       Calculate the expected finish time of job i with k processors
8:       if(job i meets the deadline)
9:          break;  //break to execute the job
10:       else
11:          if (the current time >= deadline(i) )
12:             Delete job i from queue
13:    }
14:    i = next job in queue
15: }
```

In the second approach, the scheduler would proceed to handle other jobs in queue when the first job cannot meet the deadline with currently available processors. This arrangement allows out-of-order execution and has the potential to improve resource utilization. However, out-of-order execution runs the risk of raising the probability of missing deadline for the first job. Therefore, in the following we propose the third approach which is a compromise between the first and the second approaches. In this approach, when the first job cannot meet the deadline with currently available processors, the scheduler will check whether future resource releases by current running jobs would allow the job to meet its deadline. If yes, the first job would be kept in queue. However, unlike in the second approach, out-of-order execution is not allowed in this approach in order to ensure that the first job can meet the deadline finally. On the other hand, if the result of the checking is no, the first job would be simply deleted just like in the first approach. In the following, Algorithm 3 describes this approach in details.

Algorithm 3

```
1: i = the index of 1st job in queue
2: np= number of free processors
3: While(queue is not empty and np>0)
4: {
5:    for(int k=1;k<=np;k++)
6:    {
7:       Calculate the expected finish time of job i with k processors
8:       if(job i meets the deadline)
9:          break;  //break to execute the job
10:       else
11:          if (checkEnd(i)==0 )
12:             Delete job i from queue
13:          else
14:             return
15:    }
16: }
17: Procedure checkEnd(i )
```

```
18: {
19:    Sort the running jobs in a non-decreasing order of their expected finish time;
20:    T_finish = the expected finish time of the first running job
21:    While( T_finish < deadline(i) and not all running jobs been checked)
22:    {
23:       np = np + releaseNp
24:       Calculate the expected finish time of job i with np processors
25:       if(job i meets its deadline)
26:          return 1
27:       T_finish = the expected finish time of the next running job
28:    }
29:    return 0  // not found
30: }
```

4. Performance Evaluation

To evaluate and compare the performance of the proposed approaches, we conducted a series of simulation experiments assuming a 128-processor cluster based on a public workload log on SDSC's SP2 [12]. The deadline of each job is given according to the following formula.

$$D(i) = T_{sub}(i) + k * T_{exec}(1,i) \qquad (1)$$

where i is the index of jobs; $T_{sub}(i)$ is the submission time of job i; $T_{exec}(n, i)$ is execution time of job i with n processors. k is a random number picked up within a specified range. We also introduced a parameter, called *load*, which was used to adjust the system loading by multiplying it with the original job runtime in the workload log.

Figure 1 compares the deadline miss rate of the three proposed approaches under different levels of system load. The results indicate that the third approach achieves the best performance and the performance improvement increases as the system load grows. The second approach outperforms the first one when the system load is light, but becomes ineffective as the system load increases.

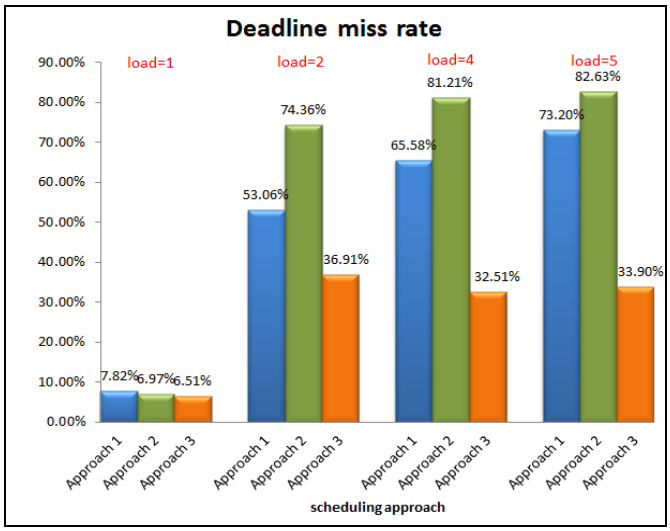

Figure 1. Comparison of deadline miss rate (0 < k <= 1)

Figure 2 compares the three approaches according to another performance criterion: average turnaround time. When the system load is light, the first approach leads to the shortest average turnaround. As the system load increases, the third approach outperforms the other two approaches significantly, similar to the trend in Figure 1.

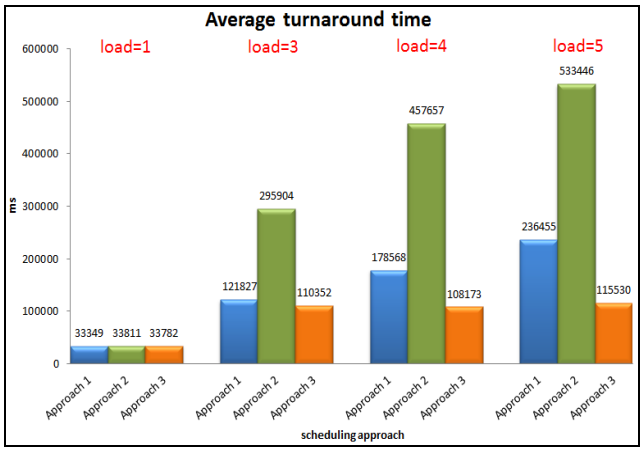

Figure 2. Average turnaround time (0 < k <= 1)

Figure 3 evaluate the proposed approaches with different k values. Smaller k values represent tighter deadline. In the experiments, the parameter *load* was set to 5. The results indicate that the third approach can consistently outperform other approaches significantly under different k values.

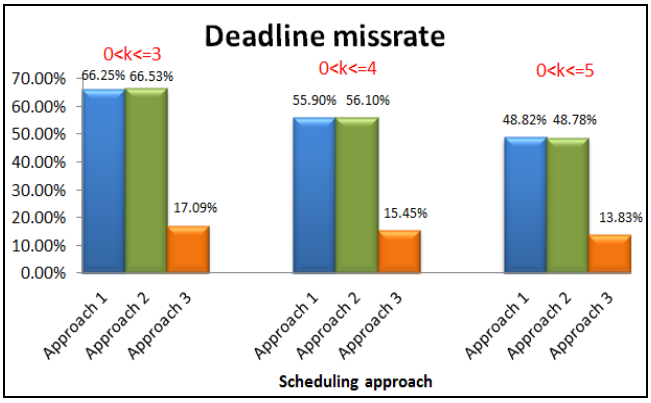

Figure 3. Deadline miss rate with different k values

5. Conclusions

This paper proposes and evaluates three possible dynamic scheduling approaches for online moldable jobs with deadline. This is an important research issue for modern HPCaaS systems, but has not received enough research attention yet. We compared and evaluated the proposed approaches through a series of simulation experiments with a public workload log. The experimental results indicate that careful selection of scheduling approaches is important and different moldable job scheduling approaches can lead to large performance difference. In general, the third approach outperforms other approaches significantly, which means that considering possible resource releases in the future is a promising direction, while out-of-order execution should be avoided although it can raise resource utilization.

References

[1] M. AbdelBaky, M. Parashar, H. Kim, E. J. JordanKirk, V. Sachdeva, J. Sexton, H. Jamjoom, Z. Y. Shae, G. Pencheva, R. Tavakoli, and M. F. Wheeler, "Enabling high performance computing as a service," *IEEE Computer*, vol. 45, pp. 72-80, Oct. 2012.

[2] D. G. Feitelson and A. M. Weil, "Utilization and predictability in scheduling the IBM SP2 with backfilling," in *Proceedings of the First Merged International 12th Int'l Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP'98)*, pp. 542-546, Apr. 1998.

[3] R. Gibbons, "A historical application profiler for use by parallel schedulers," in *Job Scheduling Strategies for Parallel Processing*, pp. 58-77, Springer-Verlag, 1997.

[4] D. Lifka, "The ANL/IBM SP scheduling system," in *Job Scheduling Strategies for Parallel Processing*, pp. 295-303, Springer-Verlag, 1995.

[5] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimate in scheduling the IBM SP2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529-543, June 2001.

[6] E. Saulea, D. Bozdag, U. V. Catalyurek, "Optimizing the stretch of independent tasks on a cluster: from sequential tasks to moldable tasks," *Journal of Parallel and Distributed. Computing*, vol. 72, no. 4, pp. 489-503, 2012.

[7] Q. Perret, G. Charlemagne, S. Sotiriadis, N. Bessis, "A deadline scheduler for jobs in distributed systems," in *27th International Conference on Advanced Information Networking and Applications Workshops*, pp. 757-764, 2013.

[8] S. Srinivasan, S. Krishnamoorthy and P. Sadayappan, "A robust scheduling strategy for moldable scheduling of parallel jobs," in *5th IEEE International Conference on Cluster Computing*, pp. 92-99, 2003.

[9] S. Srinivasan, V. Subramani, R. Kettimuthu, P. Holenarsipur and P. Sadayappan, "Effective selection of partition sizes for moldable scheduling of parallel jobs," in *9th International Conference on High Performance Computing*, Springer, Lecture Notes In Computer Science, Bangalore, India, vol. 2552, pp.174-183, 2002.

[10] H. Sun, Y. Cao and W. J. Hsu, "Efficient adaptive scheduling of multiprocessors with stable parallelism feedback," *IEEE Transactions on Parallel and Distributed System*, vol. 22, no. 4, pp. 594-607, Apr. 2011.

[11] K. C. Huang, "Performance evaluation of adaptive processor allocation policies for moldable parallel batch jobs," in *3th Workshop on Grid Technologies and Applications*, pp. 6-11, 2006.

[12] Parallel Workloads Archive, http://www.cs.huji.ac.il/ labs/parallel/workload/

[13] D. G. Feitelson, "A survey of scheduling in multiprogrammed parallel systems," *Research Report* RC 19790 (87657), IBM T. J. Watson Research Center, Oct. 1994.

[14] D. G. Feitelson, L. Rudolph, U. Schweigelshohn, K. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, Lecture Notes in Computer Science, vol. 1291, pp. 1-34, 1997.

[15] E. Caron, P. K. Chouhan, F. Desprez, "Deadline scheduling with priority for client-server systems on the grid," in *the Fifth IEEE/ACM International Workshop on Grid Computing*, pp. 410-414, 2004.

[16] G. Le, K. Xu, J. Song, "Dynamic resource provisioning and scheduling with deadline constraint in elastic cloud," in *2013 International Conference on Service Science*, pp. 113-117, 2013.

Kuo-Chan Huang received his B.S. and Ph.D. degrees in Computer Science and Information Engineering from National Chiao-Tung University, Taiwan, in 1993 and 1998, respectively. He is currently an Associate Professor in Computer Science Department at National Taichung University of Education, Taiwan. He is a member of ACM and IEEE Computer Society. He has served as workshop chair, publicity chair, and program committee member for several international conferences. His research areas include parallel processing, distributed systems, cluster, grid, and cloud computing, workflow computing, and services computing.

Wei Hsieh received his B.S. and M.S. degrees in 2012 and 2014 respectively from the department of Computer Science at National Taichung University of Education, Taiwan. His main research interests include parallel job scheduling and high performance computing.

Chun-Hao Hung received his B.S. degrees in 2013 from and is currently a graduate student in the department of Computer Science at National Taichung University of Education, Taiwan. His main research interests include parallel job scheduling, high performance computing, cloud and services computing, simulation and modeling.

Guide for Authors
International Journal of Electronics and Information Engineering

International Journal of Electronics and Information Engineering (IJEIE) will be committed to the timely publication of very high-quality, peer-reviewed, original papers that advance the state-of-the art and applications of Electronics and Information Engineering. Topics will include, but not be limited to, the following: Algorithms, Bioinformatics, Computation Theory, AI, Fuzzy Systems, Embedded Systems, VLSI/EDA, Computer Networks, Information Security, Database, Data Mining, and Information Retrieval, Digital Content, Digital Life, and Human Computer Interaction, Image Processing, Computer Graphics, and Multimedia Technologies, Information Literacy, e-Learning, and Social Media, Mobile Computing, Wireless Communications, and Vehicular Technologies, Parallel, Peer-to-peer, Distributed, and Cloud Computing, Semiconductor, Software Engineering and Programming Languages, Telecommunication, etc.

1. Submission Procedure

Authors are strongly encouraged to submit their papers electronically by using online manuscript submission at http://ijeie.jalaxy.com.tw/.

2. General

Articles must be written in good English. Submission of an article implies that the work described has not been published previously, that it is not under consideration for publication elsewhere. It will not be published elsewhere in the same form, in English or in any other language, without the written consent of the Publisher.

2.1 Length Limitation:

All papers should be concisely written and be no longer than 30 double-spaced pages (12-point font, approximately 26 lines/page) including figures.

2.2 Title page

Title page should contain the article title, author(s) names and affiliations, address, an abstract not exceeding 100 words, and a list of three to five keywords.

2.3 Corresponding author

Clearly indicate who is willing to handle correspondence at all stages of refereeing and publication. Ensure that telephone and fax numbers (with country and area code) are provided in addition to the e-mail address and the complete postal address.

2.4 References

References should be listed alphabetically, in the same way as follows:
For a paper in a journal: M. S. Hwang, C. C. Chang, and K. F. Hwang, ``An ElGamal-like cryptosystem for enciphering large messages," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 2, pp. 445--446, 2002.

For a book: Dorothy E. R. Denning, *Cryptography and Data Security*. Massachusetts: Addison-Wesley, 1982.

For a paper in a proceeding: M. S. Hwang, C. C. Lee, and Y. L. Tang, ``Two simple batch verifying multiple digital signatures," in *The Third International Conference on Information and Communication Security (ICICS2001)*, pp. 13--16, Xian, China, 2001.

In text, references should be indicated by [number].

2.5 Author benefits

No page charge is made.

Subscription Information

Individual subscriptions to IJEIE are available at the annual rate of US$ 200.00 or NT 6,000 (Taiwan). The rate is US$1000.00 or NT 30,000 (Taiwan) for institutional subscriptions. Price includes surface postage, packing and handling charges worldwide. Please make your payment payable to "Jalaxy Technique Co., LTD." For detailed information, please refer to http://ijeie.jalaxy.com.tw or Email to ijeieoffice@gmail.com.