A New Approach for Reconfigurable Massively Parallel Computers

Christophe Bobda, Klaus Danne, Ali Ahmadinia, Jürgen Teich
University of Erlangen, Informatic Institute 12, Am Weichselgarten 3, 91058 Erlangen, Germany
{bobda, ahmadinia, teich}@cs.fau.de

Abstract

We present a new approach for reconfigurable massively parallel computers. The approach uses FPGA as reconfigurable device to build parallel computers which can adapt their physical topology to match the virtual topology used to model the parallel computation paradigm of a given application. We use a case study in which a virtual ring topology is first simulated on a tree topology and then directly implemented in an FPGA configuration. Preliminary results show that we can increase the performance of the parallel computers which make use of message passing interface by a factor of up to 20 % if a reconfigurable topology approach is used.

1 Introduction

Parallel computation is a mean to reduce the computation time of an application by distributing the computation on a set of processors. Each processor computes separately a segment of the given program with a portion of the input data and exchange data with other processors as required in the parallel implementation of the given problem. For data exchange, the processors use an underlying communication network. The set of processors as well as their connection network is called the parallel computer. The topology of a parallel computer is defined by the interconnection network. Trees, toroids, rings, hypercubes, grids and systolic arrays are well known examples of topologies.
The traditional approach in the development of a parallel application involves three steps: *partitioning, communication and mapping* (figure 3 a)). In the partitioning step, the part of the application which can be run independently are identified, extracted and assigned a processor for execution. **Virtual processors** are usually used, since the number of physical processors is limited [2]. The communication step defines how the data exchange among the independent parts will be done. In the third step, a mapping of the virtual processors is done on the set of available physical processors. An assignment of a set of virtual processors to a set of physical processors at a particularly time is called a **processor mapping.**
The best topology required for an efficient modeling of a parallel application is not machine dependent, but application dependent. The quick sort and the vector sum algorithms for example will perform well on a tree topology while the parallel Jacobi iteration [2] will perform well on a ring topology. To be able to model an efficient topology for a given application, the concept of **virtual topology** has been presented in [5] and [4]. A virtual topology defines the connection network of a set of virtual processors. For each application, a virtual topology can be used to efficiently model a parallel computation scheme. As it is the case with virtual processor, the virtual topology has to be mapped on a physical one which is normally fixed. This process is called **topology mapping** or **topology simulation**. Example of systems incorporating the topology simulation are the virtual topology library for PARIX (PARallel extentios to UnIX) [5] the work of Philbin et al [4] and Aysegül [3]. Figure 1 shows an example of topology mapping.

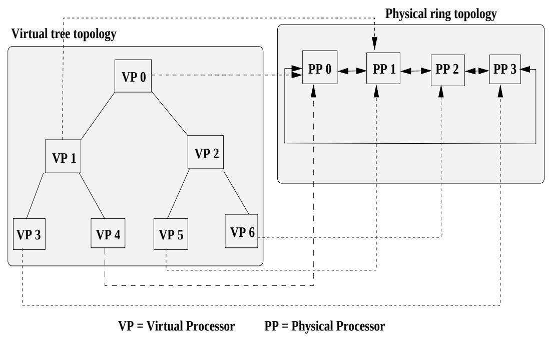


Figure 1. Tree on Ring topology mapping

A virtual tree topology is mapped to a physical ring topology. Topology simulation is likely to decrease the performance of a given application since connections between two virtual processors in the virtual topology may not be available in the physical topology. Therefore the connections have to be simulated. In this case an equivalent route involving many processors has to be found in the physical

topology to simulate the non existing connections. A way to overcome this hurdle would be the use of a **variable physical topology**. With a variable physical topology, the physical interconnection network is modified to adapt the best topology required to compute an application. The concept of reconfigurable topology or reconfigurable massively parallel computer have been presented [4]. The idea behind a reconfigurable parallel computer is to dynamically adapt the physical topology of the machine to match the virtual topology of a given application.
Although the idea of reconfigurable parallel computer was around for a while, less work have been done in building reconfigurable parallel computers. Most of the approaches proposed up to now use a set of processors physically connected as polymorphic torus. The virtual topology is simulated on the underlying mesh. Although this approach can help to reduce the overall computation time of an application, the mapping overhead can be very high. Moreover a certain degree of topology simulation still remain which can jeopardize the performance of the system. In this paper we propose and motivate the use of the field programmable gate arrays to build parallel reconfigurable computers. The first experiment on the RC1000-PP board of Celoxica shows a speedup of the reconfigurable parallel computer using FPGA toward a simulated topology of up to 20 %.
The rest of the paper is organized as follow: In section 2 we present the basic of the FPGAs and how those devices can be used in reconfigurable parallel computers. In section 3 we present the result of our experiment on a case study implemented on the RC1000 Celoxica board. Section 4 concludes the work and gives direction for future work.

2 Implemetation Concept

The straightforward approach in implementing a reconfigurable parallel computer is to use the so called high degree network. In a high degree network, each processor is connected to a great number of processors in the network. The highest degree will be reached if each pair of physical processors is connected. With the highest degree network, it is possible to implement all possible topologies. The main problem with high degree networks is the wiring complexity. For a set of n processors, $n^2 - n$ physical connections are required. For example a machine with 100 processors will need 9900 connections. This approach will be hardly realizable and too expensive with a high number of processors. A comparison of the levels of wiring complexity of high degree networks is given in [4].
Ideally, we will like to have one physical topology to implement a virtual topology. In this case, each algorithm can be parallelized with the help of virtual processors and virtual connections for which physical configuration will be defined. This implementation can be done using a full crossbar switch like the field programmable interconnection device (FPID)[1] or field programmable interconnection chip (FPIC). In this paper, we propose a model of a reconfigurable massively parallel machine based on the use of the Field Programmable Gate Arrays. FPGAs can be viewed as a kind of fully crossbar switch, since any combination of the terminal connection can be realized. However, they offer two main advantages compared to the traditional FPID. First FPGAs are made upon a set of fine grained computing and storage element which can be used to implement some communication protocols or even to carry some precomputation or temporally store the exchanged data. Second they can be partially reconfigured, thus allowing a stepwise move from one topology to the next one.
Today's FPGA are made upon an array of processing elements called logic block or configurable logic block (**CLB**) which can be connected via an array of programmable interconnection elements. In this manner arbitrary interconnections can be realized for the FPGA terminals. Therefore FPGAs are good candidates for reconfigurable topology. For some FPGAs, it is possible to change part of the device while the rest is still active. This property can help for a soft switching between two physical topologies.

2.1 Architecture

The architecture of our reconfigurable parallel computer uses the FPGA as a reconfigurable device. All the physical

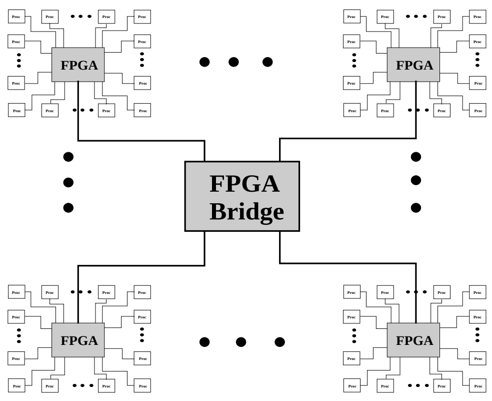
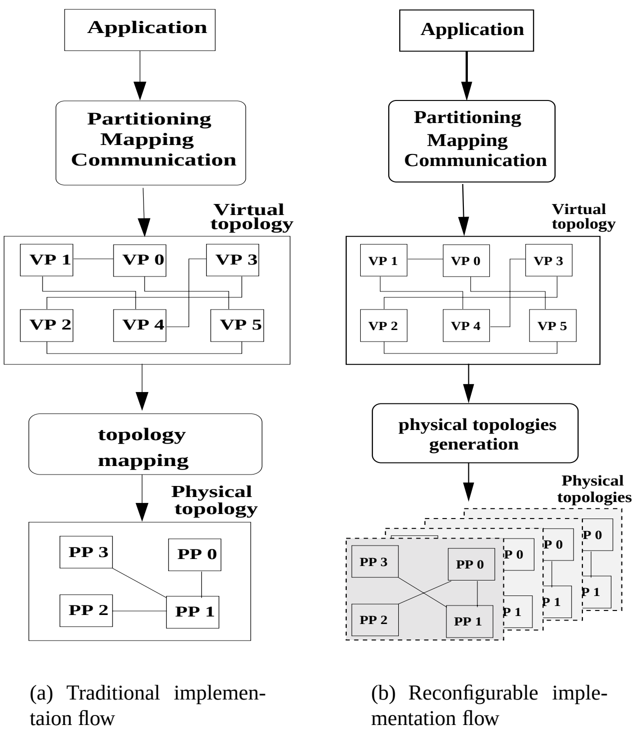


Figure 2. Clusters of Parallel reconfigurable computers connected by a bridge

processors are directly connected to the FPGA-pins. The required connections among the processors are then implemented inside the FPGA as a configuration. Each FPGA-configuration represents a physical topology which can be changed at run-time by reconfiguring the FPGA. With the limited amount of pins available on the FPGA chip, the number of processors which can be connected is also limited. In order to implement a reconfigurable parallel computer with more processors than the number of FPGA-pins can allow, we use the so called bridge. We group processors in clusters (figure 2). Each cluster is connected to an FPGA which locally implement the physical topology of the processors in that cluster. Clusters are now connected using an FPGA bridge. The topology of the parallel computer is determined at each time by the configuration of all the FPGAs in the system.

2.2 Design flow



(a) Traditional implementaion flow

(b) Reconfigurable implementation flow

Figure 3. Parallel Algorithms Implementation

Having a reconfigurable parallel computer as described in the previous section, the implementation of an algorithm will be done as follow: First the algorithm will be partitioned and the communication among the partitions will be defined. Instead of mapping the virtual topology obtained to a physical topology as it is the case in the traditional implementation scheme (figure 3 a)), a set of physical topologies will be generated as configuration bitstreams for the FPGAs (figure 3 b)). Those configurations will then be loaded at run-time to implement the topology required by a computation segment of the application. This can be realized in a framework in which the communication defined by the user is used to generate the configuration bitstream while the application code is compiled for the processors involved in the computation.

3 Case Study

As case study, we consider the implementation of the singular value decomposition (SVD) using a parallel Jacobi iterations as proposed by Brent and Luk [2]. The first implementation used the traditional approach shown in figure 3 a) where the virtual topology is simulated on a physical topology. The second implementation uses our new approach shown in figure 3 b) where the physical topology is generated for the virtual topology. In the two cases, the required physical topology was built inside an FPGA. The Singular

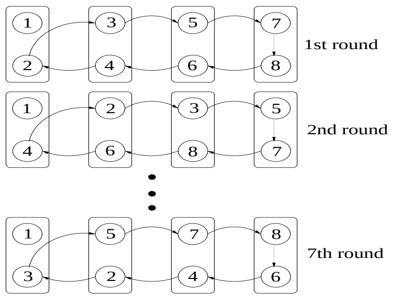


Figure 4. Pairwise distribution

Value Decomposition of an $m \times n$ real matrix A is its factorization into a product $A = U \times \Sigma \times V^T$, where U is an $(m \times n)$ matrix with orthogonal columns, $\Sigma = diag(\sigma_1, ..., \sigma_n)$ and V is an $(n \times n)$ orthogonal matrix. The values σ_i of the diagonal matrix Σ are the singular values of the matrix A, the columns of the matrices U and V are the left and right singular vectors of A. The SVD can be computed using the Hestenes method based on the classical one side Jacobi rotation. The method is a pairwise column orthogonalization. Choosing the column pairs (i,j) on serial machines is usually done according to a fixed cyclic by row ordering: $((1,2),(1,3)...(1,n),(2,3)...(2,n)(3,4)...(n-1,n))$. In this ordering, the clomun pairs can be chosen independently and therefore in parallel. Suject to this, Brent and Luk[2] use $\frac{n}{2}$(n even) processors to orthogonalize the $\frac{n}{2}$ column pairs of an n-column matrix in parallel. Each processor $P_k, 1 \leq k \leq \frac{n}{2}$ is assigned the two adjacent columns $(2k-1, 2k)$ in two variables U_k and L_k . After orthogonalization all processors exchange their left and right most columns. Processor P_k ($k \neq 1$ and $k \neq \frac{n}{2}$) sends the variables L_k and U_k respectively to processors P_{k-1} and P_{k+1} and receives variables U_{k-1} and L_{k+1} respectively from P_{k-1} and P_{k+1}. Processor P_1 permanently maintains variable U_1, sends variable L_2 to processor P_2 and receives from it variable L_4. Processor $P_{\frac{n}{2}}$ receives variable U_{n-3} from processor $P_{(\frac{n}{2}-1)}$, and sends variable U_n. The number of steps needed to complete a sweep in parallel on $\frac{n}{2}$ processors is reduced to $n-1$ (fig.4a). For very large matrices,

it is not possible to have a big amount of processors. We divide the matrix in blocks and assign each physical processor a block (fig.4b). Each processor will then orthogonalize all column pairs in the block assigned to it and exchange the left and right most columns with neighbor processors.
The Brent and Luk parallel implementation presented here define a virtual ring topology. In our experiment, the RC1000 board of Celoxica was used. The board has four external RAMs which can be accessed in parallel. Four processing (PE) elements were therefore implemented inside the FPGA (fig.5). The physical topology of the four PE could therefore be implemented together with the four PEs in an FPGA configuration. Each PE computes separately

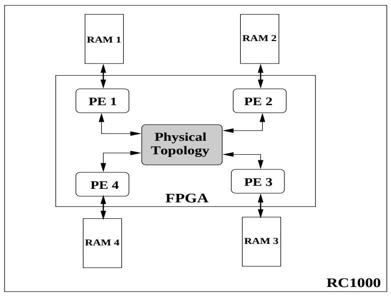
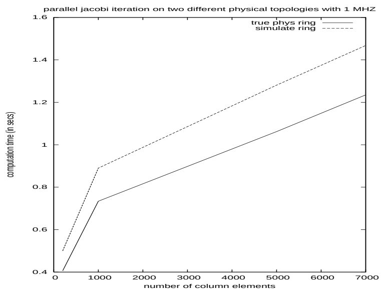


Figure 5. 4-Nodes implementation

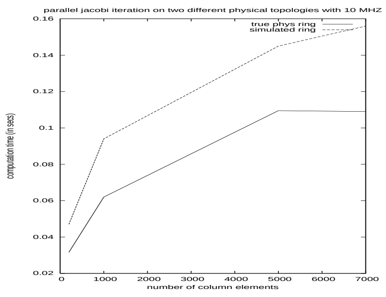
on the segment of the matrix stored in its memory and exchanges the data with its neighbors as explained before. In the first configuration we have simulated the ring topology on a tree topology that was implemented inside the FPGA. In the second case a physical ring topology were directly generated with the processing elements. The computation results are shown in figure 6.

4 Conclusion

In this paper we have presented a concept for reconfigurable massively parallel computers using FPGAs to implement the reconfigurable topology. Trough the implementation of the Jacobi rotations on a real ring topology and the simulation of a ring topology on a tree, we were able to proove the efficiency of our approach. Although the implementation of parallel algorithms using reconfigurable parallel computers presents a lot of advantages, there are a number of questions to answer an challenges to be solved. From our point of view, the most important questions and problem to solve can be formulated as follow: Given an application for which the set of virtual processors used in the modeling is greater than the set of physical processors, how can we partition the set of virtual processors in groups which can be successively implemented on the physical processors? How to realize an efficient data exchange between the processors when switching from one topology to the next one? Can



(a) Jacobi iteration with 1 MHZ FPGA clock



(b) Jacobi iteration with 10 MHZ FPGA clock

Figure 6. Implementation results

partial reconfiguration be used to increase the efficiency of the system? Our future work will concentrate on those issues as well as the implementation of a framework for an automatic generation of topologies given an application.

References

[1] Lattice digital interconnect devices-programmable interface and interconnect in www.latticesemi.com/products/digin.
[2] R. P. Brent and F. T. Luk. The solution of singular-value and eigen-value problems on multiprocessor arrays. *SIAM J. Sci. Stat. Comput.*, 6(1):69–84, 1985.
[3] A. Gencenta and B. Mukherjee. Virtual-topology adaptation for wdm mesh networks under dynamic traffic. *IEEE Transactions on Electronic Computers*, 2002.
[4] H. Li and Q. Stout. *Reconfigurable Massively Parallel computers*. Prentice-Hall, Glasgow, UK, 1991.
[5] U.-P. S. T. Römke, M. Röttger and J. Simon. Efficient mapping library for parix. In *ZUS'95, Workshop on Parallel Programming and Computation, IOS Press*, pages 275–284, 1995.