Design of a Simulator for Mesh-based Reconfigurable Architectures

Kang Sun[1,2], Jun Zheng[1,3], Yuanyuan Li[4], and Xuezeng Pan[1]

[1] College of Computer Science, Zhejiang University, Hangzhou, China.
[2] IBM Global Engineering Solutions, GCG Delivery, Shanghai, China.
swankong@126.com
[3] Zhejiang Institute of Communication and Media, Hangzhou, China.
taurus-zheng@163.com
[4] Department of Computer Science, Shanghai Jiaotong University, Shanghai, China.
brucelevy@163.com

Abstract. Reconfigurable computing has become a hot topic in research due to its high-performance and flexibility. In this paper we present a simulator called JRSim for mesh-based reconfigurable architectures. The purpose of this simulator is to provide a platform to evaluate new architectures, and to assist in analysis of algorithms as well as the visualization of their behavior. JRSim is a platform-independent tool which is implemented by Java. It supports flexible bus structure, user-defined function unit and dynamic reconfiguration. Case studies show that JRSim can simulate the behavior of mesh-based reconfigurable systems correctly and efficiently. This simulator can be used to evaluate reconfigurable system design, or demonstrate the ability of reconfigurable system in an educational environment.

Key words: reconfigurable computing; reconfigurable mesh; simulator; dynamic reconfiguration

1 Introduction

Reconfigurable computing is a new paradigm based on changing the hardware to reconfigure the computation and the communication structure [**?**]. One of the features of reconfigurable computing is spatial computation: the algorithms are directly mapped onto the reconfigurable architectures and the data are processed by spatially distributing the computations rather than temporally sequencing through a shared computational unit. The other feature is configurable data path: the function of the computational units and interconnection network can be changed by some configuration mechanism. Due to the high-performance and flexibility of reconfigurable hardware, it has become a new solution for high-performance computing.

With the rapid development of reconfigurable computing, various reconfigurable architectures have been developed by researchers and the industry [**?**], and all these systems form a very large design space. Designers are then facing the difficult choice of the target architecture which is critical since it can

strongly affect the final system's performance. Furthermore, the development of reconfigurable system CAD tools also requires a tool to help understand system properties in a way that leads to a better placing of data or utilization of available resources [**?**]. In general, a simulator developed in high level language is useful for the research on computing system architectures, because these architectures can be easily changeable with additional application specific function units or bus structures [**?**].

Currently, most reconfigurable system simulators are designed for dedicated systems (e.g. GARP Simulator [**?**]), and they are usually used for hardware functional verification. There is still a lack of research in development of general purpose reconfigurable architecture simulators which can be employed to assist the analysis of algorithms, perform design exploration, and evaluate CAD tools performance. Steckel et al. proposed a general purpose simulator for reconfigurable mesh architectures [**?**], but the processing element model in this simulator is based on RAM machine which only has a very limited instruction set and lacks extensibility. Furthermore, this simulator does not support the feature of dynamic reconfiguration. Vikram and Vasudevan designed a behavior simulator for hardware-software co-simulation of reconfigurable systems [**?**]. This tool is a hybrid system in which the reconfigurable array is designed by Verilog HDL and the micro-controller is implemented by integrating RSIM [**?**] - a C-based micro-processor simulator. A simulator designed by HDL usually needs EDA tool working environment, and the performance of most HDL simulation tools is too low for the performance analysis and design space exploration at the algorithmic and HW/SW partitioning level that we are planning to support [**?**]. In latest study results, Brito et al. introduced a dynamically reconfigurable FPGA simulator designed by SystemC [**?**].

In this paper, we propose JRSim - a 2-dimensional reconfigurable mesh architecture simulator. The advantages of JRSim include:

1. Both the processing element (PE) and the bus structure are scalable. Designer can add user-defined function unit, change data-path granularity, and define new bus structure in the system.
2. It supports the features of dynamically reconfigurable system.
3. JRSim is a platform-independent tool which is implemented by Java. Its user-friendly graphic user interface (GUI) improves the visibility of the system behavior.

2 JRSim Architecture

Fig.**??**. shows JRSim system architecture. The simulation system consists of 3 components: configuration information, a simulation engine, and a graphic user interface (GUI). The configuration information includes (1) system architecture definition, and (2) system function configuration. Simulation engine is the core component of this simulation system: it analyzes configuration information and performs simulation task. The GUI is responsible for showing working procedure and simulation results of the system.

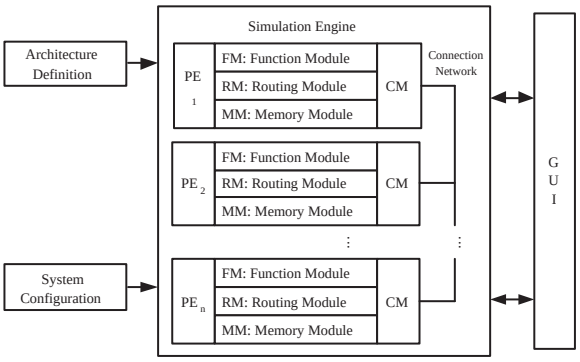

Fig. 1. JRSim system architecture

2.1 Hardware Architecture Definition

The target system of JRSim simulator is mesh-based reconfigurable architecture. Reconfigurable meshes usually contain a set of connected processing elements (PEs). They arrange their PEs mainly as a rectangular 2-D array with horizontal and vertical connections which support rich communication resources for efficient parallelism. Fig.**??**.(a). depicts a typical mesh-based architecture. In each PE, there are four ports at its north, east, west and south sides. These connection ports are used for implementing nearest neighbor connection (NN links) between PEs. Furthermore, additional communication resources can be provided by row or column buses. Each PE may be used to implement an operator and simultaneously to route data words through the array, as shown in Fig.**??**.(b).

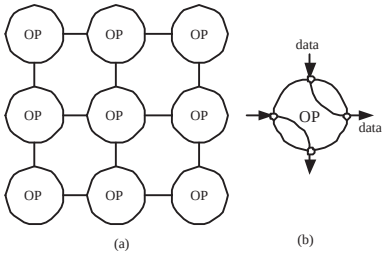

Fig. 2. Reconfigurable mesh architecture

This simulator uses special architecture definition information to describe the properties of hardware architecture, so that it can be applicable for various architectures. In architecture definition, 3 different architectural properties are specified:

1. The size of the reconfigurable array, which is the number of PEs in horizontal and vertical direction, respectively.

2. The connection resources in the reconfigurable array, such as the available repertory of nearest neighbor connections. Per side of the PE can be one or multiple unidirectional and/or bidirectional connections. Besides NN links, some systems may offer one or multiple buses as additional routing resources.
3. The PE operator repertory.

2.2 Processing Elements

A PE is a core information processing engine. It provides general purpose or application specific functions. As shown in Fig.**??**., each PE is constructed by (1) Function Module (FM), (2) Routing Module (RM), (3) Memory Module (MM), and (4) Communication Module (CM). Routing Module is responsible for implementing intra-PE communication, which is data transmission between inner ports of each PE. And Communication Module is responsible for controlling inter-PE communication, which is data transmission between different PEs.

Each PE is specified by four types of parameters:

1. Static parameter. Static parameter is some architecture definition information of PEs. Once defined, it cannot be changed before the completion of a simulation task. In JRSim, the static parameter is PE granularity, which is the data-path width of PE.
2. Function parameter. Function parameter is a set of functions which each PE can be configured to perform. So far each PE supports the integer operators provided by the programming language Java. JRSim also supports user-defined operation, which can simulate the behavior of some application specific function units. A further discussion about user-defined functions is presented in Section **??**.
3. Cost parameter. Cost parameter defines the cost with different PE functionality, such as delay, power, etc.
4. Data transfer parameter. There are 2 types of data transfer parameter. One is transfer parameter for inter-PE connection, and the other is the parameter for intra-PE connection. In each PE, the function unit may need several operands. The operands are supplied by PE input ports or internal memory module. Thus, it is necessary to build a mapping table to maintain the correspondence among function unit's input ports, PE input ports, and internal memory module. Similarly the output data of each PE may directly come from its input ports, or from the output port of the function unit in PE. A mapping table is built to maintain the correspondence between PE output ports and possible output data sources. These two input/output tables record the transfer parameters for intra-PE connection.

2.3 User-defined Functions

A simple description language is designed to describe user-defined functions in JRSim simulator. Fig.**??**. is an example of user-defined function declaration. The #DEFINE ... #END_DEFINE statements define a block of function declaration

statements. As shown in Fig.**??**., each function consists of four parts: function name (ACC), arguments (X, Y, Z), function body, and delay cost (delay=3).

User-defined function declaration is read and analyzed by the interpreter which is integrated in the simulator. Function objects will be constructed and stored in a hash table according to the declarations. Once a PE is configured as a user-defined operator and simulator runs into this operation clock cycle, the interpreter will read out corresponding function object from the hash table, execute this piece of code, and then output the result.

```
#DEFINE
ACC(X,Y, Z) {
  return X*Z+Y;
}

(delay=3)
#END_DEFINE
```

Fig. 3. Example of user-defined function

```
foreach clock cycle do
  foreach node m in system do
    if m.state == READY
      add m into ReadyList
    endif
  endfor
  foreach node n in ReadyList do
    sim(n)
    update state of n and n.child
  endfor
endfor
```

Fig. 4. Pseudo code of simulation process

3 The Implementation of JRSim

3.1 (Re)Configuration of Processing Elements

The simulator uses configuration information to configure the PEs. Configuration information includes PE architecture description, operator repertory, user-defined function declaration and connection resources. An object class is constructed in the simulator program code to record the status and properties of each PE during simulation. These properties include:

1. The position of each PE, which is the X and Y coordinates in the array.
2. The state of each PE. During simulation process, PEs will enter different states in different simulation stages. We will discuss the state transition of PE in Section **??**.
3. The input/output ports of each PE, the attributes of the operands needed by the function unit, and the source of the operands.

3.2 System State Transition

We note the state of arbitrary PE $A_{i,j}$ at clock cycle t as s, and $s \in S$=*(Busy, Ready, Waiting, Idle, Null).*

Busy - PE is executing some tasks.
Ready - PE is ready for running a new task.
Waiting - PE is waiting for some signal (or data) to get ready.
Idle - PE is configured, but there's no task to run immediately.
Null - PE is not used.

At the beginning of each clock cycle, all the PEs which are in *Ready* state will be placed in a queue named *ReadyList*. Then the task assigned to every PE in *ReadyList* will be executed and the corresponding PE's state will be updated. After this procedure, the PE which is in *Ready* state will be put into *ReadyList* in next clock cycle. This loop procedure will be repeated until the final result is produced and there's no more new input data. *ReadyList* is used for gathering all the PEs which should be processed in current clock cycle. Fig.**??**. is the pseudo code of simulation process.

In simulation process, the state transition of each PE $A_{i,j}$ at clock cycle t is determined by:

$$s_{i,j}(t)=F(t,PD_{i,j},IN_{i,j}) \tag{1}$$

In formula (**??**), t is current clock cycle, $PD_{i,j}$ is a set of parent nodes of $A_{i,j}$, which have accomplished their task and generated output data, and $N_{i,j}$ is the number of effective input ports in $A_{i,j}$. If $PD_{i,j}=IN_{i,j}$, it means that all the input data needed by PE $A_{i,j}$ are available and $A_{i,j}$ is in *Ready* state. Otherwise, if $0<PD_{i,j}<IN_{i,j}$, $A_{i,j}$ will be in *Waiting* state. $A_{i,j}$ will be in *Busy* state while executing a task. And if not used, its state will be *Null*. Fig.**??**. is the state machine of PE.

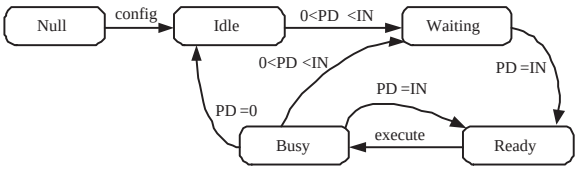

Fig. 5. State machine of reconfigurable PE

3.3 System Working Flow

As a synchronized reconfigurable system simulator, the behavior of JRSim is controlled by clock signal. The system work flow in each clock cycle is shown in Fig.**??**.

1. The system checks whether there are new input data or the system needs to be reconfigured. If some PEs get new input data, the system will update their states and put the PE whose state is *Ready* into *ReadyList*.

2. The system executes the simulation task. Each PE has two tables to record the correspondence between I/O ports and function unit. One is called operand table, where the input data indices are stored. *Operand*[*i*]=*j* means that the *ith* operand of PE is *inVal*[*j*]. *inVal*[*j*] is input port number or internal memory. The other is called output table. The output value is noted as *outVal*[*i*]. In (**??**), *outTab* is output table. If *outTab*[*i*] = -1, it means that the output value of port *i* comes from FM (Function Module) in PE. And if *outTab*[*i*] is between 0 and *inNum*, it means that the output value comes directly from input port. All the other values of *outTab*[*i*] are considered as error.

$$outVal[i] = \begin{cases} resofFM, & if \quad outTab[i] = -1, \\ inVal[outTab[i]], & if \quad 0 \leq outTab[i] \leq inNum, \\ Error, & else. \end{cases} \tag{2}$$

3. The system updates the status of each PE according to the state machine shown in Fig. **??**.
4. When all the PEs finish their task and there's no new input data, the simulation process will terminate.

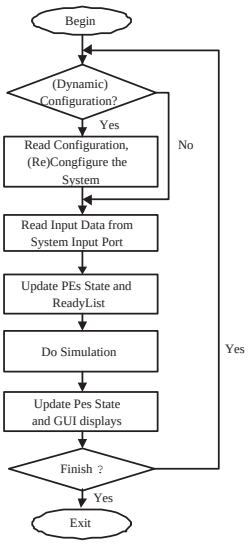

Fig. 6. The working flow of JRSim

3.4 Dynamic Reconfiguration

Reconfigurable technology includes static reconfiguration and dynamic reconfiguration [**?**]. In statically reconfigurable system, if the system needs to initiate

a new configuration, it has to stop computation. But in dynamically reconfigurable system, it permits reconfiguration of a portion of the device while other portions of the device are still performing computations.

JRSim supports dynamically multi-context reconfigurable system modeling. The simulator stores multiple configuration information in memory. A series of configuration $C_i(0 \leq i \leq n)$ are queued in time sequence. At the beginning of each clock cycle, the system will check whether the configuration signal is valid or not. If valid, configuration C_i will be dequeued and the corresponding PEs and connecting networks will be reset and configured. During configuration, all other PEs will keep executing their own tasks.

4 Case Studies

JRSim is implemented by Java programming language, which owns the advantages of object-orientated and platform-independent features. Here we just use matrices multiplication as an illustrational example to demonstrate the effectiveness of the simulator.

Given an $m \times p$ matrix $\boldsymbol{A}$ and a $p \times n$ matrix $\boldsymbol{B}$, the production of $\boldsymbol{A} \times \boldsymbol{B}$ will be an $m \times n$ matrix which is noted as $\boldsymbol{C}$. $c_{i,j}$ represents the element in *ith* row and *jth* column of $\boldsymbol{C}$. $c_{i,j}$ can be calculated by equation (**??**).

$$c_{i,j} = \sum_{k=1}^{p} a_{i,k} \times b_{k,j} \quad (1 \leq i \leq m, 1 \leq j \leq n) \tag{3}$$

According equation (**??**), $c_{i,j}$ can be regarded as the inner product between *ith* row vector in $\boldsymbol{A}$ and *jth* column vector in $\boldsymbol{B}$. A recursive formula for computation of $c_{i,j}$ can be presented as (k is the number of inner product accumulation)[**?**]:

$$\begin{cases} c_{i,j}^{1} = 0, \\ c_{i,j}^{k+1} = c_{i,j}^{k} + a_{i,k} b_{k,j}, \quad i = 1, \ldots, m; j = 1, \ldots, n \\ c_{i,j} = c_{i,j}^{p+1}, \end{cases} \tag{4}$$

4.1 Algorithm Mapping

The multiplication algorithm can be mapped onto a mesh-based reconfigurable array. Fig.**??** shows the mapping results. Each element in Fig.**??**. is a multiply-accumulator (MAC or ACC). Matrix $\boldsymbol{A}$ is preliminarily stored in the array and matrix $\boldsymbol{B}$ is inputted from the bus. After MAC operation, the results will be sent to the right-side adjacent PEs and the inputs from outside buses are also sent to next stage. The output of the PE in *ith* row and *jth* column is just the value of $c_{i,j}$. The two inputs of $PE_{i,k}$ are $c_{i,j}^{k}$ which is produced by $PE_{i,k-1}$ and $b_{k,j}$ which is received from *jth* bus input. $a_{i,j}$ is stored in $PE_{i,k}$. So $c_{i,j}^{k+1}$ can be computed by equation (**??**). After p times recursion of column i, we can get the result of $c_{i,j}$.

$$c_{i,j}^{k+1} = c_{i,j} + a_{i,k} b_{k,j} \tag{5}$$

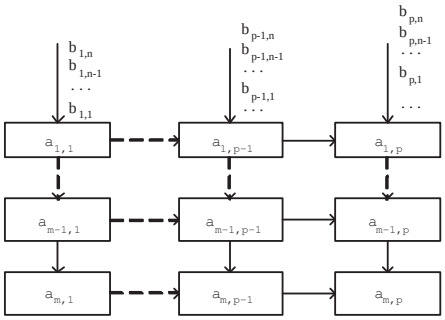

Fig. 7. Mapping matrices multiplication onto PE array

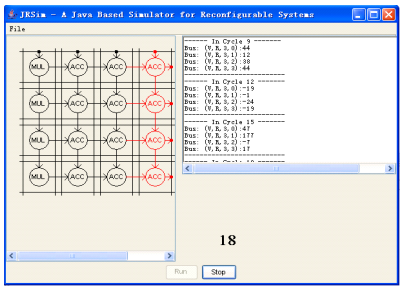

Fig. 8. JRSim runtime window

Table 1. The delay cost of each operation

Operation	Inner Product (ACC)	Multiplication (MUL)	Routing (RT)	Addtion (ADD)
Delay (Cycle)	3	2	0	1

4.2 Experiment Results

We used the two matrices shown in equation (**??**) to verify the simulator. In this experiment, the data-path width of each PE is 16-bit wide. The delay cost of each operation is listed in Table **??**. ACC is MAC operation defined in Fig.**??**.

$$A = \begin{pmatrix} 1 & 3 & 5 & 8 \\ 7 & 9 & 11 & 4 \\ -3 & 9 & 6 & 10 \\ 0 & 6 & -4 & 5 \end{pmatrix}, B = \begin{pmatrix} 6 & 0 & 9 & -4 \\ -2 & 0 & 7 & 3 \\ -4 & 1 & 5 & -2 \\ 8 & -3 & -1 & 0 \end{pmatrix} \tag{6}$$

The simulation result is correct. The computation procedure consumes 19 clock cycles under circumstance that all multi-cycle operations are pipelined. Fig.**??** is a runtime snapshot of the simulator. The red notes are working PEs.

The result shows that matrices multiplication algorithm can be mapped onto reconfigurable array to exploit the parallelism of PEs. The complexity of basic algorithm for multiplying two $n \times n$ matrices is $O(n^3)$. Strassen algorithm can reduce the time for matrices multiplication to $O(n^{lg7})$. But the time complexity of implementing matrices multiplication on reconfigurable hardware is only $O(n)$.

5 Conclusions

In this paper, we introduce a simulator for mesh-based reconfigurable systems. The proposed simulator has the integer operators provided by the programming language Java, and supports dynamic reconfiguration. Expansion interfaces are

also provided by JRSim to add user-defined function units. Case studies show that the implemented simulator system is able to profile application processing time and waiting time of each PE, as well as to investigate additional application specific functions for the architecture of a reconfigurable system. Furthermore, this system can be used to demonstrate the ability of reconfigurable system in an educational environment.

Acknowledgments. This work was supported by Natural Science Foundation of Zhejiang Province, China (Grant No.Y105355), and special project of Zhejiang High-Tech Development Plan (Grant No.2006C11105).

References

1. Vaidyanathan R., Trahan J. L., Dynamic Reconfiguration Architectures and Algorithms. KAP: Kluwer Academic Publishers, 2004, ISBN: 0-306-48428-5.
2. Compton K., Hauck S., Reconfigurable Computing: A Survey of Systems and Software. ACM Computing Surveys, 2002, 34(2): 171-210.
3. Duan R., Fan X.-Y., Gao D.-Y., Shen G., Reconfigurable Computing Technology and Developing Trends. Application Research of Computers, 2004 (8): 14-17 (In Chinese).
4. Shinozaki A, Shima M, Guo M, Kubo M. A high performance simulator system for a multiprocessor system based on a multi-way cluster. Proceedings of 11th Asia-Pacific Computer Systems Architecture Conference (ACSAC'06). Shanghai, China, 2006, Springer LNCS 4186: 231-243.
5. Hauser J. R., Wawrzynek J., Garp: A MIPS Processor with A Reconfigurable Coprocessor. Proceedings of 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines. CA. USA, 1997: 12-21.
6. Steckel C., Middendorf M., Elgindy, H., Schmeck H., A Simulator for The Reconfigurable Mesh Architecture. Proceedings of Workshops Held in Conjunction with 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing. Orlando, FL., USA, 1998, Springer LNCS 1388: 105-110.
7. Vikram K.N., Vasudevan V., Hardware-Software Co-simulation of Bus-based Reconfigurable Systems. Elsvier Microprocessors and Microsystems, 2005, 29(4): 133-144.
8. Hughes C.J., Pai V. S., Ranganathan P., Adve S.V., RSIM: Simulating Shared-memory Multiprocessors with ILP Processors. IEEE Computer, 2002, 35(2): 40-49.
9. Rosa A. L., Lavagno L., Passerone C., A Software Development Tool Chain for A Reconfigurable Processor. Proceedings of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASE'01). Atlanta, GA. USA. 2001: 93-98.
10. Britio A. V., Melcher U. K. M., Rosas W., An Open-source Tool for Simulation of Partially Reconfigurable Systems Using SystemC. Proceedings of IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures. 2006.
11. Sanchez, E., Sipper, M., Haenni, J.-O., Beuchat, J.-L., Stauffer, A., Perez-Uribe, A., Static and Dynamic Configurable Systems. IEEE Transactions on Computers, 1999, 48(6): 556-564.
12. Wu S.-Q., Wang Q., Xie Y.-X., Design and Implementation of Matrix Multiplier for Inverter Harmonic Elimination Model Calculation. Journal of South China University of Technology (Natural Science), 2003, 31(8): 1-5. (In Chinese)