Australian Journal of Basic and Applied Sciences, 6(10): 303-310, 2012
ISSN 1991-8178

Reconfigurable Mesh Simulation

[1]Abdelwadood Mesleh, [2]Mohammad Al-Rawabdeh

[1]Computer Engineering Department, Faculty of Engineering Technology, Al-Balqa' Applied University, Jordan.
[2]STS Payment Network, Jordan.

Abstract: A multi-core processor technology integrates many processing cores on a chip to achieve an increased performance. One strategy to define the relationships between cores is a reconfigurable mesh (R-Mesh). R-Mesh has drawn much attention in the recent years because of its speed and efficiency compared with traditional parallel strategies. The novel contribution of this paper is to introduce a simple R-Mesh simulator that can be used for educational purpose. The purposed simulator can assist in the analysis of algorithms implemented using R-Mesh within an educational environment. We have used the Open Multi-Processing (OpenMP) in MS Visual Studio.Net 2008 (C++) to implement the R-Mesh architecture code part “rmesh.h”, and the Open Graphics Library (OpenGL) to implement the R-Mesh graphics part “rmgraphic.h”. Simulation of a variety of problems (addition, counting, and sorting) shows the robustness of the proposed R-Mesh simulator.

Key words: Reconfigurable mesh; Simulation; parallel processing; OpenMP.

INTRODUCTION

Future technology (Giefers, H. and M. Platzner, 2007) will enable us to increasing the number of cores integrated on a single chip. The main challenge is how we can interconnect these cores in an effective way to increase processor performance in term of minimum power. The best choice is using R-Mesh which is an array of processors with an underlying bus system that traverses the processors using packet-switched technique (Giefers, H. and M. Platzner, 2007). Recently, Reconfigurable computing is potentially becoming a new phenomenon and it is becoming apparent that hardware code acceleration will soon become the norm rather than the exception. Reconfigurable architectures are potential solutions that are able to cope with the increasing complexity found in embedded architectures. Traditionally, computing was classified into general-purpose computing performed by a General Purpose Processor (GPP) and application specific computing performed by an Application Specific Integrated Circuit (ASIC). As a trade-off between GPP and ASIC, reconfigurable computing combines the advantages of them.

R-Mesh can give us greater computational power; in addition it can solve many more problems in constant time, which lead to significant reduction in computation times (Steckel, C., 1998). Accordingly studying R-Mesh might introduce new approaches for the design and programming of many-cores.

There is a lack of programming environments for R-Mesh (Maresca, M., 1993). In (Maresca, M., 1993), an architecture, the Polymorphic Processor Array (PPA), uses a strict SIMD programming model to simulate a very restrictive variant of the R-Mesh model, limited to only a subset of the entire body of reconfigurable mesh algorithms (Maresca, M., 1993). A language called Polymorphic Parallel C (PPC) (Maresca, M. and P. Baglietto, 1993) is also used as a programming model for R-Mesh.

Based on (Ben-Asher, Y., 1991), an R-Mesh Parallel C (RMPC) language is used to simulate R-Mesh. The R-Mesh simulator RMSIM (Murshed, M. and R. Brent, 1998) simulates RMPC algorithms while allowing for the visualization of bus configurations through LaTeX pictures. Further R-Mesh simulators were presented by (Bordim, J.L., 2002; Miyashita, K. and R. Hashimoto, 2000).

A recent work (Giefers, H. and M. Platzner, 2009) presented a new language ARMLang for the specification of lockstep programs on regular processor arrays, in particular R-Meshes, where the target is real implementations using FPGA.

R-Mesh (Vaidyanathan, R., J. Trahan, 2004) is a set of processors ordered in linear or multidimensional arrays. Each processor connects to its neighbors by external bus. The ports of processors are connected together by internal bus. Each processor in R-Mesh architecture has its own local data memory. And all processors share a code memory - single instruction multi data model (SIMD). There are two main types of R-Mesh (Vaidyanathan, R., J. Trahan, 2004):

(i) A one-dimensional R-Mesh with N processors (see Fig. 1) allows each processor to either segment or not segment the bus traversing. Each processor has two ports, the East and West ports (abbreviated E and W), that connect it to its neighbors. In addition to these external connections between processors, each processor can internally connect its ports to form a bus that passes through the processor. Indeed, if all processors connect

Corresponding Author: Abdelwadood Mesleh, Computer Engineering Department, Faculty of Engineering Technology, Al-Balqa' Applied University, Jordan.

their E and W ports, a bus spans the entire linear array. The internal connections in a processor of a one-dimensional R-Mesh are viewed as a partition of the set, {E, W}, of ports of the processor. The two possible partitions here have the following meanings: { $\overline{WE}$ } indicates that the E and W ports are connected within the processor, on the other hand, { $\overline{E}, \overline{W}$ } indicates that the E and W ports are not connected within the processor.

(ii) A two-dimensional R-Mesh arranges processors in an R x C 2-dimensional mesh (R is number of rows, and C is the number of columns in the 2-dimentional mesh), where each processor has four neighbors. Assuming that each processor has four ports, N, S, E, and W, through which it connects to processors (if any) to its North, South, East, and West. Besides these external connections, each processor may establish internal connections among its ports that correspond to partitions of the set, {N, S, E, W}, of ports. Fig. 2 shows a structure for a 3 x 5 two-dimensional R-Mesh (Vaidyanathan, R., J. Trahan, 2004).

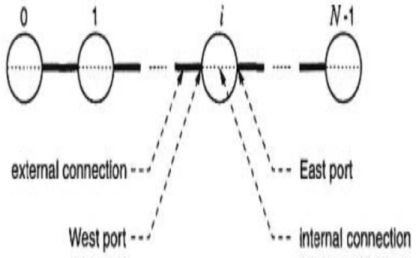


Fig. 1: Structure of One-Dimensional R-Mesh (Vaidyanathan, R., J. Trahan, 2004).

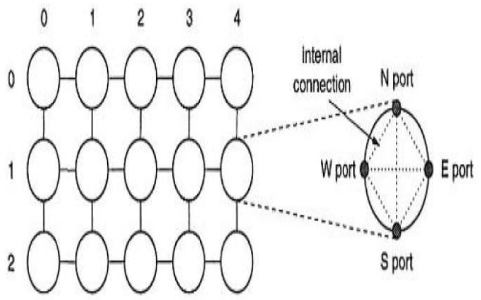


Fig. 2: Structure of Two-Dimensional R-Mesh (Vaidyanathan, R., J. Trahan, 2004).

In this paper, we have designed a simple simulator for R-mesh architecture that implements three types of R-Mesh connections ({N, S, E, W}, { $\overline{EW}$ }, and { $\overline{WS}, \overline{NE}$ }). Figure 3 illustrates the class of R-Mesh architecture for our proposed simulator. Figure 4 shows the 15 possible port partitions along with the corresponding internal connections.

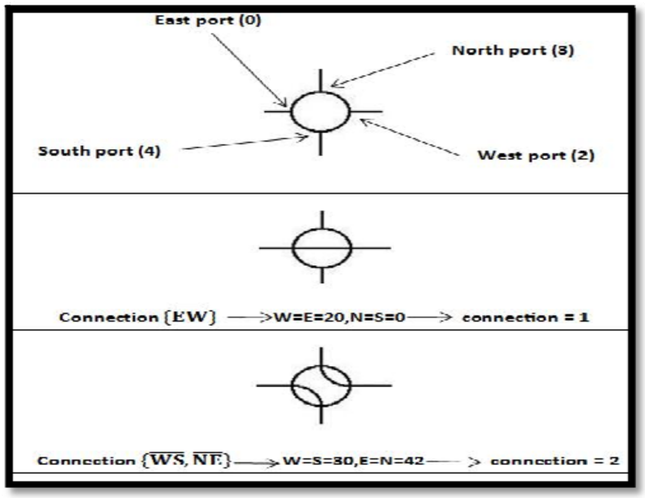

East port (0)
North port (3)
South port (4)
West port (2)
Connection $\{\overline{EW}\} \longrightarrow$ W=E=20,N=S=0 $\longrightarrow$ connection = 1
Connection $\{\overline{WS}, \overline{NE}\} \longrightarrow$ W=S=30,E=N=42 $\longrightarrow$ connection = 2

Fig. 3: Class illustrations of the Simulator (Vaidyanathan, R., J. Trahan, 2004).

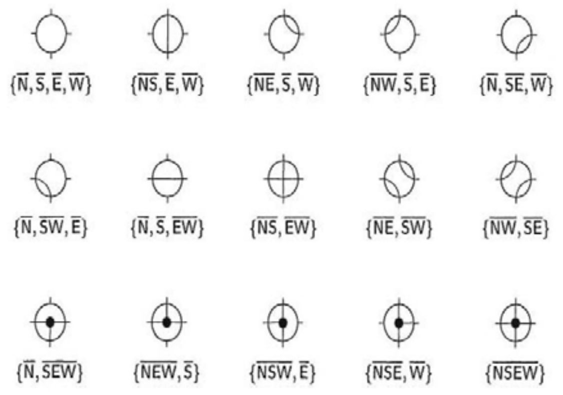

$\{\overline{N},\overline{S},\overline{E},\overline{W}\}$
$\{\overline{NS},\overline{E},\overline{W}\}$
$\{\overline{NE},\overline{S},\overline{W}\}$
$\{\overline{NW},\overline{S},\overline{E}\}$
$\{\overline{N},\overline{SE},\overline{W}\}$
$\{\overline{N},\overline{SW},\overline{E}\}$
$\{\overline{N},\overline{S},\overline{EW}\}$
$\{\overline{NS},\overline{EW}\}$
$\{\overline{NE},\overline{SW}\}$
$\{\overline{NW},\overline{SE}\}$
$\{\overline{N},\overline{SEW}\}$
$\{\overline{NEW},\overline{S}\}$
$\{\overline{NSW},\overline{E}\}$
$\{\overline{NSE},\overline{W}\}$
$\{\overline{NSEW}\}$

Fig. 4: Port partitions of R-Mesh (Vaidyanathan, R., J. Trahan, 2004).

Implementation includes two parts: R-Mesh architecture part and R-Mesh graphic part. R-Mesh architecture part shows the final results of an operation and the values stored in each processor in each execution. R-Mesh graphic part shows the internal and external bus configuration and the status of the processor.

We used C++ language to implement the proposed simulator. We have used the Open Multi-Processing (OpenMP) in MS Visual studio.Net 2010 to implement the R-Mesh architecture code part “rmesh.h” and the Open Graphics Library (OpenGL) to implement the R-Mesh graphics part “rmgraphic.h”. The functions in the two C++ header files (rmesh.h and rmgraphic.h) can be used to implement many algorithms on R-Mesh (such as non-recursive binary tree, chain sorting and counting bit). The main contents of “rmesh.h” are the following:

- R-Mesh class: contains variables and connection functions:

```
class rmesh{
public:
int north, south ,east ,west;
//ports of CPUs
int connection ;
// type of the connection
int busy;
int active;
int next;
int first;
int last;
int link;
int pred;
int next_list;
void connection_EW(){
east=west=20;
north=south=0;
connection=1;
}
// function connect both east &
// South ports and both north
// and west ports
void connection_NW_and_SE(){
west=30;
south=30;
north=42;
east=42;
connection=2;
}
};
```

The main contents of “rmgraphic.h” are the following:

- A variable to define dimensions of R-Mesh, i.e. row x column:

```
int column, row;
```

- A variable to define the R-Mesh size: size= row*column:

```
int size;
```

- A buffer to store the input data:

```
char *pr=new char();
```

- A variable to store the type of connection of each processor:

```
int connection[200];
```

- variables to store data in processors:

```
int d [200];
```

- A variable to define the status of the port of each processor:

```
int port[100];
```

- A variable to store the number of processor:

```
int cpu_num;
```

- A function to input a set of numbers {b0, b1,…., bn-1} and to assign then to processors:

```
void enter_number();
```

- According to executed operation, some functions are designed to draw the paths that connect the ports of each processor for the one/two dimensional R-Mesh architectures and to draw the background of r-mesh:

```
void draw_path();
void draw_path2;
void CALLBACK display(void);
```

In this paper, we introduce a simple R-Mesh simulator that can help students to understand R-Mesh. In addition our simulator can be used to study R-Mesh behavior within different algorithms and to compare the results with the output of traditional technique (serial and parallel). Our simple R-Mesh simulator uses C++ programming language to implement any algorithm on R-Mesh. The proposed simulator enables any student with a C++ basic information and with an R-Mesh basic knowledge to understand R-Mesh, and to compare its performance with other parallel approaches.

2. R-Mesh Simulation Results:

To test the proposed R-Mesh simulator, we have used it to simulate the Non-Recursive Binary Tree Addition (NRBTA), Bit Counting (BC) and Chain Sorting (CS) algorithms.

2.1 Non-Recursive Binary Tree Addition Problem:

Let processor i $(0 \le i \prec N)$ initially holds an input a_i. The Non-Recursive Binary Tree Addition (NRBT) algorithm (Vaidyanathan, R., J. Trahan, 2004) aims to compute $\sum_{i=0}^{N} a_i$.

The following is the consol output results for NRBT implemented on a one-dimensional R-Mesh with 8 processors.

```
Enter (8) numbers:
1, 2, 3, 4, 5, 6, 7, 8
Value of rank 4=11
Value of rank 6=15
Value of rank 2=7
Value of rank 0=3
Value of rank 4=26
Value of rank 0=10
Value of rank 0=36
```

Figure 5 shows the simulation output of the NRBT implemented on the same one-dimensional R-Mesh.

2.2 Bit Counting Problem:

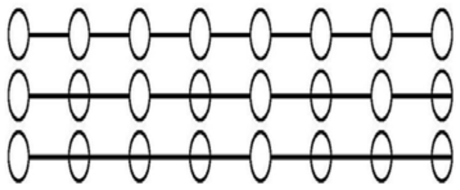


Fig. 5: R-Mesh simulation output of the NRBT algorithm implemented on a one dimensional R-Mesh with 8 processors.

Let processor i $(0 \le i \prec N)$ initially holds an input a_i. The bit counting algorithm (Vaidyanathan, R., J. Trahan, 2004) is to count the number of ones in the processors. The following is the consol output results for the bit counting algorithm implemented on a 4 x 3 two-dimensional R-Mesh:

```
Enter a Binary Number: 101
Matrix
1 0 1
1 0 1
1 0 1
1 0 1
North port is connected with
E E E
- - -
E E E
E E E
East port is connected with
N N N
W W W
N N N
N N N
South port is connected with
W W W
- - -
W W W
W W W
West port is connected with
S S S
W W W
S S S
S S S
Path = 0→3→4→5→8
```

Figure 6 shows the simulation output of the same algorithm implemented on the same 4 x 3 two-dimensional R-Mesh.

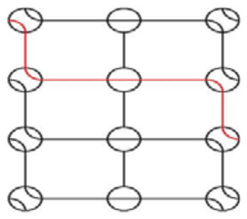


Fig. 6: R-Mesh simulation of the bit counting algorithm implemented on a 4 x 3 two-dimensional R-Mesh.

2.3 Chain Sorting Problem:

Let processor i $(0 \le i \prec N)$ initially holds an input a_i. The chain sorting algorithm (Vaidyanathan, R., J. Trahan, 2004) is to sort a_i values. The following is the consol output results for chain sorting implemented on a 4 x 5 two-dimensional R-Mesh:

```
Enter a Number: 32210
R-Mesh (row=4 X column=5)
Matrix:
1 1 2 2 3
1 1 2 2 3
1 1 2 2 3
1 1 2 2 3
List of active processors;
L(0)= {}
L(1)= {01}
```

```
L(2)= {23}
L(3)= {4}
Active processors:
0 0 0 0 0
1 1 0 0 0
0 0 1 1 0
0 0 0 0 1
First pointers:
Fp(0) = #
Fp(1) = 0
Fp(2) = 2
Fp(3) = 4
Fp(4) = #
Last pointers:
Fp(0) = #
Lp(1) = 1
Lp(2) = 3
Lp(3) = 4
Lp(4) = #
Active processors:
0 1 1 1 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
L= {123}
Link (1) = 2
Link (2) = 1
Link (3) = 1
Pred (3) = 2
Pred (2) = 1
Pred (1) = 3
First (1) = 0
First (2) = 2
First (3) = 4
Last (1) = 1
Last (2) = 3
Last (3) = 4
Next (0) = 1
Next (1) = 2
Next (2) = 3
Next (3) = 4
Next (4) = 0
```

Figure 7 shows the simulation output of the chain sorting implemented on the same 4 x 5 two-dimensional R-Mesh.

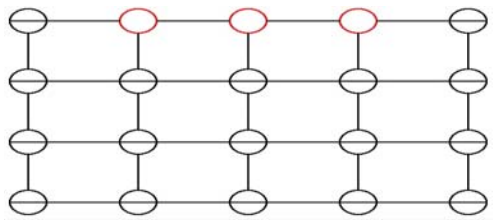


Fig. 7: R-Mesh simulation of the chain sorting algorithm implemented on a 4 x 5 two-dimensional R-Mesh.

3. R-Mesh Performance Comparison Results:

For comparison purposes, we have implemented serial, parallel and R-Mesh schemes for the Logical OR-ing and compared its performance using different performance measures such as the execution time, performance, and speed up (Hennessy, J. and D. Patterson, 2007):

- Execution Time = n x Clock Cycle Time.

- Performance = 1/ Execution Time.
- Speed Up = Old Execution Time / New Execution Time.

Assuming that all processors have the same properties, n is the number of steps, each step requires one clock cycle and Clock Cycle Time = 1 ns. Let processor i $(0 \le i \prec N)$ initially holds an input a_i. The OR-ing algorithm (Maresca, M. and P. Baglietto, 1993) is to compute the logical OR of a_1, a_2, …, a_N.

Figure 8 shows the serial implementation of the OR-ing algorithm, Figure 9 shows the parallel implementation and finally, Figure 10 shows the R-Mesh implementation. The numbers of processors were 8, 16, 32, 64, 128 and 256. Seven iterations are required in each of them.

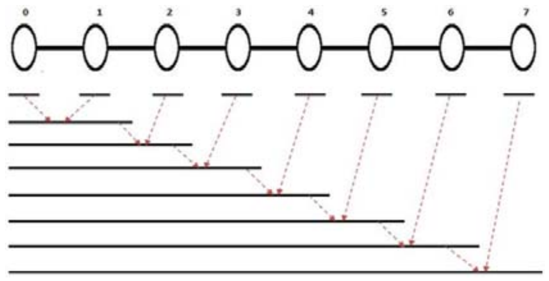
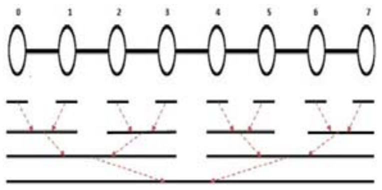


Fig. 8: Serial imlementation of the logical OR-ing algorithm using eight processors.



Fug. 9: Parallel imlementation of the logical OR-ing algorithm using eight processors.

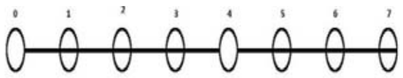


Fig. 10: One-dimensional R-Mesh imlementation of the logical OR-ing algorithm using eight processors.

It is clear that the speed up of the R-Mesh is much faster (see Table 1). For example, R-Mesh is 2.33 times faster than serial implementation when using eight processors.

4. Conclusion and Future Work:

In this paper, we have implemented a simple simulator for R-Mesh. We have used it to simulate the Non-Recursive Binary Tree Addition, Bit Counting, and Chain Sorting algorithms.

Using our simulator, we have conducted a comparison between R-mesh, parallel, and serial implementations for finding logic OR algorithm. Evaluation results reveal the superiority of the proposed R-Mesh in these applications.

Our future work is to extend our simulator to adapt the 15 connections of R-Mesh (see Figure 4), so that students can perform a simulation for any algorithm using the R-Mesh.

Table 1: R-Mesh Speed up compared with Serial and Parallel Implementations.

Number of Processors	Serial	Parallel
8	2.33	1.33
16	5	1.667
32	10.33	2
64	21	2.33
128	42.33	2.667
256	85	3

REFERENCES

Ben-Asher, Y., D. Peleg, R. Ramaswami and A. Schuster, 1991. "The Power of Reconfiguration," Journal of Parallel and Distributed Computing, 13(2): 139-153.

Bordim, J.L., T. Hayashi and K. Nakano, 2002. "An Algorithm Visualization Tool on the Reconfigurable Mesh," VLSI Design, 14(3): 239-248.

Giefers, H. and M. Platzner, 2007. "A Many-Core Implementation Based on the Reconfigurable Mesh Model," in Proc. Int. Conf. on Field Programmable Logic and Applications, FPL'07, pp: 41-46.

Giefers, H. and M. Platzner, 2007. "A Many-Core Implementation Based on the Reconfigurable Mesh Model," in Proc. Int. Conf. on Field Programmable Logic and Applications, FPL '07, pp: 41-46.

Giefers, H. and M. Platzner, 2009. "ARMLang: A Language and Compiler for Programming Reconfigurable Mesh Many-Cores," in Proc. of the Int. Parallel and Distributed Processing Symposium, Reconfigurable Architectures Workshop (RAW). IEEE.

Hennessy, J. and D. Patterson, 2007. "Computer Architecture: A Quantitative Approach," Morgan Kaufman Publishers, fourth edition.

Maresca, M. and P. Baglietto, 1993. "A Programming Model for Reconfigurable Mesh based Parallel Computers," in Programming Models for Massively Parallel Computers, pp: 124-133.

Maresca, M., 1993. "Polymorphic processor arrays," Transactions on Parallel and Distributed Systems, 4(5): 490-506.

Mesleh, A., M. Al-Rawabdeh, O. Al-Heyasat, 2013. R-Mesh Simulator Using C++, Emerging Trends in Computing, Informatics, Systems Sciences, and Engineering, Lecture Notes in Electrical Engineering, T. Sobh and K. Elleithy (eds.), 151: 159-170, DOI:10.1007/978-1-4614-3558-7_13, Springer Science+Business Media, New York.

Miyashita, K. and R. Hashimoto, 2000. "A Java Applet to Visualize Algorithms on Reconfigurable Mesh," in Proc. of the 15th Workshops on Parallel and Distributed Processing, IPDPS '00, pp: 137-142.

Murshed, M. and R. Brent, 1998. "Serial simulation of reconfigurable mesh, an image understanding architecture," Advances in Computer Cybernetics, 5: 92-97.

Steckel, C., M. Middendorf, H.A. ElGindy and H. Schmeck, 1998. "A Simulator for the Reconfigurable Mesh Architecture," in IPPS/SPDP Workshops, pp: 99-104.

Vaidyanathan, R., J. Trahan, 2004. "Dynamic Reconfiguration, Architectures and Algorithms," Kluwer Academic Publishers, New York.