Reconfigurable Fabric Interconnects

Stamatis Vassiliadis and Ioannis Sourdis
Computer Engineering, TU Delft,
The Netherlands,
{stamatis, sourdis}@ce.et.tudelft.nl

***Abstract*— Using the existing reconfigurable network infrastructure of FPGAs we describe reconfigurable interconnection networks, denoted as *FLUX Networks*. That is networks where the processing elements, forming a parallel system, have interconnects that are explicitly formed (dynamically) by request using reconfigurable fabric, rather than being fixed. We perform several experiments to show the viability of our approach. More precisely, we present examples where the FLUX Networks perform up to 32 times better compared to a rigid fixed interconnect. Furthermore, we show that, based on the data size and the processing element hardware cost, different topologies might be suitable for a single algorithm. The implication of the above is that changing interconnects (dynamically) on demand using reconfigurable fabric could be beneficial.**

I. Introduction

In multiprocessor systems, developed algorithms have in mind an interconnection network. Traditionally speaking, interconnection networks are rigid and often change from one design point to the next. A consequence of the above is that algorithms and software, when ported to a new family of multiprocessor parallel systems, will not scale in terms of performance (at least) and new software development has to be under way if performance is critical. In [1], as means to resolve (alleviate) scalability and portability we introduced the FLUX Networks, where the network configuration is adapted on demand to fit the network to the needs of an application. In this paper, we introduce the use of reconfigurable fabric as an excellent potential for FLUX Networks implementation platform. In our proposal there exists no logical interconnection of the processors, interconnects instead are established (dynamically) on demand by loading the entire network or individual physical connections. We describe some potential implementation and a programming paradigm (extension of [1], [2]) that may allow the interconnects to be fused with traditional and reconfigurable programming models.

The paper is organized as follows: In Section II we present a background discussion. In Section III we present different solutions for reconfigurable networks to change dynamically on demand processing and interconnecting of processors allowing them to adapt to the interconnect demands of software. In Section IV we provide initial experimental data supporting our approach, and finally, in Section V we present our conclusions.

II. Background & FPGA Conjunctures

As indicated earlier, multiprocessor systems are designed based on a specific hardwired interconnect topology. That is, the designer provides the physical structure of the interconnects having in mind a regular network topology such as crossbar, cube, fat-tree, etc. Furthermore, the network structure is fixed and rigid. For example, once the designer fixes the link width, it will remain the same for the entire life time of a parallel system. Additionally, since the physical structure of the network is rigid, the way communications occur may be restricted. Even when it is found that different communication/network schemes will be more beneficial to achieve better performance because of the rigid network restrictions (e.g. fixed buffer space, bus width, etc.) in most circumstances the benefits can not be achieved. Clearly for these circumstances a different physical organization is required and such an organization can not be accommodated by fixed networks. For example, as depicted in Figure 1, an application at time "t" requires a 2D mesh topology, while at time "t+1" the lower processing elements (PEs) need to transfer large amount of data to the middle PE. A fixed/rigid network would not be able to alleviate these communication requirements, resulting in substantial performance drawbacks. On the contrary, in reconfigurable fabrics the interconnects can be reconfigured, e.g. changing the PE router, link width and buffering size, (and possibly the communication scheme and algorithm) to accommodate the communication traffic. In the example of Figure 1, extra links and buffering in the middle PE are added and several PE routers change. In addition, the width of several links changes, that is, critical links become wider (64 instead of 32 bits), while links which are not often used become narrower (8 instead of 32 bits). This way, in FLUX networks the hardware resources are better utilized to facilitate the communication requirements of the current application/program phase and maximize performance.

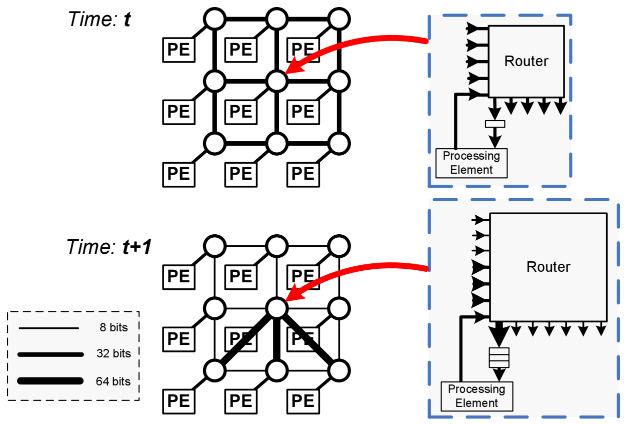

Fig. 1. Adapting interconnects on demand.

This work was supported by the European Commission in the context of the Scalable computer ARChitectures (SARC) integrated project #27648 (FP6).

Obviously, some classes of applications benefit from a specific physical structure. A general purpose parallel system is build however to accommodate a multiplicity of application classes. Given that a provided interconnection network and communication scheme does not fit a pre-specified interconnection mechanism, not all applications can substantially benefit from parallel processing.

To alleviate performance penalties, numerous researchers have provided algorithms of mapping communication networks needed for an application of different interconnections [3]. Considering VLSI chip structures, the current designer practices may not be the most appropriate. Currently, algorithms should be created to suit the multiprocessor system topology in order to maximize performance. Alternatively, we propose *the interconnection network to be provided (dynamically) on demand to fit an algorithm's/program's communication needs*. In order to allow for on demand interconnection networks, connections have to be "adapted". This is possible because reconfigurable technologies have an underlying network that can be "modified". Consequently, it may be of benefit for multiprocessors using reconfigurable fabric, to not commit in advance the underlying network structure into specific interconnects.

In [1] we experimented on rigid physical underlying networks, and showed that based on the underlying network, different mappings are suitable for different algorithms. In the next sections, we answer the following question: *"Can it be beneficial, in reconfigurable fabric, to change interconnection networks (dynamically) on demand depending on the algorithm/program needs?"* We answer affirmatively taking into account the characteristics of current reconfigurable technology and new parameters such as the area cost of the interconnection network and the processing elements.

In ASIC a direct logical connection[1] from a point S to a point D and an indirect connection S to D via a third switching node M can be the same, since they share the same physical interconnections. On the contrary, reconfigurable physical networks allow a multiplicity of interconnects for two points. Consequently, the conjuncture stated above for rigid networks may not hold true for reconfigurable fabrics. We use Xilinx Virtex2Pro and show that the direct connection (Figure 2) is substantially faster (60% of the indirect latency), keeping the distance of the nodes constant. Thus *the physical reconfigurable fabric could match the logical network of an application and possibly improve performance*. Considering that mapping a single edge onto two edges is common to occur when mapping a topology into another, especially when mapping a denser network into a sparser one, our argument becomes even stronger.

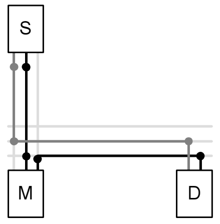

Fig. 2. Direct connection of nodes S,D ($S \rightarrow D$) and indirect connection through node M ($S \rightarrow M \rightarrow D$).

[1]Logical is the network which the application designer has in mind. Physical network is the network available by the designed chip.

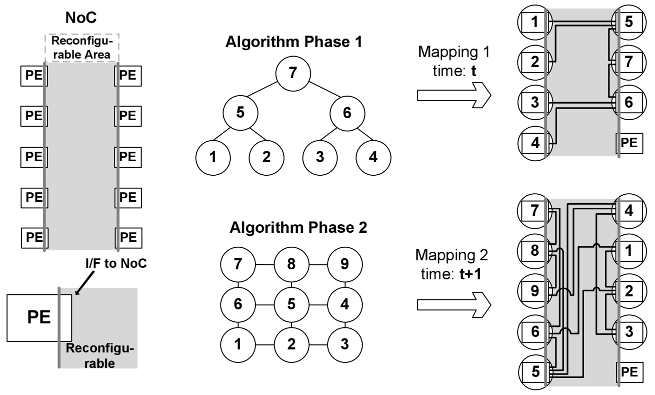

Fig. 3. FLUX Networks on demand with static PE placement.

III. Reconfigurable FLUX Networks on Demand

To improve some of the network-related bottlenecks for parallel processing, we investigate and propose to use the existing reconfigurable fabric on demand rather than statically setting up a logical network in a physical (as performed by the existing systems) and then attempt to map algorithms network necessities on the preexisting network. That is, before (or during) program execution the most suitable network is installed, and consequently is replaced by a different network if it is no longer needed. This is achieved, in difference to existing proposals, explicitly by the program.

Reconfigurable Interconnects: Figure 3 depicts an implementation scheme of our proposal. This multiprocessor system consists of several PEs and a reconfigurable part that can interconnect them in different topologies. For instance, in case of an algorithm implemented for binary-trees (BT), this scheme can connect the PEs in a BT topology. Similarly, for an algorithm that is suitable for a mesh interconnect, the network can be a mesh. Clearly the topologies will follow different physical links to match the logical structure of each algorithm (phase). Obviously, this flexibility is limited by the resources available for the interconnection. This means that the number of the PEs which can be connected in a specific topology depends on the routing resources available (wires and switch boxes). The reconfigurable FLUX networks can change during the execution of a single program. More precisely, if different phases of a program "prefer" different topologies, then the interconnection network could change at run-time. Finally, the PEs can either be statically (hardcores), or dynamically (softcores) placed.

Direct "point-to-point" & Chaotic Interconnects: The FPGA routing architectures provide an underlying "unused" reconfigurable network. That means that a network structure per se may not be needed and processors could be connected on demand at point to point networks if there are available connections (unused routing resources). Figure 4 depicts the way unused wires can be used to connect two PEs additionally to the

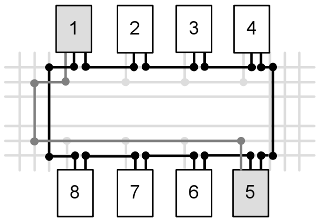

Fig. 4. Direct connections additional to the network topology.

interconnection network. In this example a direct connection between PEs #1 and #5 can be established besides the existing Ring topology due to a critical event. This connection should be *set* when needed and *released* when the data exchange is finished. Alternatively, the interconnections can be build on dynamically established connections (*chaotic network*), if some specific conditions are satisfied, discarding any fixed network topology to directly interconnect PEs based on the communication requests of the application and the available connections. Apart from the complex routing algorithms that this solution requires, a second problem is *timing*. Not knowing in advance the wire length of each connection implies that proper mechanisms are required to guarantee correct communication between the PEs (e.g. GALS).

Multi-Chip Interconnects: The FLUX Network schemes mentioned above can be considered to connect several multiprocessor chips in order to construct a larger system. However, we should note that off-chip interconnection has different characteristics and therefore we need to deal with some additional design and performance issues such as limited off-chip communication bandwidth, due to limited fixed location I/O pins that cannot operate as fast as the on-chip busses.

Technology Considerations: Current technology allows for reconfiguration to be done before program execution. Thus loading an network before program execution is readily available. Regarding dynamic reconfiguration, we first note that a network is used for substantially long time (e.g. scientific applications) in parallel systems that perform massive data operations with the same network requirements. Direct point-to-point and chaotic interconnects could be difficult to implement in current technologies because they require small area and fast reconfigurability not supported by current FPGA technologies [4]. Numerous approaches can be envisioned, however outside of the scope of the paper, to change current commercial chips to incorporate smaller dynamic reconfigurability slides for point-to-point and chaotic interconnects in the near future.

Programming Paradigm for Reconfigurable FLUX Networks: In order for a network to exhibit the properties described above, explicit network calls should be added to the programming paradigm. The FLUX networks allow physical network hardware descriptions to coexist with common programming constructs. Arbitrary interconnection networks can be applied/mapped before program execution or at runtime. A network can be called on demand in reconfigurable technology via explicit calls. To achieve explicit calls we extend the Molen paradigm [2] to support network reconfigurations on demand. Figure 5 illustrates the way `SET Network` instructions activate the network reconfiguration process before or during the program execution. Similarly to the MOLEN programming paradigm, the desired network configuration (bitstream) is downloaded to the reconfigurable unit and subsequently the program execution starts/continues using the newly installed interconnection network.

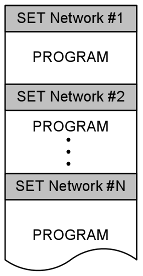

Fig. 5. `SET Network` before or during program execution.

TABLE I

IMPLEMENTATION RESULTS: 255-NODES BT & 256-NODES MESH.

	link width	Routing Area logic cells	Freq. MHz	diameter #cycles(delay)	#nodes	#links	average links/node
8-L BT	32	22,008	254	14 (120 ns)	255	254	2
16×16 mesh	32	77,184	250	30 (55 ns)	256	480	3.3

IV. EXPERIMENTAL RESULTS

In this section, we perform several case studies using Xilinx Virtex2Pro-100 and several parallel algorithms and interconnection networks. We show the area and performance gains when using the most suitable topology, and investigate the parameters that affect the network choice.

Case Study 1: Binary-Tree and 2-D Mesh Implementation. In this case study, we implemented a 256-nodes mesh in Virtex2Pro-100 FPGA, which is the maximum we could fit on the chip (87% occupancy). Then, we implemented an 255-nodes BT. Both use a 32-bit bus width and have the same cycle time. Table I depicts our findings. The main conclusion is that even though the BT has 53% (254/480) total links and 60% (2/3.3) average links per node, it requires for the same number of nodes 22K vs. 77K logic cells, which account 29% of the mesh. The suggestion of the example implementation is the following: one would expect, due to the average total number of links, average number of links per node and number of nodes, the mesh network to require twice the area of the BT. This however, is not the case as it requires almost 4× more area. This implies that for a given area constraint, depending on the network, substantially bigger network than expected could be implemented and vice versa.

Case Study 2: Binary-Tree vs. its 2-D Mesh mapping. In the second case study we evaluate the performance of BT and 2D mesh networks on a BT algorithm. In order to run the BT algorithm into a 2-D mesh we utilize two mapping algorithms with different edge congestion, expansion, and dilation[2] to map the logical BT topology of the algorithm into the 2D mesh physical network. The first one is proposed by Lee and Choi[3] [5] and the second one is the well known H-trees[4] [6]. Subsequently, we evaluate the performance of a 2-D mesh physical topology, compared to a BT physical topology when running the prefix sum algorithm for BTs described in [7]. For a given set $X = x_0, x_1, \ldots, x_{n-1}$ the algorithm calculates the prefix sums $S = s_0, s_1, \ldots, s_{n-1}$, where $s_i = x_0 + x_1 + \ldots + x_i$ for $i = 0, 1, \ldots, n-1$. Figure 6 depicts the total number of cycles required to execute the algorithm for different sizes of sets and number of nodes. Clearly, the BT topology is up to 32× faster, compared to the 2-D mesh (Lee_Choi and H-tree mappings). We can also notice that the processing data affect the performance of the networks. Medium size networks have

[2]When embedding topology A into topology B, edge congestion is the maximum number of A edges, mapped onto any B edge, expansion is the ratio of number of the B nodes to the number of A nodes, and dilation is the maximum number of links in B that any edge of A is mapped onto.

[3]Edge congestion=2, optimum expansion ($\frac{2^p}{2^p-1}$), dilation $\frac{D}{2}+1$ for the edges between the 2^{nd} and 3^{rd} level of the tree, where D is the dimension of a $D \times D$ mesh.

[4]Edge congestion one, expansion asymptotically twice the optimum, and dilation=$\frac{D+1}{4}$.

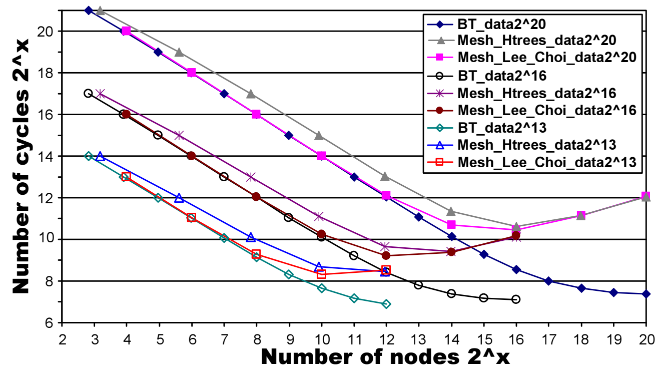

Fig. 6. Performance of the Prefix sums algorithm for different data sizes, number of nodes, and networks.

larger BT-Mesh performance gap when processing smaller data sets. In addition, for larger data sizes and large networks the performance gap is higher. Considering the area results of the previous case study for the BT and the 2-D mesh, we can state the following: given a specific area constraint for an interconnection network of a multiprocessor system, then a 1023-node BT (10-levels) could fit in about the area of a 256-node mesh. In this case, the BT would be 4 to 8× better depending on the data size, even for small or medium systems.

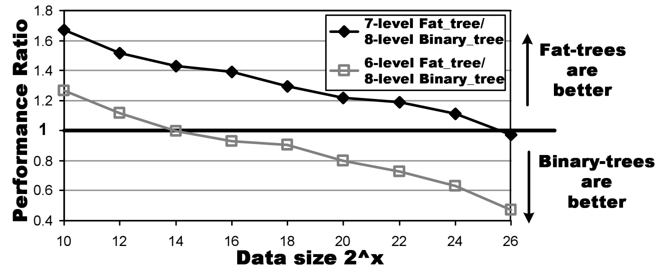

Fig. 7. Performance ratio between a 7-L or 6-L FT and a 8-L BT in sorting.

Case study 3: Sorting in Binary and Fat trees. For this study we implemented the routing structures of an 8-level binary-tree (BT) and 7 and 6-level fat-trees (FT). We chose to implement BTs and FTs of different levels in order to create structures of similar area. However, since the trees do not have the same number of PEs, we need to take into account the area of the PEs in order to fairly compare their area cost. As Table II depicts, the 8-level BT has similar area to the 6 or the 7-level FT, when the PEs are small (<29 logic cells) or large (>357 logic cells) respectively. We evaluate the performance of the sorting algorithm described in [8], including the latency of loading and unloading data for these three networks for different data sizes. Figure 7 illustrates the performance ratio between the 7 and 6-level FT and the BT. The 7-level FT is up to 1.7× better than the 8-level BT, however for large data sets is less efficient. For small data sizes the 6-level FT performs better than the BT up to 1.3×, while for larger data sets it is less efficient and requires 2× the latency of the 8-L BT. In general, as the data set gets larger the performance ratio decreases. That is because the initial sorting in the leaf nodes, becomes the dominant factor compared to the load/unload communication delay. One would assume that

TABLE II

IMPLEMENTATION RESULTS: 8-L BINARY-TREE & 6,7-L FAT-TREES.

	link width	Routing Area Logic Cells	Total Area PE=29LC	Total Area PE=357LC	Freq. MHz	diameter #cycles(delay)	#nodes	#links	max links per node
8L-BT	32	22,008	**29,403**	**113,043**	254	14 (55 ns)	255	254	3
7L-FT	32	67,592	71,275	**112,931**	238	12 (50 ns)	127	384	128
6L-FT	32	27,432	**29,259**	49,923	257	10 (39 ns)	63	160	64

the FT topology is more suitable for sorting than the BT, since FT I/O bandwidth is substantially higher. However, this case study clearly shows that we cannot choose in all cases the most suitable topology according only to the application. There are other parameters that should be taken into account such as the data size, the underlying technology and the architecture of the PEs. This makes our argument stronger, meaning that reconfigurable interconnects can be proved beneficial even when it is not clear in advance which topology is suitable. The above case study indicates that the overall performance depends on the data size and PEs area requirements. *Assuming a certain area constrain on a chip, the same program may demand a different interconnection network, depending on the amount of data it has to operate upon, implying that the networks on demand will have been the correct choice.*

V. CONCLUSIONS

In this paper, we introduced the use of reconfigurable fabrics as an implementation platform for the FLUX networks and discussed some performance potential for parallel applications suitable for different interconnection topologies. We showed that when using the reconfigurable fabric, FLUX networks can be beneficial compared to rigid physical underlying networks. We studied different types of reconfigurable interconnections and presented case studies which suggest that the performance of a parallel algorithm drops when using other topologies than the appropriate one (up to 32×). The implication of the above is that by determining the network in advance and by exploiting network instalments (statically or dynamically) substantial gain can be expected. Reconfigurable FLUX networks can provide the most suitable topology to match the logical structure of an application and maximize performance.

REFERENCES

[1] S. Vassiliadis and I. Sourdis, "FLUX Networks: Interconnects on Demand," in *Int. Conf. on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS)*, July 2006, pp. 160–167.

[2] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The Molen Polymorphic Processor," *IEEE Transactions on Computers*, pp. 1363– 1375, November 2004.

[3] B. Monien and I. Sudborough, "Embedding one interconnection network in another," *In Computational Graph Theory, G. Tinhofer et al. Eds., Computing Supplementa, vol. 7*, pp. 257–282, 1990.

[4] P. Sedcole, B. Blodget, J. Anderson, P. Lysaght, and T. Becker, "Modular Partial Reconfiguration in Virtex FPGAs," in *Proceedings of 15th Int. Conference on Field Programmable Logic and Applications*, 2005.

[5] S.-K. Lee and H.-A. Choi, "Embedding of Complete Binary Trees into Meshes with Row-Column Routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 5, pp. 493–497, 1996.

[6] S. A. Browning, "The Tree Machine: A Highly Concurrent Computing Environment," Ph.D. dissertation, CS Dept., CalTech, 1980.

[7] S. G. Akl, *The design and analysis of parallel algorithms*. NJ, USA: Prentice-Hall, Inc., 1989.

[8] T. H. Cormen, E. Leiserson, Charles, and R. L. Rivest, *Introduction to Algorithms*, ser. Cambridge. Massachusetts: The MIT Press, 1990.