PAPER

Accelerating Large-Scale Interconnection Network Simulation by Cellular Automata Concept*

Takashi YOKOTA^{†a)}, Kanemitsu OOTSU[†], and Takeshi OHKAWA[†], *Members*

SUMMARY State-of-the-art parallel systems employ a huge number of computing nodes that are connected by an interconnection network. An interconnection network (ICN) plays an important role in a parallel system, since it is responsible to communication capability. In general, an ICN shows non-linear phenomena in its communication performance, most of them are caused by congestion. Thus, designing a large-scale parallel system requires sufficient discussions through repetitive simulation runs. This causes another problem in simulating large-scale systems within a reasonable cost. This paper shows a promising solution by introducing the cellular automata concept, which is originated in our prior work. Assuming 2D-torus topologies for simplification of discussion, this paper discusses fundamental design of router functions in terms of cellular automata, data structure of packets, alternative modeling of a router function, and miscellaneous optimization. The proposed models have a good affinity to GPGPU technology and, as representative speed-up results, the GPU-based simulator accelerates simulation upto about 1264 times from sequential execution on a single CPU. Furthermore, since the proposed models are applicable in the shared memory model, multithread implementation of the proposed methods achieve about 162 times speed-ups at the maximum.

key words: interconnection networks, simulation, GPGPU, multithreading

1. Introduction

Today's state-of-the-art supercomputers employ a huge number of processors and the scale is growing steadily [2]. Such large-scale parallel systems inherently employ corresponding large-scale interconnection networks. As interconnection networks (ICNs) offer crucial features in performance, power consumption, and economics, thus, we should find appropriate solutions of ICNs to build practically effective systems [3]–[5]. In the exploration processes of ICNs, simulation is one of the most important methodologies in evaluating and comparing new ideas. What we should do is very simple. We implement the new ideas in an appropriate simulator and measure necessary performance metrics such as throughput and latency. In general, we need repetitive evaluations until achieving satisfactory results. Since large-scale simulation is time and resource consuming, it prevents sufficient evaluations.

As an alternative approach of large-scale simulation, we should discuss *extrapolation* from smaller-scale simulation results. We can expect the methodology to some extent

as stated in our prior work [6], [7], however, it never guarantees accuracy. We should consider strong non-linear features in ICN performance and predictability of accurate aggregate behaviors from individual operation of router. Through this discussion, we again return to the large-scale simulation. The inevitable problem of the large-scale simulation is time and resource consuming feature. This paper addresses efficient and practical methods for large-scale ICN simulation to overcome the problem.

The center idea of this paper is the cellular automata (CA) based modeling principle, which has originated from our early work in literature [8], [9]. Physics research has introduced cellular automata into wide range of modeling and simulations [10]. In the simulation applications of CA, the common idea is *self-driven particle*, where each particle acts autonomously based on a certain common rule. Its practical applications include traffic flow, in which a car is modeled as a particle, and fluid dynamics (i.e., liquid and gas flow). We have introduced the idea of self-driven particle into interconnection network study to unveil complicated phenomenon in dense (or jammed) traffic situations under a rough approximation [8], [9]. A simple uni-directional model of network reveals non-linear feature caused by congestion and phase transition phenomena through large-scale simulations [8].

Through the successful results in the simplified model, we have introduced the harvest of the early CA study into practical ICN simulation [11]–[13]. We have extended a simple *box-ball* model [14] to match the ICN simulation and proposed a simple but practical simulation model. Although the model has many advantageous features in accuracy and speed of simulation [11], [12], it restricts practical simulation applications since it can represent only independent particles, i.e., a packet is represented as a particle whereas an actual packet is composed by multiple portions (flits) that are transferred in a line. Furthermore, a naive implementation of the CA-based ICN simulator consumes large memory resources.

To overcome the problems, we have further extended the CA-based model as the third step of our CA-based approach. Our preliminary results are drawn in literature [1]. Based on the third approach, this paper shows detailed discussions in data structure, modeling of router functions, and miscellaneous optimization techniques. The major contribution of this paper is to propose *practical* ICN models based on the CA-based concept, which is applicable to realistic large-scale simulation within a certain accuracy. Furthermore, this paper offers comprehensive discussions on the

Manuscript received April 12, 2018.

Manuscript revised August 27, 2018.

Manuscript publicized October 5, 2018.

[†]The authors are with Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University, Utsunomiya-shi, 321–8585 Japan.

*This paper is an extended version of [1].

a) E-mail: yokota@is.utsunomiya-u.ac.jp

DOI: 10.1587/transinf.2018EDP7131

proposed method from various aspects. Although this paper assumes a simple ICN topology and router organization, the discussion results would be useful for other types of topologies and router organizations.

The rest of this paper is organized as follows. Section 2 refers to related work of this paper from two aspects. Section 3 shows the design policy of our ICN simulator. Section 4 shows our fundamental ICN simulator design that refers to the CA-based concept. Section 5 offers comprehensive discussions from the optimization viewpoint. Section 6 shows the evaluation results followed by discussions and additional evaluations in Sect. 7. Section 8 concludes this paper.

2. Related Work

As the previous section introduced, this paper stands on an extension of cellular automata for accelerating large-scale simulation, in which table-based operations are introduced. We should discuss the novelty of our work from the following two aspects; acceleration of ICN simulation and application of cellular automata.

2.1 Acceleration Techniques for ICN Simulation

The first aspect is acceleration techniques in ICN simulation, since quick evaluations are inevitable for estimating the performance of large-scale system in a limited evaluation time. Literature shows three major techniques in accelerating ICN simulation (or emulation), which include FPGA, GPGPU, and other technologies as follows.

Takamaeda and Kise et al. have proposed an FPGA-based platform for system-level evaluation of their tiled architecture [15], [16]. They achieved 1487K cycles/sec. for four-core systems and 342K cycles/sec. for 16-core systems. Wang et al. have developed their DART system based on the FPGA technology in which various optimizations are introduced [17]. They achieved from 6M to 11M cycles/sec. on a 64×64 system. Chu and Kise et al. have proposed an extremely optimized operation on an FPGA platform to emulate NoCs [18], [19].

FPGA-based models have advantages in straightforward implementation of hardware design. For example, if a router is designed in a hardware description language, the corresponding ICN will be easily implemented in FPGA devices. This feature guarantees preciseness of evaluation results in performance. Another advantage of the FPGA-based methods is simulation speed. Since the FPGA device directly implements actual router hardware, necessary router functions are operable without any overheads.

Although today's state-of-the-art FPGA devices can employ large-scale digital circuits, only a limited number of routers are implemented in a single FPGA device. Thus, many of these FPGA-based methods employ multiplexed operations for large-scale systems that cannot be embedded in the FPGA device. For example, literature [17] reports that multiplexed implementation slows down the simulation

speed at ten to twenty times, since multiplexed sub-modules make shared use of embedded RAM modules within the FPGA device. Many of FPGA-based simulators actually cover relatively small-scale systems, where undesirable performance issues are suggested for large-scale systems. We will discuss the multiplexing behavior in Sect. 7.5 from the viewpoint of extremely large-scale simulation. Furthermore, on the other hand of the merits, FPGA-based methods have weak points in debugging and metrics acquisition, which includes statistical features required for system evaluation such as throughput and average latency.

GPGPU offers wide range of solutions where the objective problem has sufficient parallelism. Absolutely, ICN simulation is not an exception of GPGPU applications, and several studies are reported by Pinto [20], Zolghadr [21], Kumar [22], and Raghav [23]. Our research is unique in the fundamental standpoint, i.e., our method is based on the cellular automata concept.

Design of large-scale parallel systems drives large-scale simulation environments. Recent representative platforms are BigSim [24] and NSIM (OpenNSIM) [25], [26]. Literature [7] also reports MPI implementation of an ICN simulator.

Fast and accurate feature of simulator is inevitable for large-scale (or exa-scale) simulation. Some simulators that include OMNeT++ [27], SST [28], and CODES [29] perform as system-level simulation/estimation tools based on their discrete event simulation features. These simulators support multiple levels of abstraction in the common fashion of discrete event simulation and, furthermore, they are well parallelized to support large-scale simulations. This paper will make qualitative comparison in Sect. 7.6.

2.2 Cellular Automata Based Simulation

As the previous section briefly introduced, CA-based modeling is applied widely to physics and its related area [10], [30]–[32]. One of the remarkable applications is *flow*, which includes car traffic, pedestrian flow, and fluid dynamics [33]–[36]. Car and pedestrian traffic research has contributed in theoretical fundamentals, such as qualitative features and phase transition, in the corresponding research area, although, preciseness of simulation results is left in the discussions.

Fluid dynamics application contributes as lattice gas methods (LGM) [32], which offers an approximate solution of fluid dynamics problem. Computation in LGM can be formulated in a massively parallel fashion, thus, GPGPU technology is introduced to accelerate LGM computation. Johnson et al. have reported their GPGPU implementation of fluid dynamics computation [37]. They have introduced a table-based method in updating status of neighboring particles. The similar table-based technique can be found in many fluid dynamics programs in github. This paper also introduces a table-based method to efficiently update particle (i.e., packet) status.

3. Design Policy

Before starting the center point of our discussions, this section clarifies underlying assumptions as our design policies in discussing effective ICN simulation methods.

3.1 Homogeneous Organization

This paper intends to build a realistic platform for large-scale ICN simulation. As practical massively parallel systems employ various types of ICNs, they sometimes have complicated or high-radix topologies. Although the simulation platform should be widely applicable, we prioritize enabling extreme-scale simulation, instead of wide generality, so as to establish fundamental methodologies. In this paper, we simplified the topological discussions to two-dimensional torus networks, which we can extend based on the results and knowledge without loss of generality.

3.2 Simplification by CA Concept

This paper intends to offer a fundamental and practical tool for simulating and discussing large-scale ICNs. The simulator should be sufficiently accurate in results. Although faithful straight-forward implementations of router models are preferable, the simulator does not always require 100-percent accuracy in the simulation results.

This paper tries to apply the CA-based concept to router functions, while maintaining sufficient accuracy in simulation results. Fundamentally, the CA-based method assumes that the ICN consists of arrays of cells and that the cell maintains a packet based on the box-ball behavior. This approach sometimes simplifies the router behaviors. Theoretically speaking, if the simulator successfully handles appropriate entities of packets, buffers, switches, and input/output ports, we can expect sufficient accuracy in the simulation results.

3.3 Practical for Massively Parallel Systems

The simulator should cover wide range of system scale from tens to millions of nodes. As existing simulators cover small- and mid-scale systems, this paper specifically intends large-scale systems of 64×64 , 128×128 , ..., 1024×1024 nodes and more. Needless to say, simulation time should be sufficiently short for repetitive evaluations of ICN performance.

3.4 Applicable to Parallelization Techniques

Large-scale simulation inevitably requires fast execution of simulation operations. Fortunately, each router in an ICN acts independently in a parallel processing fashion. This suites parallelization techniques that include GPGPU and multithreading. This paper intends two parallelization aspects. One is GPGPU-aware acceleration, in which we

should consider warp divergence, coalesced accesses to global memory, and specialized memory components (constant, shared, and texture memories). The second one is non-GPU platforms where multi-core processor is available so that we can make efficient use of multithreading techniques.

4. Fundamental Cellular Automata-Based Model

4.1 Router Organization

This paper assumes a specific router organization as Fig. 1(a) shows. As Sect. 3.1 shows, we assume two-dimensional torus topology, where each router has four input/output ports for its neighboring routers (in N , E , S and W directions, respectively). The router also has one input/output port for processor, which is denoted as P in the figure. Each input port consists of a multiplexor and a set of buffers. Each buffer corresponds to a specific virtual channel. The head packet in each buffer requests an appropriate output port to the crossbar switch. The crossbar switch accepts an appropriate output request and the selected packet is transferred to the next router (or processor). Since virtual channels share a physical link and only one virtual channel is operational, the router assumes a demultiplexor to select an appropriate virtual channel at each output port. Starting from the fundamental organization of router, we will conduct our discussion to the CA-based simulation models.

As described in Sect. 2.2, various *flow* simulation problems are modeled in the cellular automata principle. In the flow simulation application, the target system is modeled by means of self-driven particles that exclusively occupy cells. Each particle occupies a cell and it moves to the neighboring cell according to an appropriate function of cellular automata, which is explained as the *box-ball* model. Each cell holds at most one particle.

Message packets in parallel systems have quite similar features with those in flow simulation problems; no packets disappear (unless they arrive at their destinations), and they interfere to each others when they require the same resource (i.e., collision). With the simple computational features of cellular automata, literature [11]–[13] shows the

Fig. 1 Router organization and its CA model.

```

1 struct packet {
2   char valid; /* valid flag */
3   char porder; /* flit number (payload order) */
4   short nhops; /* number of hops (optional) */
5   short src_x, src_y; /* source address (optional) */
6   short dst_x, dst_y; /* destination address */
7   int inj_time; /* injection time */
8 };

```

Fig. 2 Packet representation by struct.

advantageous simulation method.

The key point in the CA-based ICN simulation is that all of the router components are modeled in the cellular automata principle. That is, every component is composed of a set of *cells*. Figure 1 (b) shows the conceptual model of the router that we use in this paper. Each packet buffer consists of an array of cells, each of which is occupied by a packet to act as an FIFO buffer. Crossbar switch is also modeled by an array of cells, each of which shows arbitration status in the switch. Actually, only the winner packet can occupy the corresponding crossbar switch cell and other (loser) packets stay in the corresponding packet buffer cells.

4.2 Packet/Flit Representation

Each cell in a cellular automata in the box-ball model only indicates whether it is occupied by a particle (packet) or not in principle. However, each particle (packet) has its properties that are required for simulating practical systems. The CA-based ICN simulator employs packet information as shown in Fig. 2 at every cell. When a packet occupies a cell, the packet information at the cell represents the occupying packet. Furthermore, as a packet moves from a router to another router or destination processor, its property information (*packet information*) is also transferred to the adjacent cell along with the packet. As Sect. 4.4 shows, when a packet consists of multiple flits, each cell represents a flit (not a whole packet).

4.3 Fundamental Function

As Sect. 4.1 and Fig. 1 show, this paper assumes a simple router organization that consists of buffer, crossbar switch, and output port. Thus, this paper models the corresponding three functions; *buffer* (bf), *crossbar switch* (cb), and *virtual channel* (vc) functions.

Based on the fundamental principle of cellular automata, operations in a router are defined in terms of cells that logically organize the router. Every cell determines its own status in the next simulation step according to the status of its neighboring cells. Literature [11]–[13] appropriately designs the update rule of cells so that the simulation model operates router functions consistently.

The design considers effective parallel execution by means of the GPGPU technology through the following points.

Premise of parallel execution. ICN simulation is executed

Fig. 3 Basic buffer organization.

Fig. 4 Basic buffer function by rule table.

by one or more threads whose maximum equals to the number of routers. The simulator avoids centralized control and shared data among threads for effective parallel execution at high level.

Avoidance of conditional branches. The simulator introduces table-based operations to avoid conditional branches. This matches to the SIMD operation principle in GPGPU.

Grouped update of cells. In conservative cellular automata systems, every cell updates its status independently. However, the ICN model allows a group of cells update in a single operation. This contributes simulation accuracy as well as acceleration of simulation speed.

Each function is defined as an algorithm that specifies update rule of cells to simulate appropriate movement of packet particles along with their properties.

4.3.1 Buffer Function

Each input port employs packet buffers that correspond to the virtual channels. The router receives series of packets from its neighboring router as well as its corresponding processor. The received packet is once stored in the appropriate packet buffer in the packets' virtual channel in a flit-by-flit basis. Packet buffers act in the first-in first-out (FIFO) principle, and the head flit in the packet buffer goes to the crossbar switch without waiting for the tail flit of the packet. If the flit is blocked by the following process, the flit and its succeeding flits wait until the block is released. The buffer should also support the read/busy handshaking function with the neighboring router and the processor so as not to drop any flits.

Each packet buffer is organized by an array of cells in which packets are maintained in an FIFO manner. Figure 3 depicts the buffer organization and the bf function is illustrated in Fig. 4.

Every received packet (or injected packet) is once stored in the tail cell in the buffer, then, it moves forward to the head cell as illustrated in Fig. 4 (1)–(3). The occupation status of the tail cell directly shows ready/busy status of the buffer, which is used for handshaking purpose in the transferring process between adjacent routers. The packet in the head cell moves to the crossbar switch cell through an appropriate arbitration process of the crossbar switch func-

Fig. 5 Crossbar switch function.

tion. When the head cell becomes empty, the cell is filled by moving other packets in the same buffer in order (Fig. 4 (3) and (4)).

4.3.2 Crossbar Switch Function

The router should transfer every incoming packet to an appropriate output port according to its destination node. A crossbar switch is used for this purpose. This function performs a specific routing algorithm, which determines an appropriate output port and virtual channel for every packets, and an arbitration algorithm, which selects one of the competing packets for the same output port and virtual channel.

A crossbar switch can be considered as an integration of multiplexers that are attached at each output port and virtual channel to select an appropriate packet to be transferred. Thus, the cb function is simply organized as a set of cells each of which corresponds to a multiplexer that corresponds to a certain output port and virtual channel. Figure 5 (a) shows the crossbar switch cell (denoted 'X' in this figure). Occupation state of the cb cell (X) represents the status of the multiplexing function. When the cb cell is occupied by the head packet from input port N, this represents that the input port N wins at the arbitration and other ports loose.

4.3.3 Virtual Channel Function

Virtual channel (VC) is introduced to avoid deadlocks and improve communication performance. This paper uses three VCs. Since VCs share the corresponding physical link (i.e., wire), at most one VC can transfer packets while other VCs are blocked. In our CA-based model, the winner packet in the crossbar switch function requests for the physical link so that it is transferred to the neighboring router. Since each virtual channel in each output port has its own crossbar switch function, at most three VCs request the physical link. The virtual channel function selects an appropriate packet from the crossbar switch function.

The cb cells arbitrate conflicting packets at the corresponding output port and VC. Since each output port has multiple VCs, the router model should select at most one cb cell to transfer its packet and prohibit other cb cells to transfer packets. The vc function controls the inter-VC arbitration at each output port. Furthermore, the function controls ready/busy handshaking with the succeeding router (i.e., flow control); when the corresponding VC in the input

port at the succeeding router has no room for packet reception (i.e., busy), the vc function pauses packet transfer until the receiver buffer becomes ready.

Figure 6 shows the vc function. As the previous section (4.3.2) implicitly states, an occupied cb cell represents a request to transfer the corresponding packet to the adjacent router (or destination processor). The vc function arbitrates the requests from the three cb cells and selects an appropriate one that satisfies the flow control.

4.4 Extensions for Multi-Flit Packets

The fundamental cellular automata model of ICN described in Sect. 4 succeeds in accelerating ICN simulation in combination with the GPGPU technology [12], [13]. However, the ICN model still has a fatal disadvantage that the model can only represent packets whose size is one [flit]. In general, a message packet consists of a sequence of flits each of which contains a fixed-sized information. The limited operation in a single flit prevents the simulator to practical simulation applications. Thus, we extend the original ICN model to support packet length.

We, in this paper, assume that a packet consists of p flits[†]. Each processor generates a packet as a sequence of p flits. During a traversal of a packet, from the source node to destination, the sequence of flits is not interrupted by other packet flits and the flits are delivered in-order. We extend the simulation model as follows.

4.4.1 Additional End-of-Packet Property

Each cell has a *valid* flag in a packet information that represents whether the cell is occupied or not (Fig. 2). We extend the *valid* information to indicate that the corresponding flit is the last one in the packet sequence (i.e., *end-of-packet: eop*). Actually, flits before the eop flit has valid flag as $01_{(2)}$ and the eop flit has valid= $11_{(2)}$. Cells that are not occupied have valid= $00_{(2)}$.

[†]It is preferable that packets are composed of various length of flits, in principle. This paper assumes fixed-length packets for simplicity in discussions.

4.4.2 Packet Generation State at Processor

Every processor should maintain its packet generation state. At every simulation cycle, each processor determines its packet generation behavior according to its own generation state. The state transition is simple. When the corresponding node is waiting for generating a new packet, the state shows *idle*. After starting a new packet injection, the state directly shows the number of generated flits at that time. We added the state information to every node.

4.4.3 Interlocking Function

As a sequence of packet flits cannot be interrupted by other packets in the same virtual channel during its trip to the destination processor, arbitration operation in the *cb* function should support interlocking. We added a simple variable to the *cb* cells so that the arbitration operation is suppressed during the consecutive flits of a packet. Once the *cb* cell determines the winner packet as its arbitration operation, the switching function of the cell is fixed until the packet sequence shows an end-of-packet flag.

Note that other functions than *cb* are unchanged, and that the rule tables that are introduced in each functions of *bf*, *cb* and *vc* are unchanged. Packet length can be maintained only by the additional variable attached to the *cb* cells.

4.5 Table-Based Operation

One of the important keys in the CA-based ICN model is table-based update rule. Figure 4 shows the table-based rule as well as the buffering behavior. As a packet buffer is modeled as an array of cells (Fig. 3), the buffer status can be represented in a bit-mapped fashion where each bit represents whether the corresponding cell is occupied (by a packet) or not. When a buffer state is given, the next state of the buffer should be determined, as (1)–(4) in Fig. 4 represent four examples.

The ICN simulator prepares a rule table that defines next buffer status for any buffer conditions. The *bf* function refers the rule table with the bit-mapped representation of the buffer cells as address. Each table entry consists of next state and update rule for each of buffer cells. Figure 4(2) shows that the rightmost two cells are occupied and the tail (leftmost) cell receives an incoming flit. In this situation, the buffer is represented as 10011 in the bitmap form and the corresponding table entry shows the next state 00111 and the update information na na 4 na na. The next state shows the bit-mapped representation of the buffer and the update information specifies the update rule for each cell. ‘4’ in the update rule shows that the third right cell receives flit from the fourth cell (i.e., the tail cell), and na, which is the abbreviation of *not-applicable*, shows that the corresponding cell should not be updated.

Similarly, the idea of rule table is applicable to other

router functions, *cb* and *vc*. Figures 5 (b) and 6 depict the corresponding rule tables, in which each table entry consists of new state information and update rule.

As Fig. 5 (b) shows, the *cb* function uses head cells in five input port (denoted as N, E, S, W, and P) and the crossbar cell (denoted as X) and determines the next state of N, E, S, W, P, and X and update information that shows the arbitration winner. As examples in Fig. 5 (b), example (0) shows none of input buffers requests. Similarly, (1) shows the situation when only E-port requests, (2) shows that two ports (port-E and -S) requests and port-E wins, and (3) shows that no arbitration is drawn since the crossbar cell is occupied (and thus it cannot accept a new packet). Asterisk used in the cell status shows *don't care*.

The update rule in the *vc* function is shown in the lower part in Fig. 6. In the examples in this figure, (0) shows that no buffers request (state 0) while all input buffers are ready to receive (state 0). (1) shows that channel-1 requests output. (2) shows that channels-0 and -1 request output and channel-0 wins. (3) shows that the similar situation with (2) but the input channel-0 is busy (state 1) thus channel-1 wins.

4.6 Implementation Issue

The CA-based simulation method is firstly implemented in CUDA C for GPGPU technology. This section mainly describes implementation issues for GPGPU.

The simulator simply consists of initialization and iteration processes. Rule tables for the three functions (in Sect. 4.5) are prepared by CPU as a major part in the initialization. After transferring necessary information to the GPU device, the simulator enters the main loop and runs for the specified number of iterations. To guarantee synchronous simulation, router behaviors are simulated in two phases; the first phase determines the next state of each router and the second one updates.

The target system has $N \times N$ routers, each router has $D = 5$ input ports (i.e., N, E, S, W, and P), each input/output port has $V = 3$ virtual channels, each virtual channel has $B = 5$ (or 15) flit capacity. Since each virtual channel has B cells for buffer and 1 cell for crossbar switch, we concatenate the buffer and crossbar switch cells in a single data structure. Thus, any cell in the simulator can be indexed in 5-dimensional space. For example, in a router whose address is (x, y) , the b -th cell in the buffer of v -th virtual channel in d -th port is represented as `cell[x][y][d][v][b]` ($1 \leq b \leq B$). $b = 0$ (i.e., `cell[x][y][d][v][0]`) represents the corresponding crossbar switch. The simulator has a linear (1-dimensional) array for all cells, which include buffer cells and crossbar switch cells. We will discuss mapping from 5 dimensional space to linear array in Sect. 5.3 and show performance results in Sect. 7.1.

The parallel programming model of the CA-based method is fundamentally shared-memory model. Thus, executables for GPGPU, multithreaded, and sequential execution models share the same source code. By specifying the compiler toolchain and directives, an appropriate executable

```

1 void procedure_of_router(int x, int y){
2     (( procedure of router at (x,y) ))
3 }
4 #ifdef _CUDA_
5 void CA_sim_kernel(){
6     int x = blockIdx.x * blockDim.x + threadIdx.x;
7     int y = blockIdx.y * blockDim.y + threadIdx.y;
8     procedure_of_router(x, y);
9 }
10 #else
11 void CA_sim_kernel(int from_x, int from_y, int to_x, int to_y){
12     for( int x=from_x ; x<to_x ; x++ ){
13         for( int y=from_y ; y<to_y ; y++ ){
14             procedure_of_router(x, y);
15         }
16     }
17 }
18 #endif

```

Fig. 7 CUDA kernel and multithreaded code example.

can be generated. The main iteration routines that simulate individual router can easily be applied to CUDA and pthread programming models. Figure 7 shows an example. The simulator has a reporting function of statistical metrics in every specified intervals, i.e., the number of received packet and average latency in the interval time.

5. Optimization Issues

5.1 Buffer Models

This section discusses buffer models used in our ICN simulation models. We start with the naive model that directly implements the box-ball model, then, we discuss alternative models for speed-up and reducing resource usage.

5.1.1 Naive Model

The naive buffer model simply implements the box-ball principle as depicted in Sect. 4.3.1 and Fig. 4. As this model is fundamental in the CA-based method, we still use the model for reference purpose.

5.1.2 Compressed Model

Another fatal disadvantage of the originated simulator arises for large buffer sizes. According to the descriptions in Sect. 4.5 and Fig. 4, size of the rule table in the bf function depends on the buffer size. When a buffer has B flits of capacity, the corresponding rule table should have 2^B entries. Furthermore, each buffer should be composed of B cells to maintain the fundamental box-ball model of cellular automata. Thus, for large B cases, large resource usage of the rule table and cells is a problem.

We propose an alternative buffer model that can eliminate unnecessary resource consumption. The newly proposed model utilizes the packet transfer principle that is assumed in this paper. A packet is transferred by flit-by-flit basis and, once a packet transfer begins, it is never interrupted in each virtual channel. This transferring principle guarantees that the next incoming flit to a buffer is determined when the buffer is receiving intermittent flits of the

Fig. 8 Compressed representation of buffer status.

packet.

Here, we introduce an additional and optional assumption that the simulator does not necessarily guarantee the contents of packet, i.e., payload. Appropriateness of this assumption depends on the design policy of the simulator. Although the simulator should guarantee the payloads of the packets, if the simulator should support benchmark (or application) programs, this paper adopts the alternative option. According to the assumption, we extend the buffer cells to maintain multiple flits of a packet by adding a simple variable that represents the number of flits in the cell.

Figure 8 shows the compressed representation of a packet buffer in the right-hand side along with the corresponding behaviors in the naive buffer model in the left-hand side. In this figure, after a new flit arrives at the buffer (1), the rightmost cell receives the succeeding flits (2–3) until it receives the eop flit (4). The next incoming flit should be a new packet, thus, the second right cell receives the flit (5) and the succeeding flits (6–8). As Fig. 8 shows, eight-flit buffer can be represented by using three cells. We further add a cell that is used for buffer tail as shown in Fig. 8. The additional cell, which is unnecessary in the buffer principle, is used for simple ready/busy handshaking in the vc function. The number of cells used in the compressed model depends on the buffer capacity and packet length. For example, as Fig. 8 shows, an eight-flit buffer contains upto three distinct packets: the red, blue and green packets in this example (Fig. 8 (10)–(11)).

The new buffer model can drastically reduce the number of cells in a packet buffer, although, the model cannot control the buffer capacity, since each of the three right-hand cells can contain multiple flits. Thus, the new model should count the number of flits maintained in the buffer to determine its ready/busy status and the additional leftmost cell.

Figure 9 shows the rule table for the compressed representation of buffer model. Two-bit valid flags in the right-hand three cells and one-bit valid flag in the leftmost cell form seven bits of the address of table entry. The table entry includes next state (valid flag) and update information for the three right-hand cells. The update information specifies the cell from which the corresponding cell receives flit(s). For example, the Fig. 9 (3) case shows that the rightmost cell maintains three flits and the tail (leftmost) cell

Fig. 9 Rule table for compressed buffer status.

receives the eop flit. Behavior of the buffer is determined by the rule table entry at 1 00 00 01 which shows that the rightmost cell receives flits from 4th cell (i.e., the leftmost cell) and its status is changed to 11 (which represents eop). Note that na and z mean not-applicable and zero-clear, respectively.

5.1.3 Alternative Naive Model

In a packet buffer, flits stored in the buffer are aligned. In other words, flits are never stored sparsely except the head and tail cells. Thus, naive bit-mapped representation of packet buffer status has a limited number of possible cases. This means that only a limited number of entries in a update rule table are used and other majority of entries are left unused. This redundant representation causes inefficiency. Furthermore, a15, which is the naive buffer model with 15-flit capacity, requires $2^{15} = 32768$ entries that cannot be implemented in the constant memory in GPU devices.

The naive buffer model naively represents buffer status in a bit-mapped manner, however, that representation is not necessary required. Thus, as an alternative naive model, we use the number of stored flits and occupation state in the head and tail cells.

Figure 10 shows the ratios of referred entries in 5-flit and 15-flit buffer models. We use axx, bxx, cxx, and rxx to simply distinct buffer models (where xx shows buffer capacity). axx shows the naive buffer model (Sect. 5.1.1), bxx shows the alternative buffer model (Sect. 5.1.3), cxx shows the compressed model (Sect. 5.1.2), and rxx shows the ring buffer models (Sect. 5.1.4). Specifically, in 15-flit buffer cases, the naive buffer model shows extremely low utilization in update rule table. The alternative naive model enables the simulator to use a constant array in GPU devices, as well as it significantly improves utilization. This figure also shows that other buffer models have some levels of redundancy in their update rule tables.

5.1.4 Ring Buffer Model

Our prior work [1] has clarified pros and cons in the CA-based simulation method. Its most significant benefit is simplicity that can accelerate the simulation speed. On the other hand, naive application of the box-ball principle causes excessive global memory accesses that limits the simulation performance. Thus, the key issue for improving performance is global memory accesses as we will discuss in

(a) ratio of referenced entries, 5-flit buffers

(b) ratio of referenced entries, 15-flit buffers

Fig. 10 Ratios of referenced table entries, 256×256 network.

Fig. 11 Ring buffer model.

Sect. 7.3.3.

The compressed buffer model, which is introduced in Sect. 5.1.2, is one of the promising approaches that intends to reduce the total amount of global memory resources. We discuss an alternative approach that intends to reduce the number of global memory accesses instead of reducing the total amount of memory. We introduce the ring buffer method which realizes first-in first-out buffers with no specialized mechanisms.

The naive buffer models (in Sects. 5.1.1 and 5.1.3) have a linear array of cells as a FIFO buffer. Since these buffer models strictly follow the box-ball principle, an incoming flit is stored in the tail cell and it moves forward to reach the head cell. This fit movement within a buffer causes many global memory accesses.

The ring buffer model has a linear array of cells like the naive buffer models, however, no flits move within the buffer. Figure 11 shows the principle. The model has two pointers, ip (input pointer) and op (output pointer), that point


```

1 struct packet {
2   char valid;
3   char porder;
4   short dst_x, dst_y;
5   int inj_time;
6 } cells[NUM_CELLS];

```

(a) Array of structure

```

1 char valid[NUM_CELLS];
2 char porder[NUM_CELLS];
3 short dst_x[NUM_CELLS], dst_y[NUM_CELLS];
4 int inj_time[NUM_CELLS];

```

(b) Plain Arrays

Fig. 12 Flit representations.

head and tail cells, respectively. An incoming flit is stored in the cell that specified by the ip pointer and the pointer is incremented. Alike the incoming flit, an outgoing flit is stored in the cell that is pointed by the op pointer. This model does not follow the CA principle.

Advantage of the ring buffer model is reduction of global memory accesses, however, it has possible disadvantage in buffer management. In the naive buffer models, the tail cell shows whether the buffer can receive a further flit or not, i.e., ready/busy status. The ring buffer model requires some calculation from the ip and op pointers to determine the ready/busy status of the buffer.

5.2 Packet/Flit Representation

In principle, each cell in the original ICN model maintains complete packet information as shown in Fig. 2 except *porder*. The extended model also uses the same data structure, including *porder*, for each flit. This means that each packet consists of multiple instances of flits and, thus, some of properties in Fig. 2 are redundant.

We should consider the data structure carefully. Redundancy in the data structure causes unnecessary memory resources, although its simple structure will match the GPGPU technology for acceleration. Since we firstly expect fast execution speed, we selected simplicity in data structure as discussed in the following part in this section. With respect to memory resource, we will discuss the compressed buffer model in Sect. 5.1.2.

5.2.1 Arrays of Structure and Plain Arrays

The naive data structure is array of `struct packet`, where each instance of the array corresponds to a cell (Fig. 12 (a)). This data representation is natural and easy to understand, however, its does not match coalesced accesses in GPU devices.

To obtain the maximal effects of coalesced accesses in GPU devices, we use a decomposed data structure as shown in Fig. 12 (b). We can expect continuous memory accesses that will accelerate performance, specifically in GPGPU environment.

Fig. 13 Composite packet information.

5.2.2 Composite Representation

Since some simulation situations may not require some of packet information such as the number of hops and source address, we can try to re-compose packet information by selecting the minimum information for ICN simulation. As Fig. 13 shows, we use a 32-bit composite data structure that includes two-bit valid flag, 6-bit of flit order and two 12-bit destination addresses (in *x*- and *y*-axis), in addition to a 32-bit injection time information.

This composite packet information can reduce memory usage as compared to the original packet information in Fig. 2. We can expect that the composite information increases efficiency in memory accesses, whereas it pays additional computational overheads in composing and decomposing individual information from the composite one.

5.3 Memory Ordering

As discussed in Sect. 4.6, each cell in our CA-based model is indexed in 5-dimensional space. Alignment method from the multi-dimensional space to linear array greatly affects coalesced accesses in GPU devices, thus, we should exchange dimension order according to the computing devices. Actually, the naive representation of `cell[x][y][d][v][b]` (that is shown in Sect. 4.6) does not match GPGPU, since neighboring threads do not access consecutive memory addresses. To maximize the effects of coalesced accesses, `cell[d][v][b][y][x]` is preferable than `cell[x][y][d][v][b]`.

This alignment, or memory ordering, affects differently in multithreaded execution in the ordinary CPUs, where coarse grain parallelism is extracted. The `cell[d][v][b][y][x]` representation requires stride memory accesses for consecutive positions of buffer cells. We will discuss the memory ordering methods in Sect. 7.1.

5.4 Other Optimizations

We adopted ordinary optimization techniques to our prior work [1]. One of the major concerns is elimination of unnecessary operations, which include debugging and checking functions. We further introduce the following optimizations, where we intend to apply GPGPU technology.

5.4.1 Optimization in Rule Tables

We carefully discuss optimization options. We find that bit-mapped representation of cell status that are used in update rules does not contribute in acceleration of execution speed. Thus, we eliminate the `next_state` fields in update rule ta-

bles (from Figs. 4, 5, and 6).

The buffer function uses *update* fields in its update table as shown in Fig. 4. The number of update fields equals to the number of cells in the buffer. For example, a 15-flit buffer uses 15 fields of update information, each of them requires 4 bit width to represent buffer position from 0 to 14. Thus, the naive implementation requires $4[\text{bits/field}] \times 15[\text{fields}] = 60[\text{bits}]$ of table entry. We reduce the width of the table entry by introducing coded representation in 2 bits. **00** shows *no change*, **01** shows *move from +1 position cell*, **10** shows *move from +2 position cell*, and **11** shows *move from the tail cell*. The coded representation enables bit width of $2[\text{bits/field}] \times 15[\text{fields}] = 30[\text{bits}]$.

5.4.2 Table-Based Operation

We further find that table-based operations are effective in the GPGPU environment. We introduce *destination table* that specifies the destination node for every node in the system. The destination table is used except random traffic. The table eliminates conditional branch operations to determine the address of the destination node.

Furthermore, we introduce *neighboring table* to determine the addresses of the neighboring cells in N, E, S, and W directions. This table also reduces conditional branches. The table, as well, offers parallel simulation capability of small-scale networks at a time. Appropriate setting of the table enables the simulator to simulate many small-scale networks simultaneously. It is suitable for parameter study for small-scale networks[†].

6. Evaluation

6.1 Evaluation Environment

We built a CA-based ICN simulator with the optimization techniques described in Sect. 5. The CA-based simulator is parallelized by means of multithreading (i.e., `pthread` library) as well as the CUDA environment for GPGPU technology. We also use a detailed ICN simulator for performance comparison purpose. The latter simulator offers comparative functions with BookSim [3], [39], whereas it is parallelized by means of MPI [7].

Extensions in the previous section are categorized as

- buffer model which represents a packet buffer and
- data structure of packet information.

This paper evaluates the proposed ICN simulation methods as possible combinations of buffer model and data structure. Furthermore, we use two cases in buffer capacity; 5 and 15 [flits] per packet buffer. Packet length is eight [flits]. With respect to data structure, for fair comparisons, we do not use

the number of hops and source address information^{††}.

This paper uses **axx**, **bxx**, **cxx** and **rx** symbols to represent buffer models as described in Sect. 5.1.3. ‘xx’ is substituted by the capacity of the packet buffer, **a15** for example.

With respect to data structure, we use following three models; **str** that uses naive data structure depicted in Fig. 12 (a), **pln** that decomposes the above data structure to a group of arrays (Fig. 12 (b)), and **cmp** that is the composite representation given in Sect. 5.2.2. Actual simulator instance is represented as a combination of the buffer model and data structure, like (**a05 pln**) for example.

Simulated system is 256×256 2D-torus network where dimension-order routing (DOR) is applied. The number of virtual channels is three. Traffic pattern is uniform random except accuracy evaluation in Fig. 16 in Sect. 6.2. Each simulation runs for 10,000 cycles with a fixed injection rate, where each processor node starts injecting a new packet according to the specified injection rate that indicates the average number of flits per simulation cycle at every node.

In the GPGPU environment (CUDA 9.1), we use parallelized random number generation library ‘cuRand’ offered by NVIDIA, in which we use the Mersenne Twister algorithm.

6.2 Accuracy

We firstly confirm the accuracy of simulation results in the proposed simulator. Figure 14 shows throughput and average latency curves in the simulators. In this figure, **sim** shows the conventional ICN simulator result. Horizontal axis shows packet injection rate and vertical axis shows throughput (in solid lines) at the left-side axis and average latency (in dashed lines) at the right-side axis. Although **sim** results show slightly large average latency, throughput curves match to each other.

The naive buffer model (**axx**) and the alternative naive model (**bxx**) show exactly the same behaviors. Thus, Fig. 14 shows **axx** results as a representative one. The two other buffer models (i.e., the compressed **cxx** and ring buffer (**rx**) models) show slightly different behaviors. **cxx** shows slightly different performance curves in Fig. 14 (a), since the model has a conservative implementation.

We show other accuracy evaluation results from different point of views. Figure 15 shows performance curves in the 64×64 network. This figure shows very similar results with the 256×256 case that is depicted in Fig. 14. This reveals that our simulator runs accurately in other configurations in system size.

Figure 16 shows performance curves in some different traffic patterns: bit-complement, bit-reverse, bit-rotation and perfect shuffle. The figure shows **a05** results as the representative one in the our proposed method, since differences in buffer models are discussed in Fig. 14. This figure also shows qualitatively similar results, where the sim-

[†]We applied the technique to a PSO problem as referred in [38]. We will omit the parameter study application in this paper since it is out of scope.

^{††}Composite representation (Sect. 5.2.2) cannot employ these information.

Fig. 14 Performance curves of the proposed methods and conventional one, 256×256 network.

Fig. 15 Performance curves of the proposed methods and conventional one, 64×64 network.

ulator runs with an acceptable accuracy. Each of the traffic patterns determines destination node address by means of a bit-wise operation of the one-dimensional representation of the source node address. Assume that the notation of the 1-D address is $w_{2n-1}w_{2n-2} \cdots w_0$, the bit-complement traffic use bit-wise complement operation where the source node $w_{2n-1}w_{2n-2} \cdots w_0$ sends packet to $\overline{w_{2n-1}}\overline{w_{2n-2}} \cdots \overline{w_0}$. The bit-reverse pattern re-orders the 1-D address notation, thus the source node sends to $w_0 \cdots w_{2n-2}w_{2n-1}$. Similarly, in bit-rotation and perfect shuffle traffic patterns, the

Fig. 16 Performance curves in various traffic patterns, 256×256 network, 5-flit buffers.

source node sends to $w_{2n-1} \cdots w_1w_0 \rightarrow w_0w_{2n-1} \cdots w_1$ and $w_{2n-1}w_{2n-2} \cdots w_0 \rightarrow w_{2n-2} \cdots w_0w_{2n-1}$, respectively.

Differences in the throughput and average latency come from the sim's modeling method, i.e., the simulator merges the cb and vc functions. Theoretically, the merged cb and vc function is not impossible in the CA-based simulator. However, it requires unnecessary large rule table that is not acceptable in the practical simulation implementation. Thus, we selected the decoupled implementation of the two functions. Note that the difference does not violate our design policy described in Sect. 3.2.

(a) buffer capacity is 5 [flits]

(b) buffer capacity is 15 [flits]

Fig. 17 Speed-up ratio by GeForce GTX1060.

6.3 Simulation Speed

6.3.1 Speed-Up Ratio

Figure 17 shows the simulation speed results in terms of speed-up ratio between the proposed simulator and the conventional one. The figure shows the GPGPU results that employs NVIDIA's GeForce GTX1060 GPU device with 6GB global memory and CUDA 9.1 as development environment. The horizontal axis shows packet injection rate, the vertical axis shows speed-up ratios of the possible combinations of buffer model and data structure. Difference between Figs. 17 (a) and (b) is the size of packet buffer.

Figure 18 shows elapsed time of individual simulator models where the conventional simulator results is also shown as sim. Since the sim takes long time to complete simulation, its elapsed time is displayed with the right-side vertical axis.

Figures 19 and 20 show speed-up ratio and elapsed time in the pthread version of the simulator. In this evaluation, we use Core i7-6700 CPU (3.4GHz clock frequency) with 16GB main memory with the Ubuntu 16.04LTS operating system. The simulator is compiled by gcc (version 5.4.0) with the -O3 optimization level.

Table 1 summarizes the simulation performance results of 128×128 and 256×256 networks. This table shows peak speed-up ratio with its corresponding injection ratio in parentheses for every combination of buffer model and data structure, and it also shows geometric mean of speed-

(a) buffer capacity is 5 [flits]

(b) buffer capacity is 15 [flits]

Fig. 18 Elapsed times (GeForce GTX1060).

(a) buffer capacity is 5 [flits]

(b) buffer capacity is 15 [flits]

Fig. 19 Speed-up ratio (eight threads in Core i7-6700).

up ratio in the right hand. The best performance results are marked by † and ‡ for peak and average performance, respectively.

As this table shows, the r15-str model of the CA-based simulator achieves the best peak performance that marks about 1264 times faster than the conventional sequential

Table 1 Peak and average speed-ups

(a) CPU performance														
system size	buf. size	data str.	peak speed-up						average speed-up					
			axx		bxx		cxx		rxx		axx	bxx	cxx	rxx
128x128	5	cmp	85.7	(5e-6)	77.5	(1e-6)	54.0	(1e-6)	79.5	(1e-6)	‡74.7	68.6	49.9	70.0
		pln	92.7	(1e-6)	93.2	(5e-6)	58.5	(1e-6)	‡95.1	(1e-6)	72.2	72.3	51.1	71.5
		str	46.0	(5e-2)	43.5	(2e-2)	49.5	(1e-6)	51.2	(1e-7)	43.1	40.5	47.5	49.4
	15	cmp	34.4	(5e-2)	37.4	(5e-2)	68.2	(5e-2)	42.1	(1e-7)	25.8	28.8	43.9	37.3
		pln	46.4	(5e-6)	81.9	(1e-7)	56.0	(5e-2)	‡93.6	(5e-6)	37.1	54.6	45.4	‡60.7
		str	34.0	(5e-2)	36.6	(5e-2)	59.4	(5e-2)	42.2	(5e-2)	20.8	23.4	37.9	28.8
256x256	5	cmp	68.3	(5e-2)	62.5	(5e-2)	59.5	(5e-2)	65.8	(2e-7)	57.0	52.2	48.8	59.0
		pln	89.6	(2e-7)	90.8	(2e-7)	59.4	(1e-5)	‡91.8	(2e-7)	‡67.0	66.4	51.5	64.8
		str	58.6	(5e-2)	54.5	(5e-2)	53.3	(5e-2)	60.3	(5e-2)	39.8	37.8	39.9	43.6
	15	cmp	102.7	(5e-2)	104.9	(5e-2)	‡162.0	(5e-2)	122.4	(5e-2)	31.2	34.5	50.2	42.8
		pln	79.7	(5e-2)	79.3	(5e-2)	145.0	(5e-2)	86.1	(5e-2)	44.6	54.9	55.6	‡62.9
		str	97.6	(5e-2)	106.4	(5e-2)	153.3	(5e-2)	120.8	(5e-2)	25.2	29.0	43.2	33.8

(b) GPU performance														
system size	buf. size	data str.	peak speed-up						average speed-up					
			axx		bxx		cxx		rxx		axx	bxx	cxx	rxx
128x128	5	cmp	340.4	(1e-6)	476.6	(1e-7)	344.8	(1e-7)	442.7	(5e-7)	277.3	‡354.2	278.2	345.0
		pln	456.9	(1e-6)	‡531.5	(1e-7)	479.5	(1e-7)	473.7	(1e-7)	313.7	333.5	335.2	322.2
		str	260.7	(5e-7)	375.7	(2e-7)	275.1	(1e-6)	404.6	(2e-6)	231.2	306.7	226.6	322.2
	15	cmp	241.8	(1e-7)	486.3	(1e-7)	334.5	(5e-2)	485.8	(2e-7)	212.8	354.0	288.3	‡384.0
		pln	395.0	(1e-7)	‡555.3	(1e-7)	394.1	(1e-7)	506.0	(1e-7)	268.0	325.2	298.3	340.6
		str	209.0	(5e-2)	385.7	(1e-7)	268.6	(5e-2)	446.6	(1e-7)	160.2	314.3	228.9	366.8
256x256	5	cmp	419.8	(2e-7)	543.1	(2e-7)	405.4	(2e-7)	602.1	(2e-7)	352.1	424.4	337.6	‡471.5
		pln	594.4	(2e-7)	‡665.8	(2e-7)	554.2	(2e-6)	651.1	(2e-6)	413.1	428.0	407.9	435.3
		str	379.1	(5e-2)	427.0	(5e-6)	325.5	(5e-2)	556.4	(2e-6)	302.2	376.1	284.9	456.5
	15	cmp	705.9	(5e-2)	929.7	(5e-2)	1161.6	(5e-2)	1184.4	(5e-2)	278.1	499.0	404.3	‡570.1
		pln	570.8	(5e-2)	675.2	(2e-7)	1063.2	(5e-2)	776.2	(5e-2)	387.4	483.4	458.2	509.7
		str	766.4	(5e-2)	1022.0	(5e-2)	992.1	(5e-2)	‡1264.3	(5e-2)	232.1	452.3	339.8	564.1

Fig. 20 Elapsed times (eight threads in Core i7-6700).

simulator (sim). With respect to average speed-up scores, the r15-cmp model performs 570 times speed-up.

The pthread-version of simulator also achieves remarkable performance. The best peak speed-up ratio is 162 times in the c15-cmp model, and the best average performance is

74.7 times in the a05-cmp model (in 128×128 system).

In the GPGPU environment, the alternative naive model (bxx) performs faster than the (original) naive model (axx) in the 15-flit buffer conditions. This improvement mainly comes from the use of constant memory in GPU devices. As Sect. 5.1.3 describes, b15 can make use of the constant memory for the rule table in the buffer function, whereas a15 has no option but to use the global memory. In the CPU (multithreaded) environment, b15 has no significant advantages over a15.

As Sect. 5.1.2 discussed, the compressed buffer model (cxx) reduces memory usage (i.e., advantage), whereas it requires slightly complicated operations in updating buffer status (i.e., disadvantage). In 5-flit buffer cases, the disadvantage slows down the simulation speed. On the other hand in 15-flit cases in the GPGPU environment, the advantage accelerates the simulation speed, since it reduces the number of accesses to the *slow* global memory.

In the GPGPU environment, bxx beats axx in all the cases in Table 1 (b). This table shows that bxx is strong in peak speed-ups and that, in average speed-ups, rxx is the fastest (or comparable to bxx).

The composite packet representation (cmp, Sect. 5.2.2) does not show significant effects in the CPU (multithreaded) environment. In the GPGPU environment, pln has advantages over cmp in the peak speed-ups. However, in the average speed-ups, cmp beats pln in many conditions as Table 1 (b) shows.

Through the above discussions, results in Table 1 (b)

Fig. 21 Network performance and elapsed time (2048 × 2048 system, GTX1060).

suggest that access to the global memory is the major bottleneck in the GPGPU environment. We will discuss the bottlenecks in Sect. 7.3 and Sect. 7.3.3 supports the suggestion.

6.3.2 Large-Scale Simulation Speed

Our GPU-based simulator allows us to simulate large-scale ICNs. As a demonstrative example, elapsed times of simulation runs of a 2048 × 2048 system for 10,000 cycles are given in Fig. 21 along with the measured network performance (throughput). Elapsed times of the b05-pln model are from 184.6 [secs.] (at 1.0e-7 of injection rate) to 758.9 [secs.] (at 2.0e-3 injection rate).

As a rough comparison, when we measure ten cases of performance for rough evaluation, we need about 30.7 minutes (184.6 [sec/run] × 10 [runs]) for ten simulation runs. According to a rough estimation of elapsed time, the conventional sequential simulator requires more than about 14.2 days[†].

6.4 Maximum Simulation Size

The CA-based simulator dynamically allocates necessary memory area according to the system size. Thus, the simulator fails when the required memory area exceeds the available global memory. Many commercial GPU devices em-

[†]The b05-pln model runs 184.6 seconds at injection rate 1.0e-7. With the 256 × 256 results shown in Sect. 6.3, we estimate that the sequential simulation of 2048 × 2048 requires 184.61 [secs.] × 665.8 [times] = 122,906.68 [secs.] for single run. Thus, ten simulation runs require 122,906 × 10 = 1,229,066 seconds, which corresponds to 14.2 days.

Table 2 Maximum simulation size available at GeForce GTX1060 (6GB global memory).

buf.model	system size ($N \times N$)										
-pkt.str.	2112	2176	2240	2304	2368	2432	2496	2560	2624	2688	2752
a05 cmp	o	o	o	o	o	o	.	.	.	.	.
a05 pln	o	o	o	.	.	.	.	.	.	.	.
a05 str	o	o	o	.	.	.	.	.	.	.	.
c05 cmp	o	o	o	o	o	o	o	o	o	.	.
c05 pln	o	o	o	o	o	o	.	.	.	.	.
c05 str	o	o	o	o	o	o	.	.	.	.	.
r05 cmp	o	o	o	o	.	.	.	.	.	.	.
r05 pln	o	o	o	.	.	.	.	.	.	.	.
r05 str	o	o	o	.	.	.	.	.	.	.	.
	1536	1600	1664	1728	1792	...	2304	2368	2432	2496	2560
a15 cmp	o	o	o	o	.	.	.	.	.	.	.
a15 pln	o	.	.	.	.	.	.	.	.	.	.
a15 str	o	.	.	.	.	.	.	.	.	.	.
c15 cmp	o	o	o	o	o	.	o	o	o	o	.
c15 pln	o	o	o	o	o	.	o	.	.	.	.
c15 str	o	o	o	o	o	.	o	.	.	.	.
r15 cmp	o	o	o	o	.	.	.	.	.	.	.
r15 pln	o	.	.	.	.	.	.	.	.	.	.
r15 str	o	.	.	.	.	.	.	.	.	.	.

ploy limited amount of global memory that limits the simulation size. Thus, we should evaluate the maximum simulation sizes in accordance with our design policy described in Sect. 3.3.

This section tries to discuss the maximum system sizes that the proposed methods can simulate successfully.

We ran simulation runs in every combination of buffer model (axx, bxx, cxx, and rxx) and data structure of packet/flit attribute (cmp, pln and str) for several variants of system size. Table 2 shows the maximum simulation size results in the GeForce GTX1060 device that employs 6GB of global memory. In this figure, the symbols ‘O’ and ‘.’ represent whether the simulation run was successful or not, respectively. The table reveals the effectiveness of the compressed buffer model (Sect. 5.1.2) and the composite representation (Sect. 5.2.2).

The compressed buffer model (cxx) achieves considerable reduction in memory resources and the model enables extremely large-scale simulations. All buffer models except cxx use five cells for a 5-flit buffer and fifteen cells for a 15-flit buffer. Actually, c05 uses three cells for each packet buffer, and c15 uses only four cells.

The composite representation (cmp) also reduces memory resources compared to the pln and str models. The naive buffer model axx and ring buffer model rxx show equivalent results, since they consume memory area in proportion to the buffer capacity. Note that the alternative buffer model bxx also shows equivalent behaviors to axx and the results of bxx are omitted. We can select an appropriate model according to simulation speed and memory restriction.

7. Discussion

7.1 Effect on Memory Ordering

As we discussed in Sect. 5.3, difference in memory allo-

Fig. 22 Performance difference by memory order, GPU, 256×256 network.

Fig. 23 Performance difference by memory order, CPU, 256×256 network.

cation order causes performance degradation. We compared two different memory orders that expect coalesced access or not in the GPGPU environment. Figures 22 and 23 show speed-up ratios in the GPU and CPU environments, respectively. In these figures, we denote the `cell[d][v][b][y][x]` order as *coalesced*, and the `cell[x][y][d][v][b]` order as *non-coalesced*, from the GPGPU view. Performance difference in 15-flit buffer case is larger than that in 5-flit case. This shows the strong effects in the memory order of global memory data[†].

Figure 22 shows that coalesced accesses to the global memory gain performance, as emphasized in the CUDA textbook [40], and that the benefit is larger in 15-flit cases than that of 5-flit cases. Interestingly, Figure 23 shows the opposed results where the non-coalesced accesses show definitely higher performance than coalesced accesses. At every simulation cycle, the simulator *sweeps* a large part of the cells, thus it has large working-set. As the cache memories are limited in the GPGPU environment, coalesced accesses to the global memory make full use of the memory bandwidth. On the other hand, in the CPU environment, non-coalesced accesses to the memory area is preferable, since locality characteristics benefit the performance from caching and prefetching.

[†]We use *coalesced* memory order for GPGPU evaluation and *non-coalesced* for CPU evaluation in Sect. 6.3.

(a) injection rate = $1.0e-4$

(b) injection rate = $5.0e-2$

Fig. 24 Number of cores and speed-up ratio, conventional simulator.

7.2 Effective Parallelism

As described in Sect. 3.4, we assume the shared-memory programming model. This section discusses the effective parallelism according to the number of running cores in the parallel processing of the simulator. We compare the effective parallelism in the conventional simulator (parallelized by MPI) and the CA-based simulator (parallelized by the pthread library).

Figure 24 shows the effective parallelism of the conventional simulator, where the horizontal axis shows the number of cores and the vertical axis shows the speed-up ratio (based on the execution time of the single-core condition). We ran the simulator on two different environments. In Fig. 24, *dsm* shows a 10-node cluster where each node employs a Core i7 3770S processor (four cores, 3.10GHz clock and 4GB memory). Also, *smp* shows a 16-core SMP workstation that employs two Xeon E5-2650 processors (eight cores, 2.00GHz clock and 32GB memory). The two numbers in the parentheses show buffer size and system size. Difference in Figs. 24(a) and (b) is injection rate, the former uses 0.0004 [flits/node/cycle] of injection rate that corresponds to a sparse condition, whereas the latter (i.e., Fig. 24(b)) shows heavily congested condition.

The conventional simulator shows good linearity in speed-up. In the congested situation (Fig. 24(b)), the simulator shows super-linear characteristic in 15-flit buffer cases. This implies that the conventional simulator consumes large memory resources in congested situations and that sequential execution of the simulator (i.e., when the number of node is one,) suffers from ineffectiveness in the vast amount

(a) Core i7 6700, 4 cores, 8 hyperthreads

(b) Xeon E5-2650 x2 (8x2=16 cores)

Fig. 25 Number of cores and speed-up ratio, CA-based simulator.

of memory accesses. The 15-flit buffer condition consumes large memory resources especially in congested situation, where many of buffers are filled by transferring packets. Thus, working-set in sequential execution becomes extremely large compared to the parallelized cases.

This fact can explain peculiar speed-up results shown in Figs. 17 (b) and 19 (b), and also Table 1. As shown in Table 1, differences between peak and average speed-up ratios are not so large in 128×128 and 5-flit buffer cases, however, 15-flit buffer cases in 256×256 results show large difference. For example, according to Table 1 (b), speed-up ratios of the a05-pln model are 340.4 (at peak) and 277.3 (in average) in 5-flit buffer in 128×128 network and their difference is $340.4/277.3 = 1.23$. However, in 15-flit buffer in 256 network case, peak and average speed-up ratios are 705.9 and 278.1 (i.e., difference is $705.9/278.1 = 2.54$). Thus, extremely large achievement in speed-up ratio (i.e., 1264 times faster) comes from ineffectiveness in huge memory accesses in the conventional simulator.

We also ran the CA-base simulator on two environments as Fig. 25 shows, both environments are based on SMP architecture. Figure 25(a) shows the ordinary CPU results of one Core i7 6700 processor. The execution environment of Fig. 25(b) is the same with *smp* in Fig. 24, where two Xeon E5-2650 processors are employed. Results in Fig. 25 suggest that the CA-based is still memory intensive and limited parallelism in SMP machines.

smp in Fig. 24 and *pth* in Figs. 25 (a) and (b) are common in the platform architecture that is based on SMP. Although *dsm* results show good linearity in the performance curves, *smp* and *pth* results show saturated performance.

Fig. 26 Number of divergent branches and elapsed time.

Especially, *pth* curves saturate in the early stage of the number of nodes (cores). This suggests that each thread has large working-set that interferes to each other in the shared cache. Furthermore, our simulator (*pth*) inevitably requires inter-thread communication on the shared memory, which sometimes causes false sharing on the cache memory, to implement the flow control mechanism between two adjacent routers. Moreover, our simple implementation of the multithreaded simulator employs frequent barrier synchronization operations, which draw considerable overheads, to maintain appropriate memory accesses.

7.3 Bottleneck Analysis

This subsection discusses possible bottlenecks in CUDA-implemented version of our simulator. CUDA toolkit offers a profiler that can acquire various items in GPU [41]. According to the guide from NVIDIA Corp. [40], we have measured three metrics by *nvprof* profiler; the number of divergent branches within a warp (*divergent_branch*), ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor (*achieved_occupancy*), and the number of global memory load and store access transactions (*gld_transactions* and *gst_transactions*, respectively).

In the actual situations, the execution speed is not affected simply by a unique factor. The aim of this section is to clarify the dominant factor for execution speed in the GPGPU environment. To discuss the bottleneck factors, we acquire the every relationship between a possible factor and the elapsed time. When the factor shows linear relationship with the elapsed time, the factor dominantly affects the execution speed.

7.3.1 Divergent Branches

Figure 26 shows the number of divergent branches in the CUDA-version simulator. The horizontal axis shows the packet injection rate and the number of divergent branches are drawn with the left-side vertical axis. This figure also illustrates elapsed time of each simulator instance (in dashed line) with the right-side vertical axis. This figure shows that difference in data structure does not affect execution

Fig. 27 Occupancy and elapsed time.

Fig. 28 Number of global memory accesses and elapsed time.

speed (elapsed time). This figure shows that the divergent branches are extremely increased in congested situations (where the injection rate exceeds 0.002), whereas the elapsed time is gradually increased. This shows that the divergent branches are not dominant factor in the CA-based simulator.

7.3.2 Occupancy

Figure 27 shows the achieved occupancy metrics acquired by *nvprof*. Similar to the divergent branch graphs in Fig. 26, Fig. 27 shows achieved occupancy with solid lines at the left-side vertical axis along with elapsed times at the right-side vertical axis. This figure illustrates that the decrement in the achieved occupancy corresponds to the increase of elapsed time, however, it shows that the achieved occupancy is not the dominant factor in simulation performance, since two of the occupancy curves are similar and the other one differs while the three models (*cmp*, *pln*, and *str*) show totally different curves in elapsed time.

7.3.3 Global Memory Access

Figure 28 shows the number of global memory accesses (i.e., sum of the numbers of global loads and stores) with respect to injection rate. This figure also illustrates elapsed times in dashed lines with right-side vertical axis, and it compares *cmp* and *pln* data structure from the point of view in global memory access.

In Fig. 28, we can recognize strong correlation between the number of global memory accesses and elapsed time, where this factor shows linear relationship. This suggests the bottleneck of the CA-based simulator.

Fig. 29 Elapsed time with/without rule table.

7.4 Effects of Rule Tables

To clarify the effects of the table-based operation in the update rule, we extend the CA-based simulator to emulate the table. Figure 29 shows the results, where the system size is 256x256, buffer size is 5 flits (a05), and plain array data structure (*pln*) is used. The horizontal axis shows injection rate and vertical axes on the both sides show elapsed time of 10,000-cycle simulation. Red curves show GPU results and green curves show CPU results. Table-based operation results are shown in solid curves.

To avoid complicated modification of source programs in w/o table cases, we emulate rule tables by their index. We prepare corresponding functions for each of the rule tables and substitute the statements that read the rule table to call the corresponding function. For example, “*upd* = *rule_table*[*idx*];” in the original program is substituted by “*upd* = *rule_table_func*(*idx*);” in the emulated version. This evaluation method cannot guarantee 100-percent accuracy in the effect of rule tables, however, we consider that we can roughly recognize the effects.

Figure 29 shows that the table-based operations are effective in the CPU (multithreaded) environment. The rule tables are referred in every simulation cycle, even when most of the routers are in an idle status (e.g., there are only a few numbers of packets in the system). Performance difference between tabular access and procedural one shows the effectiveness of the rule tables.

On the other hand, in the GPGPU environment, the effect is not so large as we initially expected, as Fig. 29 shows. The GPU version of the simulator places rule tables in constant memory if the capacity of the table matches the constant memory in the target GPU device. Although the GPGPU environment for Fig. 29 makes use of the constant memory, the table-based method does not sufficiently benefit the performance. Its possible reasons include limited bandwidth of the constant memory and operating frequency in the GPU device, however, detailed analysis is left for future work. Note that Fig. 29 does not deny the effectiveness of the proposed method in this paper.

7.5 Router Context

As described in Sect. 2.1, due to insufficient number of com-

ponents that an FPGA device can implement, FPGA-based simulation (or emulation) techniques employ multiplexed operations for large-scale systems. Although an FPGA device offers inherent parallel operations in itself, only a part of the system can be simulated (emulated) simultaneously. Thus, it should load necessary information for simulation (emulation) from global memory before it starts its simulation (emulation) operations. Since the loaded information is updated during the simulation operation, the information is stored into global memory again. Therefore, FPGA-based methods should repeat the load-operate-store steps until it covers the whole system before it can proceed to the next simulation cycle.

GPU, as well as FPGA, cannot cover the whole system in its simulation process, although it has thousands of processing cores that can work simultaneously. Thus, it should repeat the load-operate-store steps in the same way with the FPGA-based methods.

We can recognize that both of the two devices (i.e., FPGA and GPU) load necessary information of router and then it update and save the information. This multiplexed operation is inevitable as far as the devices can cover only a part of the system. We think of the information of router as *context*.

Here, we introduce a rough assumption in which each acceleration device load router contexts naively[†]. Dominant component in the naive router context is packet buffer. In the fundamental implementation of our CA-based models, each flits in each packet employ a packet information as Fig. 2 shows. Each router has five ports, including the processor port, and each port has three virtual channels which has its own packet buffer. Thus, the router context is roughly calculated as $n_{\text{DIR}} \times n_{\text{VC}} \times n_{\text{buf}} \times \text{sizeof}(\text{packet info.})$. For example, in case of $n_{\text{buf}} = 5$ and the composite representation (as shown in Sect. 5.2.2), as $n_{\text{DIR}} = 5$, $n_{\text{VC}} = 3$, $n_{\text{buf}} = 5$, and $\text{sizeof}(\text{packet info.}) = 8$, the router context becomes $5 \times 3 \times 5 \times 8 = 600$ [bytes]. By multiplying the router context by the number of routers, we can obtain the total amount of contexts, which means necessary memory accesses to complete the one cycle of simulation.

In a 1024×1024 system, roughly 600MB of global memory is accessed in one simulation cycle. If the simulation platform employs a memory system that can perform 1.0GB/sec throughput, it takes about 0.6 seconds for memory accesses in one simulation cycle. According to our evaluation results on 1024×1024 systems, our GPU-based simulator requires 47.99 seconds to complete 10,000 cycles when the injection rate is $1.0\text{e-}7$ whereas it requires 204.7 seconds when the rate is $1.6\text{e-}3$ (in the b05-pln model). However we can guess that the four-time difference comes from the net context (in other word, size of *working-set*), the GPU-based simulation has a room for further performance improvement. From the platform's memory band-

width of 192GB/sec (in catalog, GeForce GTX1060 device), the 600MB naive router context requires $600 \text{ [MB]} / 192 \text{ [GB/sec]} = 3.125 \text{ [msec.]}$, and thus, the elapsed time of the 10,000 cycle simulation is estimated as 31.25 [secs.] in an ideal case.

7.6 Comparison with Other Simulation Principle

As described in Sect. 2.1, recent advances in discrete event simulators (DESSs) demonstrate their effectiveness for exascale systems. This section makes qualitative discussions on the performance issues by comparing with DESSs, assuming GPGPU environments, in terms of following two factors: simulation principle and internal simulation behavior.

In a discrete event simulation model, *events* are scheduled at discrete points in simulated time and the simulator processes every event in a time-series fashion. Thus, in general in DESSs, every event should be ordered in time for correct simulation results and this limits the parallelism in a naive implementation. SST and other representative DESSs parallelize their simulation kernels by means of MPI (and other adequate methods) and they employ effective acceleration techniques that decades of research in parallel and distributed computing has contributed [42]. To process series of events appropriately, the parallelization is basically in an MIMD fashion, whereas GPGPU operates in the SIMD principle.

On the other hand of the wide generality in DESSs, some classes of dedicated simulators are successful in efficient parallelization in GPGPU environments. When the simulation target is modeled in a directed graph, the graph is divided into many sub-graphs that are mapped to parallel tasks [42]. For example, literature [43] shows effective discrete event simulation of digital circuits in GPGPU environments, in which adjacent GPU threads synchronize locally to handle simulation events.

This parallelization principle is applicable to DESSs for the network simulation purposes in GPGPU environments, where local events are transferred between adjacent routers. In this case, each GPU thread simulates the behavior of a router instance and each discrete event shows traversal of individual flit to the neighboring router.

In an ideal assumption, the simulation performance of DESSs depends on the frequency of the discrete events and the computational complexity of the corresponding router behavior. In the sparse situations, in which packets are rarely injected from processors, event frequency is low and the simulation time rapidly advances in DESSs.

On the other hand, in the modest and congested situations, the discrete events occur in (almost) every cycle of the simulation. In these situations, simulation behaviors are quite similar to those of the cycle-based simulators, in which our CA-based method is also categorized, since all parallel threads are synchronized in every simulation cycle. Thus, difference in simulation performance arises from computational complexity in router simulation. Our CA-based method is advantageous in its simple and light-weight

[†]Many of acceleration methods reduce the context to avoid unnecessary memory accesses and operations. We can recognize the minimum information as a kind of *working-set*. This paper, however, tries simplified discussion by assuming constant context.

simulation mechanism that matches to the GPGPU environments. Thus, we can expect better performance in our method than that of DESs.

The second issue is about the internal simulation behaviors, namely, memory access characteristic. As the network becomes congested, the number of in-flight packets, which are being transferred among routers until their destinations, is drastically increased. In such situations, the simulator should handle large amount of packet information, where we cannot expect locality in memory accesses.

In our GPGPU implementation of the CA-based simulator, many of memory accesses are ordered by means of the SIMD execution in the GPU device, where we can expect effective coalesced accesses to the global memory and good performance. Figure 22 (in Sect. 7.1) suggests large benefits of the coalesced accesses in the GPGPU implementation.

Our CA-based method is advantageous in global memory access behaviors because of simple data structure and its good affinity to GPGPU environment. On the other hand, our conventional simulator suffers from ineffective memory accesses caused by large working-set as Sect. 7.2 indicates. The memory access performance is one of the major factors of the large speed-ups (1264 times at the maximum). Thus, if network simulation modules in a discrete event simulator are properly designed for the GPGPU environment, the simulator will also achieve similar memory performance with our CA-based one.

7.7 Further Extensions in Router Functions

This paper assumes simple organizations and algorithms in the ICN configuration, routing and arbitration algorithms. This simplification matches to the underlying concept of cellular automata, which brings rich results as shown in the previous sections. On the other hand, we should discuss potentials of the proposed method for other algorithms. This section depicts some implementation examples in arbitration algorithms, that are applied in the crossbar switch function and the virtual channel function. Discussions on generality and limitation of the proposed method are given in the following section (Sect. 7.8).

As described in Sect. 5.1, we introduced several buffer models in this paper. Each model requires different update rule that corresponds to different representation of the rule table. Figure 30 summarizes the table formats used in this paper[†]. This figure shows the index of rule table in the left part and table entry in the right. *buf* shows the buffer status: bit-mapped representation is used in Fig. 30 (a) and the number of occupied cells in the buffer is used in Fig. 30 (b). *t* and *h* show the tail and head cell, respectively. Suffixes *c* and *n* mean ‘current’ and ‘next’, respectively. *upd* (*u* for short) means the update information for each buffer cell (as described in Sect. 4.5). *pa*, *pb*, *pc* and *pd* show the number of flits in the corresponding cell (Sect. 5.1.2).

As denoted in Figs. 5 and 6, the crossbar switch func-

Fig. 30 Variants in update rules in buffer function.

Fig. 31 Variants in update rule in crossbar switch function.

tion and the virtual channel function also use update rule tables. Each function may vary its arbitration method and we can implement the variants of the arbitration algorithms with minor extensions that can still maintain the table-based operation principle^{††}.

Figure 31 shows the variants of arbitration algorithms in the crossbar switch function. *req* and *rdy* indicate bit-mapped representations of *request* and *ready*, respectively. *cc* shows the occupation status of the crossbar switch cell. *flg* shows flag bits. In this figure, *fixed* shows the fixed priority arbitration. In the *flagged* case, the simulator has an additional five-bit flag that corresponds to the five crossbar switch cells in each input port (N, E, S, W, and P). In an arbitration process, when two or more flits simultaneously request the same crossbar switch cell, the flag shows the directions of loser flits so that the loser is prioritized in the next arbitration process. This approximates the first-come first-service arbitration.

By a similar mechanism to the *flagged* case, *round-robin* implements the round-robin arbitration. In this case, each flag shows the index of the winner direction, thus, the crossbar switch function searches the winner flit from the next direction according to the contents of the flag.

With effective use of the flag bits, alternative arbitration algorithms can be implemented in the same configuration of rule table. For example, the *round-robin* configuration can implement an alternative prioritized algorithm where the processor port has the lowest priority.

Figure 32 shows the variants in the virtual channel function. The function determines the winner virtual channel according to the requests from crossbar switch cells and occupation status (i.e., busy/ready status) of the tail cells in the neighboring router. In addition to the fixed priority al-

[†]Note that the ring buffer model does not use rule table.

^{††}The arbitration algorithm used up to here is *fixed priority*.

Fig. 32 Variants in update rule in virtual channel function. **Fig. 33** Performance variants in crossbar switch/virtual channel functions.

gorithm, and likely to the crossbar switch function, the virtual channel function has variants as shown in Fig. 32, i.e., flagged and round-robin arbitration methods.

Different arbitration algorithms cause different performance results. Figure 33 (a) shows the variants in the crossbar switch function, and Fig. 33 (b) shows the variants in the virtual channel function. In these figures, c: and v: show variants in crossbar switch function and virtual channel function, respectively. *:fx, *:fl, *:rr, and *:rc show fixed priority, flagged, round-robin, and round-robin with the lowest priority in processor port, respectively. Modification of the source program is not so costly at introducing different arbitration algorithms. Except the generation of rule tables, only a few lines of modifications were added. This indicates the efficiency of the proposed simulation models.

7.8 Generality and Limitation of CA-Based Method

7.8.1 Network Topology

As described in Sect. 3, this paper assumes 2D-torus topology for simplification of discussions. Thus, we implemented the CA-based simulator for 2D torus, however, the simulator is extensible to other topologies that have arbitrary fixed number of degrees, in principle. We discuss the extensibility from following two viewpoints: network configuration and router functions.

In the current implementation of our simulator, routers and computing nodes are addressed in a two-dimensional space as (x, y) and the simulator uses 1-dimensional index converted from the 2D representation. Using higher dimension systems is a natural extension from the current implementation.

As described in Sect. 5.4.2, the simulator employs the neighboring table. In the current implementation, the table shows the index of the adjacent cell via a specified output port (either N, E, S, or W). This table determines the topology of the objective network and it maintains two-dimensional torus topology in the current form. However, the neighboring table can theoretically represent any four-degree topology. Furthermore, the table is also naturally extensible to arbitrary number of degrees.

We should discuss router functions for increasing the number of ports in each router. Assuming n -dimensional topology, each router has $2n$ ports for neighbors and one port for the corresponding processor. Crossbar switch function receives possible $2n + 1$ requests from each input port and determines the winner port. To match the configuration, the corresponding rule table has $(2n + 2)$ -bit index and $\lceil \log_2(2n + 1) \rceil$ -bit entry (in the ‘fixed’ priority arbitration case in Fig. 31). For example, 5D-torus topology requires 11-bit index and 4-bit ($= \lceil \log_2(5 \times 2 + 1) \rceil$) entry for the rule table, and this configuration is sufficiently feasible in the GPGPU implementation as well as CPU (and multi-threaded) implementations.

The required number of virtual channels may be changed according to the number of degrees in the topology. This affects the configuration of the rule table for the virtual channel function, however, the extension is applicable in the simulator way in the crossbar switch function discussed above.

7.8.2 Pipelined Router

This paper implicitly assumes non-pipelined router architecture, where the three router functions (buffer, crossbar switch and virtual channel) are updated in a single simulation cycle. However, general-purpose simulators requires to support pipelined routers, as the BookSim simulator does. Our simulator can also support pipelined routers by careful synchronization of the three automata that correspond to buffer, crossbar switch and virtual channel functions. However, we cannot guarantee precise behaviors in complicated pipeline architecture since we have insufficient discussions at this stage. Precise pipelined router models are future work.

7.8.3 Buffer Organization

This paper assumes simple buffer organization and behavior, where each buffer works independently from each others and it offers a simple first-in first-out function. Other buffer organizations require more complicated operations in packet buffers, which may require other ideas.

7.8.4 Routing Algorithm

This paper assumes dimension-order routing, where every packet goes along the x -axis until the x address of its destination node matches to the node's x address, and then, the packet goes y direction to reach the destination node. According to the CA-based concept, the key issue in routing algorithm is whether the transferring direction (which includes processor port) is determined by some simple combinatorial operations. In this paper's case, the operation requires the destination address of packet (x, y) , router address (t, u) , and the system size N . If the routing algorithm is deterministic (i.e., not adaptive) and it requires combinatorial operations of packet destination and router addresses and the system configuration parameters, the CA-based concept will easily be applicable to other routing algorithms.

With respect to adaptive algorithms, the CA-based concept will similarly be adoptable, in theory. However, implementation in a simulator will be difficult, since many adaptive algorithms refer many status of router components and their operations are complicated. For example, Duato's protocol, which is one of the representative adaptive algorithm [4], refers to the flow control status between the adjacent routers and it determines the output port and virtual channel for every packet. The routing operation does not match to our CA-based policy, where the router function consists of three distinct functions. Thus, some new ideas are required to implement the adaptive routing algorithm in a CA-based fashion. Detailed discussions on the applicability issues are future work since they are out of scope of this paper.

7.8.5 Flow Control

This paper assumes a simple ready/busy handshaking at router-router and router-processor flow control. Some general-purpose simulator supports more complicated flow control method, credit-based control for example. However, the simulator does not directly support such complicated flow control.

7.8.6 Variable Length Packets

Section 4.4 describes the multi-flit extension in the simulator and this paper assumes fixed-length packets. However, in principle, the extended mechanism can support variable length packet except the compressed buffer model that assumes fixed length packets.

7.8.7 Benchmark Programs

In the simulator, a processing node is modeled as a finite state machine only with a simple numbers of states as described in Sect. 4.4.2. To match practical benchmark programs, such as NPB (NASA Parallel Benchmarks), the simulator should implement the complicated behaviors that can

precisely emulate the execution of the benchmark programs. The current simulator does not support such complicated behaviors.

8. Conclusion

This paper extends the existing CA-based ICN simulator for practical evaluation purpose of large-scale ICNs. The naive implementation of the originated simulator prevents practical application of ICN evaluation, since it restricts packet size and consumes large memory. This paper introduces new ideas to solve the drawbacks of the prior simulator and proposes new simulation models. This paper extends the packet representation by adding eop (end-of-packet) flag and introduces an appropriate handling method of long packets. This enables the new simulator applicable to practical situations. This paper furthermore extends the buffer model and packet representation so as to reduce memory consumption. It enables commercial reasonable GPU device to run a huge-scale simulations. The extended CA-based simulator achieves remarkable speed-up. GPGPU implementation of the simulator on the GeForce GTX1060 device accelerates simulation at most 1264 times faster than the ordinary sequential simulator. Our simulator also achieves at most 162 times speedup in a multithreaded execution on the Core i7-6700 CPU.

Acknowledgments

The authors thank anonymous reviewers for their valuable comments. This work was partly supported by JSPS KAKENHI Grant Numbers 15K00068, 16K00068, and 17K00072.

References

- [1] T. Yokota, K. Ootsu, and T. Ohkawa, "Large-scale interconnection network simulation methods based on cellular automata," *Proc. 5th International Symposium on Computing and Networking CANDAR'17*, pp.58–67, Nov. 2017. DOI: 10.1109/CANDAR.2017.52.
- [2] "TOP500 supercomputer sites." <http://www.top500.org/>.
- [3] W.J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Pub., 2004.
- [4] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*, Morgan Kaufmann Pub., 2003.
- [5] R. Trobec, R. Vasiljević, M. Tomašević, V. Milutinović, R. Beivide, and M. Valero, "Interconnection networks in petascale computer systems: A survey," *ACM Comput. Surv.*, vol.49, no.3, pp.44:1–44:24, Sept. 2016. DOI: 10.1145/2983387.
- [6] T. Yokota, K. Ootsu, and T. Baba, "Are uniform networks scalable?," *Proc. 9th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2008)*, pp.137–140, 2008. DOI: 10.1109/PDCAT.2008.40.
- [7] T. Yokota, K. Ootsu, and T. Baba, "Preliminary discussions on scaling effects of interconnection networks," *IPSI SIG Technical Report*, vol.2008, no.75, pp.91–96, 2008. (in Japanese).
- [8] T. Yokota, K. Ootsu, F. Furukawa, and T. Baba, "Phase transition phenomena in interconnection networks of massively parallel computers," *Journal of the Physical Society of Japan*, vol.75, no.7, p.078401 (7 pages), 2006. DOI:10.1143/JPSJ.75.078401.

- [9] T. Yokota, K. Ootsu, F. Furukawa, and T. Baba, "A cellular automata approach for understanding congestion in interconnection networks," *IPSJ Trans. Advanced Computing Systems*, vol.47, no.7 (ACS-14), pp.21–42, May 2006. (in Japanese).
- [10] S. Wolfram, ed., *Theory and Applications of Cellular Automata*, World Scientific Publishing, 1986.
- [11] T. Yokota, K. Ootsu, and T. Ohkawa, "A cellular automata approach for large-scale interconnection network simulation," *Proc. 1st International Symposium on Computing and Networking (CANDAR'13)*, pp.545–551, Dec. 2013. DOI: 10.1109/CANDAR.2013.97.
- [12] T. Yokota, K. Ootsu, and T. Ohkawa, "Cellular automata as acceleration kernel of interconnection network simulation," *Proc. 2014 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2014)*, pp.561–567, 2014.
- [13] Y. Suzuki, T. Yokota, K. Ootsu, and T. Ohkawa, "Performance improvement of large-scale interconnection network simulator by using GPU," *Proc. 3rd International Symposium on Computing and Networking (CANDAR'15)*, pp.585–589, Dec. 2015. DOI: 10.1109/CANDAR.2015.98.
- [14] Y. Sugiyama, "Physics of traffic flow," *Nagare, The Japan Society of Fluid Mechanics*, vol.22, no.2, pp.95–108, 2003. DOI: 10.1142/nagare1982.22.95. (in Japanese).
- [15] S. Takamaeda, S. Watanabe, K. Kyou, N. Fujieda, K. Uehara, T. Miyoshi, and K. Kise, "ScalableCore system: Scalable HW evaluation environment for many-core architecture research," *IPSJ SIG Technical Report*, vol.2009–ARC–185, no.3, pp.1–10, Oct. 2009. (in Japanese).
- [16] S. Takamaeda-Yamazaki, R. Sasakawa, Y. Sakaguchi, and K. Kise, "An FPGA-based scalable simulation accelerator for tile architectures," *SIGARCH Comput. Archit. News*, vol.39, no.4, pp.38–43, Dec. 2011. DOI: 10.1145/2082156.2082166.
- [17] D. Wang, C. Lo, J. Vasiljevic, N.E. Jerger, and J.G. Steffan, "DART: A programmable architecture for noc simulation on FPGAs," *IEEE Trans. Comput.*, vol.63, no.3, pp.664–678, March 2014. DOI: 10.1109/TC.2012.121.
- [18] T.V. Chu, S. Sato, and K. Kise, "Ultra-fast NoC emulation on a single FPGA," *Proc. 25th International Conference on Field Programmable Logic and Applications (FPL)*, pp.1–8, 2015. DOI: 10.1109/FPL.2015.7294021.
- [19] T.V. Chu, S. Sato, and K. Kise, "Fast and cycle-accurate emulation of large-scale networks-on-chip using a single fpga," *ACM Trans. Reconfigurable Technol. Syst.*, vol.10, no.4, pp.27:1–27:27, Dec. 2017. DOI: 10.1145/3151758.
- [20] L. Benini, C. Pinto, D. Atienza, A. Marongiu, S. Raghav, and M. Ruggiero, "GPGPU-accelerated parallel and fast simulation of thousand-core platforms," *Proc. IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2011)*, pp.53–62, 2011. DOI: 10.1109/CCGrid.2011.64.
- [21] M. Zolghadr, K. Mirhosseini, S. Gorgin, and A. Nayebi, "GPU-based NoC simulator," *Proc. Ninth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE'11)*, pp.83–88, 2011. DOI: 10.1109/MEMOCODE.2011.5970514.
- [22] N. Kumar and A. Sahu, "Bufferless NOC simulation of large multi-core system on GPU hardware," *arXiv:1508.03235v1*, Aug. 2015.
- [23] S. Raghav, M. Ruggiero, A. Marongiu, C. Pinto, D. Atienza, and L. Benini, "GPU acceleration for simulating massively parallel many-core platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol.26, no.5, pp.1336–1349, 2015. DOI: 10.1109/TPDS.2014.2319092.
- [24] G. Zheng, G. Kakulapati, and L.V. Kalé, "BigSim: A parallel simulator for performance prediction of extremely large parallel machines," *Proc. 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, p.78b, April 2004. DOI: 10.1109/IPDPS.2004.1303013.
- [25] H. Miwa, R. Susukita, H. Shibamura, T. Hirao, J. Maki, Y. Inadomi, K. Inoue, Y. Ajima, I. Miyoshi, T. Shimizu, and H. Ando, "NSIM: Communication simulator for next generation extreme-scale interconnection networks," *IPSJ SIG Technical Report*, vol.2010–HPC–125, no.5, pp.1–9, June 2010. (in Japanese).
- [26] H. Shibamura, R. Susukita, T. Hirao, M. Yoshida, T. Kando, H. Miwa, I. Miyoshi, K. Inoue, and K. Murakami, "OpenNSIM interconnect simulation service via a cloud environment," *IPSJ SIG Technical Report*, vol.2010–ARC–192, no.15, pp.1–9, Dec. 2010. (in Japanese).
- [27] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," *Proc. 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools'08)*, pp.60:1–60:10, 2008.
- [28] A.F. Rodrigues, K.S. Hemmert, B.W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol.38, no.4, pp.37–42, March 2011. DOI: 10.1145/1964218.1964225.
- [29] J. Cope, N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross, "CODES: Enabling co-design of multi-layer exascale storage architectures," *Proc. Workshop on Emerging Supercomputing Technologies*, 2011.
- [30] T. Toffoli and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling*, MIT Press, 1987.
- [31] H. Gutowitz, ed., *Cellular Automata: Theory and Experiment*, MIT Press, 1991.
- [32] G.D. Doolen, ed., *Lattice Gas Methods: Theory, Applications, and Hardware*, MIT Press, 1991.
- [33] K. Nagel and M. Schreckenberg, "A cellular automaton model for freeway traffic," *Journal of Physics I France*, vol.2, no.12, pp.2221–2229, Dec. 1992. DOI: 10.1051/jp1:1992277.
- [34] T. Nagatani, "Self-organized criticality and scaling in lifetime of traffic jams," *Journal of the Physical Society of Japan*, vol.64, no.1, pp.31–34, Jan. 1995. DOI: 10.1143/jpsj.64.31.
- [35] S. Tadaki, "Distribution of jam clusters in a two-dimensional cellular automaton traffic flow model with open boundaries," *Journal of the Physical Society of Japan*, vol.66, no.3, pp.514–517, 1997. DOI: 10.1143/jpsj.66.514.
- [36] M. Fukui and Y. Ishibashi, "Self-organized phase transitions in cellular automaton models for pedestrians," *Journal of the Physical Society of Japan*, vol.68, no.8, pp.2861–2863, Aug. 1999. DOI: 10.1143/JPSJ.68.2861.
- [37] M.G.B. Johnson, D.P. Playne, and K.A. Hawick, "Data-parallelism and GPUs for lattice gas fluid simulations," *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'10)*, pp.210–216, July 2010.
- [38] T. Yokota, K. Ootsu, and T. Ohkawa, "A static packet scheduling approach for fast collective communication by using PSO," *IEICE Trans. Inf. & Syst.*, vol.E100-D, no.12, pp.2781–2795, Dec. 2017. DOI: 10.1587/transinf.2017PAP0015.
- [39] N. Jiang, J. Balfour, D.U. Becker, B. Towles, W.J. Dally, G. Micheliogiannakis, and J. Kim, "A detailed and flexible cycle-accurate network-on-chip simulator," *Proc. 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp.86–96, April 2013. DOI: 10.1109/ISPASS.2013.6557149.
- [40] NVIDIA Corp., "Cuda C best practices guide." <http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html>.
- [41] NVIDIA Corp., "Profiler user's guide." <https://docs.nvidia.com/cuda/profiler-users-guide/index.html>.
- [42] R.M. Fujimoto, "Parallel discrete event simulation," *Commun. ACM*, vol.33, no.10, pp.30–53, Oct. 1990. DOI: 10.1145/84537.84545.
- [43] D. Chatterjee, A. DeOrio, and V. Bertacco, "Event-driven gate-level simulation with GP-GPUs," *Proc. 46th Annual Design Automation Conference (DAC'09)*, pp.557–562, 2009. DOI: 10.1145/1629911.1630056.

Takashi Yokota received his B.E., M.E., and Ph.D. degrees from Keio University in 1983, 1985 and 1997, respectively. He joined Mitsubishi Electric Corp. in 1985, and was engaged in several research projects in special-purpose, massively parallel and industrial computers. He was engaged in research and development of a massively parallel computer RWC-1 at Real World Computing Partnership as a senior researcher from 1993 to 1997. From 2001 to 2009, he was an associate professor at

Utsunomiya University. Since 2009, he has been a professor at Utsunomiya University. His research interests include computer architecture, parallel processing, network architecture and design automation. He is a member of IPSJ, IEICE, ACM, and the IEEE Computer Society.

Kanemitsu Ootsu received his B.S. and M.S. degrees from University of Tokyo in 1993 and 1995 respectively, and later he obtained his Ph.D. in Information Science and Technology from University of Tokyo in Japan. From 1997 to 2009, he is a research associate and then an assistant professor at Utsunomiya University. Since 2009, he is an associate professor at Utsunomiya University. His research interests include high-performance computer architecture, multi-core multithread processor architecture, mobile computing devices, binary translation, and dynamic optimization. He is a member of IPSJ, IEICE, and ISCIE.

Takeshi Ohkawa received his B.E. and M.E. and Ph.D. degrees in Electronics Engineering from Tohoku University in 1998, 2000 and 2003, respectively. He was a postdoctoral fellow at Tohoku University, where he engaged in the reconfigurable LSI and software/hardware co-design in 2003. In 2004, he joined National Institute for Advanced Industrial Science and Technology (AIST), Japan as a researcher, where he started working on distributed object technology and its application to FPGA.

From 2009, he started work in start-up company TOPS systems, where he engaged in R&D of heterogeneous multi-core processor and its software platform. From 2011, he is an assistant professor in Utsunomiya University. His research interests include reconfigurable technology, software/hardware co-design, parallel and distributed architecture and component-based design technology. He is a member of ACM, IEICE, and IPSJ.