A Java Applet to Visualize Algorithms on Reconfigurable Mesh

Kensuke Miyashita and Reiji Hashimoto

Okayama University of Science, Ridai-cho, Okayama 700-0005, Japan
{miyasita, reiji}@ee.ous.ac.jp

Abstract. Recently, many efficient parallel algorithms on the reconfigurable mesh have been developed. However, it is not easy to understand the behavior of a reconfigurable mesh. This is mainly because the bus topology can change dynamically during the execution of algorithm. In this work, we have developed JRM, a Java applet for visualizing parallel algorithm on the reconfigurable mesh to help on understanding and evaluating it. This applet accepts an algorithm written in C-like language and displays the behavior of it graphically. It also displays much information to evaluate the exact performance of an algorithm. Because this software has been developed as a Java applet, it can run on any operating system with standard web browser.

1 Introduction

A reconfigurable mesh (RM) is one of models of parallel computation based on reconfigurable bus system and mesh of processors. Many results and efficient algorithms on the RM are known, such as sorting, graph related problem, computational geometry and arithmetic computation[5].

We make use of the channel of data flow (bus topology) to solve problems on an RM, but it is not easy to follow data since the bus topology changes dynamically during the execution of algorithm on RM. For same reason, it is also hard to find collisions of data on bus. Therefore it is very hard to design, analysis or understand the algorithms on an RM by hand.

Algorithm visualization is very useful to understand the behavior of computational model. It also assists the user to design, analysis and debug the algorithms. Normally, the user can specify the data and control the execution of algorithm via some user interfaces.

As far as we know, there are three tools to visualize algorithms on RM, the Simulator presented by Steckel et al.[6], the Visualization System presented by us[3] and the VMesh by Bordim et al.[1].

The first one displays the behavior of algorithm graphically and the results of simulation are stored in a file. However the algorithm to be visualize in this simulator must be written in assembler language. Hence a sophisticated algorithm is implemented as a large file and there is more probability of existence of miss-codings or bugs. In addition, it cannot display the data that each processor holds neither statistic information.

The second one is written in Java and accepts an algorithm written in C-like language. It displays the behavior of algorithm graphically with the source code of algorithm in another window. However it can visualize only a constant-time algorithm (the algorithm without loops), and it runs very slowly.

The third one is written in C language and use X Window System as graphical interface so that it can run on the operating system with X Window System. It accepts an algorithm written in C-like language and displays the behavior of algorithm graphically with the source code of it in another window. An algorithm written in C-like language is translated into a real C source file and is compiled and linked with the VMesh objects. Thus it provides flexible programming interface. However the user has to maintain all source files of the VMesh, libraries used by it, C compiler and linker to recompile it.

In this paper, we present the JRM, a Java applet to visualize the behavior of algorithm on an RM. An applet can run in a standard web browser with Java Virtual Machine (JVM) without recompiling. Thus the JRM can run on Windows, Mac OS and many other UNIX-like operating systems. The JRM can visualize the behavior of algorithm and the data that each processor holds. It can also report bus length, the number of inner-bus configurations and some statistic information. Another important feature is that the JRM accepts an algorithm file specified by the URL (Uniform Resource Locator). Thus it can read an algorithm from anywhere in the Internet.

2 Reconfigurable Mesh

A reconfigurable mesh (RM) is formalized as follows. An RM of size $m \times n$ has m rows and n columns, and there are mn SIMD processors located in a 2-dimensional grid. A processor located at the intersection between ith row and jth column is denoted by PE(i, j) $(0 \leq i < m,\ 0 \leq j < n)$. Each processor has its own local memory, while no shared memory is available.

A static (outer) bus connects 2 adjacent processors. An outer bus is attached to a processor via a port. Each processor has 4 ports denoted by N (North), S (South), E (East) and W (West). Each port can be connected by reconfigurable (inner) buses with any possible configurations. A connected component made of outer buses and inner buses is called a sub-bus.

All processors work synchronously and a single step of an RM consists of the following 4 phases: **Phase 1:** change the configuration of inner buses, **Phase 2:** send data to a port (the data which is sent at this phase is transferred through a sub-bus), **Phase 3:** receive data from a port (all processors connected to same sub-bus can receive same data sent at the previous phase), **Phase 4:** execute constant-time local computations. The configuration of inner buses is fixed from phase 2 till phase 4.

In this paper, we employ the CREW (Concurrent Read, Exclusive Write) model of bus. That is, at most one processor connected to a sub-bus is allowed to send data at any given time. When more than one processor send data along same sub-bus, a collision occurs and the data is discarded.

3 Specification of Software

In this section, we present the JRM. The JRM is an applet and it is implemented in Java. An applet can be included in an HTML (Hyper Text Markup Language) page, and it can run inside a standard web browser with JVM.

3.1 User Interface

The JRM has some text fields, 2 buttons and an area to draw graphics as shown in Figure 1.

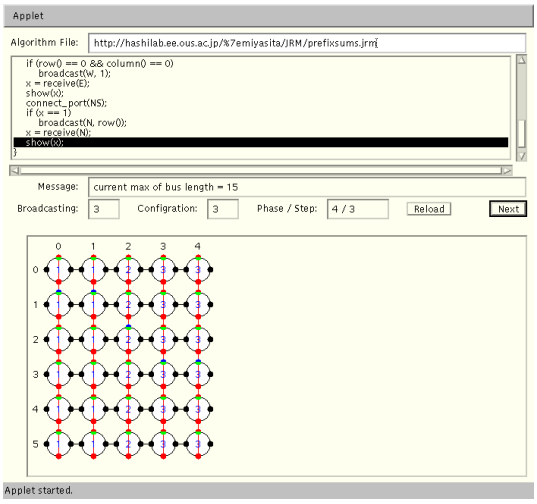

Fig. 1. The appearance of JRM

The algorithm file is specified in the editable text field at the top of window. The JRM accepts an algorithm file specified by the URL. Hence it can read an algorithm file from anywhere in the Internet just like reading a local file and browsing a web page. For this reason, the user can easily publish algorithms and get the other user's algorithms with the JRM.

The JRM accepts an algorithm written in the programming language which has C-like grammar. There are some pre-defined functions to assist the user to write algorithm. The contents of algorithm file are visualized in the main text field and the JRM highlights the line(s) being executed.

The user can execute the algorithm block by block with clicking mouse button on the "Next" button. A block means either a phase or a statement of constant-time computations. Hence the number of blocks is independent of the number of lines and the theoretical computation time of the algorithm.

When each block has been executed, the statistical information is updated and displayed. That information includes the number of broadcasts, the number of configurations used, current number of phase, the number of steps being executed, the length of bus (the number of inner and outer buses) used by current broadcast and the maximum length of bus used by the former broadcasts.

Whenever the user clicks on the "Reload" button, the algorithm file specified in the top text field is re-loaded into the JRM. Thus the user can easily re-execute the algorithm.

The RM is visualized in the main graphic area. The RM of specified size is displayed with row id and column id. The buses and ports on which data flow are painted by some colors to help the user to follow them. Using `show` function in the algorithm file, the user can visualize the specified data being held by each processor at any step.

When a collision of data occurs, a warning message is displayed in the text field at the middle of window and the data is discarded.

3.2 Programming Language

An algorithm file which is accepted by the JRM consists of two parts.

In the first (setup) part, the user must declare the size of the RM on which the algorithm runs and the size of local memory within each processor. And the input of algorithm is stored by each local memory in this part. As well as the user specified value, random value can be used to set up each local memory.

A theoretical algorithm is translated into C-like language and written in the second (main) part of algorithm file. These codes are interpreted and processed by each processor synchronously.

4 Execution of the JRM

In this section, we explain how the JRM works through an implementation of Prefix Sums Algorithm to the JRM for example. Then we show the statistical information of some algorithms by the JRM.

4.1 Visualization of the Prefix Sums Algorithm

The Prefix Sums problem is to compute prefix sums of a binary sequence and defined as follows:

[Prefix Sums Problem]

Input: $a = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ $(a_k \in \{0,1\},\ 0 \leq k < n)$.
Output: $s = \langle s_0, s_1, \ldots, s_{n-1} \rangle$ $(s_k = \sum_{i=0}^{k} a_i)$.

This problem can be solved on an RM of size $(n+1) \times n$ in $O(1)$ time[7].

The algorithm to compute the prefix sums of a binary sequence of length 6 is implemented as follows:

```
 1  # Prefix Sums Algorithm
 2  BEGIN {
 3    COLUMNS = 5;
 4    ROWS = COLUMNS + 1;
 5    MEMORYSIZE = 1;
 6    make_input(0, 0, 0,
        COLUMNS - 1, 0, 0, 1);
 7    alias(x, 0);
 8  }
 9  {
10    show(x);
11    connect_port(NS);
12    if (row() == 0)
13      broadcast(N, x);
14    x = receive(N);
15    show(x);
16    if (x == 0)
17      connect_port(EW);
18    else
19      connect_port(NE, SW);
20    if (row() == 0 &&
                column() == 0)
21      broadcast(W, 1);
22    x = receive(E);
23    show(x);
24    connect_port(NS);
25    if (x == 1)
26      broadcast(N, row());
27    x = receive(N);
28    show(x);
29  }
```

When the user specifies this algorithm file by URL, the JRM loads it and displays the contents, some information and an RM of specified size.

When the user clicks on the "Next" button, the function `show` in line 10 is executed and each value of 0th local memory is visualized. Then the user clicks on the "Next" button four times, the commands in line 11 through 15 have been done.

Next two clicks make the processor PE$(0,0)$ broadcast "1" (line 16 through 21), and the length of bus is informed in the message field. And finally the user clicks eight times and the algorithm has been done. The results and some statistical information of this algorithm are shown as Figure 1 (The binary sequence $\langle 1,0,1,1,0\rangle$ is given as input).

Now the user can re-execute the algorithm by clicking on the "Reload" button, or execute another algorithm by specifying it by URL.

4.2 Some Algorithms Implemented on the JRM

We have implemented some algorithms on the JRM and the statistical information of them are shown in Table 1. The brief description of them are as follows:

Prefix Sums: It is shown in Section 4.1.

Prefix Remainders: A binary sequence $a = \langle a_0, a_1, \ldots, a_{n-1}\rangle$ and an integer w are given, it computes $r_k = (a_0 + a_1 + \cdots + a_k) \bmod w$ $(0 \le k < n)$ on an RM of size $(w+1) \times 2n$ in $O(1)$ time [4].

Sum of Integers: Simulating a look-ahead carry generator, it computes the sum of n d-bit binary values on an RM of size $2n \times 2dn$ in $O(1)$ time [2].

5 Conclusion

We have presented the JRM, a Java applet to visualize the algorithms on Reconfigurable Mesh, in this paper.

Table 1. Statistical information of some algorithms by the JRM

Algorithm	RM Size	Steps	Broadcasts	Configurations	Max. bus length
Prefix Sums	6×5	3	3	3	15
Prefix Remainders	4×12	3	3	8	37
Sum of Integers	8×24	4	6	5	63

The JRM equips the methods to visualize graphically the behavior of algorithm and data. The way of control the execution of algorithm is also provided by the JRM. Furthermore, the JRM can find a collision of data and report statistical information. Thus the JRM helps the user to understand, develop, improve and also study algorithms on the RM.

Because the JRM is implemented as an applet, it can run on any operating system using a standard web browser without re-compiling. The algorithm files can be specified in the URL form, so that the JRM can read them from anywhere in the Internet. And the user can easily publish and share algorithms with the JRM.

We are improving the JRM continuously and planning now to do as follows:

Add pop-up menu: when the user clicks on a processor, a pop-up menu which includes values of its local memory and some statistical information appears.

Make the JRM a multi-threaded applet: each processor parses and executes an algorithm on each thread in parallel.

References

1. J. L. Bordim, T. Watanabe, K. Nakano, and T. Hayashi, A Tool for Algorithm Visualization on the Reconfigurable Mesh, *Proceedings of ISPAN'99*, 1999.
2. J.-W. Jang, and V. K. Prasanna, An Optimal Sorting Algorithm on Reconfigurable Mesh, *Proceedings of 6th International Parallel Processing Symposium*, pp. 130–137, 1992.
3. K. Miyashita, Y. Shimizu, and R. Hashimoto, A Visualization System for Algorithms on PARBS, *Proceedings of International Conference on Computers and Information Technology (ICCIT)*, pp. 215–219, 1998.
4. K. Nakano, T. Masuzawa, and N. Tokura, A Sub-logarithmic Time Sorting on a Reconfigurable Array, *IEICE Transactions on Information and Systems*, E-74-D, Vol. 11, pp. 3894–3901, 1991.
5. K. Nakano, A Bibliography of Published Papers on Dynamically Reconfigurable Architectures, *Parallel Processing Letters*, Vol. 5, No. 1, pp. 111–124, 1995.
6. C. Steckel, M. Middendorf, H. ElGindy, and H. Schmeck, A Simulator for the Reconfigurable Mesh Architecture, *IPPS/SPDP'98 Workshops*, *Lecture Notes in Computer Science*, Vol. 1388, *Springer-Verlag*, pp. 99–104, 1998.
7. B. F. Wang, G. H. Chen, and F. C. Lin, Constant Time Sorting on a Processor Array with a Reconfigurable Bus System, *Information Processing Letters*, No. 34, Vol. 4, pp. 187–192, 1990.