Reconfigurable FLUX Networks

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 present the *reconfigurable FLUX interconnection networks*. That is, networks where the processing elements, forming a parallel system, have interconnects that are explicitly formed by request using reconfigurable fabric, rather than being fixed. We perform several experiments to show the viability of our approach using the existing FPGA infrastructure (Virtex2Pro). We compare the FLUX networks against rigid/fixed networks using synthetic benchmarks. Our experimental results show that reconfiguring the network to suit a given traffic pattern can be up to 2.6 and 5.5$\times$ faster than a rigid mesh and binary tree network, respectively. In addition, the reconfiguration overhead can become negligible, given a traffic load that runs for sufficient time. This clearly shows that, based on the traffic pattern, different network configurations might be suitable. The implication of the above is that changing interconnects on demand could be beneficial.**

I. Introduction

Given that uniprocessor microarchitectures may experience some difficulties to exploit technological advances [1], it can be envisioned that multiprocessors could be the answer to the performance quest. Improvement has been achieved with the technological advances in terms of area (which presumably increases exponentially), delay and chip I/O count (which we postulate increases at best linearly). In the very near future, it is almost certain that the VLSI technology will allow single chip multicore general processors to become feasible (possibly exceeding the order of 10^x, where $x \geq 2$). Multiprocessor multichip parallel systems are not new (e.g. see ILIAC IV [2]) and it will appear that using past multiprocessor experiences and applying them in single chip VLSI implementations will provide a solution to general purpose uniprocessor performance scalability. We note however, that the VLSI design of single chip massive multiprocessors is only one of the challenges and by no means the only one. Being able to fit numerous processors in a single chip, does not necessarily imply that the performance increases substantially. It is well known, that in the past only a small fraction of peak performance has been achieved in parallel systems. There are numerous problems that prohibit top performance achievements. For example, assuming shared memory paradigms, scalability is not guaranteed a priori. In addition, software performance is not "portable". That is, software development for a system at time t may not scale to a system developed at time $t+1$. One of the fundamental reasons, but by no means not the only one, is that software does not "mutate" to take into account new network topologies, while seldom parallel systems use a single network topology from one design point to the next.

In multiprocessor parallel systems, developed algorithms have in mind an interconnection network [3], while, traditionally speaking, interconnection networks are rigid and often (actually usually) the interconnection network changes 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. Currently, algorithms should be created to suit the multiprocessor system topology in order to maximize performance. For decades, researchers study efficient ways to port algorithms of one topology into a different physical interconnect. For example, several researchers discuss embedding one interconnection network into another [4]–[7] and/or use some of the nodes as routing nodes to facilitate such mapping [8]. Alternatively, *the interconnection network can be adapted (dynamically) to fit an algorithm'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 (dynamically) "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 [9], 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 [10], we proposed the use of reconfigurable fabric as an excellent implementation platform for the FLUX networks. In this paper, we investigate the efficiency of the reconfigurable FLUX networks when *dynamically* adapting the network on demand to facilitate the communication needs of the running application/program. In our proposal there exists no logical interconnection of the processors. Interconnections 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 [9], [11]) that may allow the interconnects to be fused with traditional and re-

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

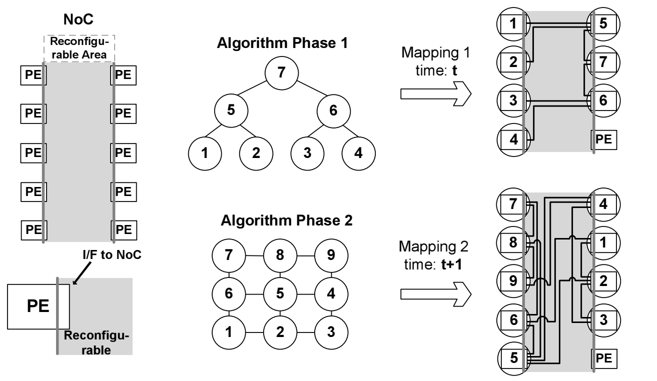


Fig. 1. Reconfigurable FLUX Networks on Demand, with fixed PE placement.

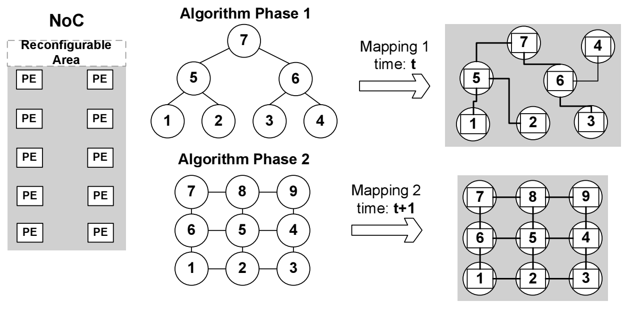


Fig. 2. Reconfigurable FLUX Networks on Demand, without fixed PE placement.

configurable programming models. We provide experimental evidence suggesting that our proposal is promising.

The paper is organized as follows: In Section II we present different solutions for reconfigurable interconnection networks to change dynamically on demand processing and interconnecting of processors allowing them to adapt to the interconnect demands of software. In Section III we provide experimental data and evaluate the efficiency of dynamically adapting the FLUX networks to suit the current traffic pattern. Finally, in Section IV we present our conclusions.

II. Reconfigurable FLUX Networks on Demand

To improve some of the network-related bottlenecks for parallel processing in reconfigurable fabric, 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 algorithm's 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. We describe two different types of reconfigurable interconnects for multiprocessor systems. We first discuss the potential of reconfigurable network topologies, having either static or dynamic processing elements (PE) placement. In the second case, we introduce direct "point-to-point" connections on demand to interconnect processors. That is, the scenario we assume that there is no interconnection network whatsoever and that the connections from one node to another are established directly on demand. Finally, we explain the way the above approaches can be used for multichip multiprocessor systems.

Reconfigurable Interconnects with static/dynamic PE placement: Figure 1 depicts a multiprocessor system that consists of several PEs and a reconfigurable part that can interconnect them in different networks/topologies. For instance, in the 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 interconnection 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 that 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. Consequently, at time t the interconnection topology is a BT and at time $t+1$ changes to a 2-D mesh. However, this run-time reconfiguration of the network introduces overheads. *When run-time reconfiguration is decided, it should be clear that the performance gain, which results from the network switching, is greater than the reconfiguration overhead, otherwise adapting the interconnects will be proven inefficient.*

Each PE consists of two parts, the first part is fixed and executes part of the algorithm, while the second part involves the PE interface to the interconnection network. Since the network is reconfigurable, the interface between the PE and the interconnection network should also be reconfigurable in order to support different networks, network topologies, switching techniques, routing algorithms, etc.. Thus, this latter part of the PE should include a reconfigurable routing module and interface between the variable number of network links and the processor core.

The above approach implies that the processing engines are fixed (statically placed). Static PE placement may restrict the network routing. To overcome this restriction, an alternative solution is that PEs are softcores (Figure 2). In this case, the interconnection topology and the PE placement can change over time (in different phases of an application). Consequently, when a network topology is decided, the PEs will be placed together with the network. However, this approach introduces other overheads. Assuming a hybrid technology, where the PEs would be implemented in ASIC and between PEs reconfigurable hardware would be available for the interconnect, the PEs could operate faster than if they were implemented in reconfigurable hardware. Even if both approaches were implemented in reconfigurable hardware, then in the first case the network reconfiguration process could be substantially faster. That is because in the first case the reconfigured area

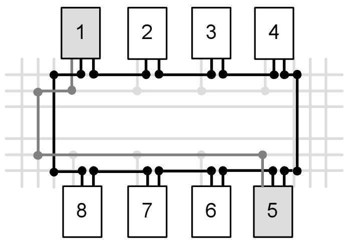


Fig. 3. Direct “point-to-point” connections besides the network topology.

(only the network) is smaller and can be chosen to be partially reconfigured. Furthermore, the PEs can continue running their part of the algorithm without any interruption, while this is not true for the second case.

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 3 depicts the way in reconfigurable hardware unused wires can be used to connect two PEs additionally to the interconnection network. In this example, a direct connection between PEs #1 and #5 can be established besides the existing Ring topology. This connection should be *set* when needed and *released* when the data exchange is finished. That is, if on a specific time “PE 1” needs to communicate with “PE 5” without going via the existing (Ring) network, because of a critical/unpredicted event, then a direct connection is established (and afterwards released) on demand.

The PE interconnections can be build on dynamically established connections (*chaotic network*) if some specific conditions are satisfied. This approach discards 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 (i.e. GALS). Furthermore, a priori analysis of the routing resources is required to determine the maximum communication load that the interconnection network can handle. For each connection request, a specific methodology should be followed: a routing path should be established; then the data should be sent and last, the connection should be released. Having said the above, in case the underlying structure is partially reconfigurable dynamically in acceptable speeds, the point-to-point and the chaotic networks could be of interest.

Multi-Chip Interconnects: We can consider a similar approach to connect several multiprocessor chips in order to construct a larger system. Figure 4 illustrates a multichip multiprocessor system that changes in different time phases. In this case we can apply the same ideas described in the previous subsection. In addition, we should consider 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, since chips have limited fixed location I/O pins that cannot operate as fast as the on-chip busses. Consequently, off-chip interconnection should be carefully designed applying techniques such as Time Division Multiplexing (TDM) to create an efficient multichip multiprocessor parallel system.

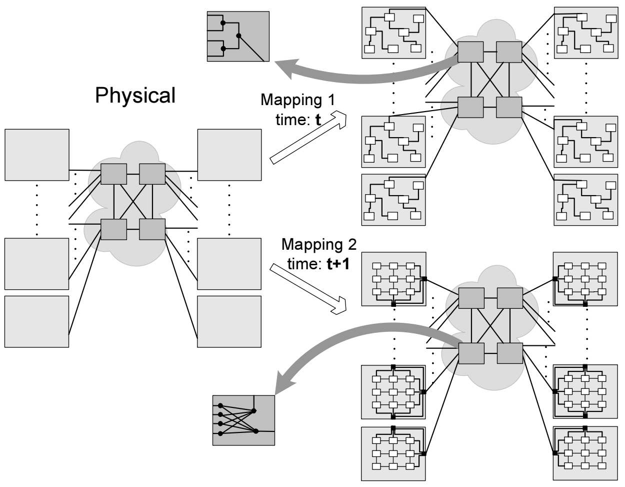


Fig. 4. Off-chip FLUX interconnection network. The left part of the figure depicts the physical structure of the system while the right part illustrates different configurations of the system at different time instances.

Technology Considerations: An interesting question regarding what has been presented is which of the proposed mechanisms can be implemented by currently available technologies and which are the directions for making the remaining mechanism a reality. Current technology allows for reconfiguration to be done before program execution. Thus loading an interconnection 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. Consequently, it can be suggested that interconnects can be dynamically changed with current technology. Direct point-to-point and chaotic interconnects could be difficult to implement in current technologies because they require small area and fast reconfigurability and current technologies such as Xilinx allow partial reconfiguration of areas which may span the entire height of a device and a fraction of one column and require few msecs [12]. This restriction can provide substantial difficulties for point-to-point and chaotic interconnects. Numerous approaches can be envisioned, however, outside of the scope of the paper, to change current commercial chips to incorporate smaller dynamic reconfigurability slides to achieve point-to-point and chaotic interconnects in the near future.

Programming Paradigm for Reconfigurable FLUX Networks: FLUX networks allow physical network hardware descriptions to coexist with common programming constructs. Arbitrary interconnection networks can be applied (or mapped)

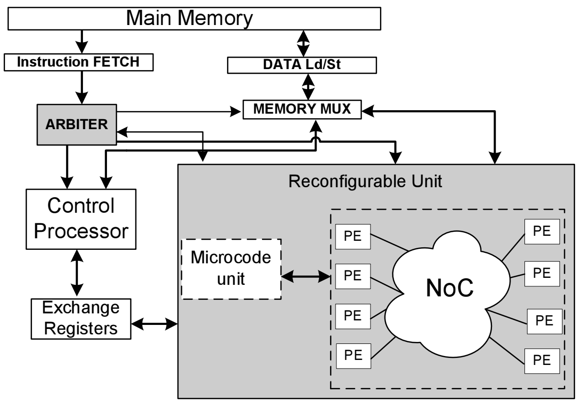


Fig. 5. A FLUX parallel system scheme with reconfigurable FLUX networks.

before program execution or at runtime. They could be predetermined "off-line" at hardware/software co-design stage or detected "on-the-fly". A network can be called on demand in reconfigurable technology via explicit calls. To achieve explicit calls we extend the Molen paradigm [11] to support Network reconfigurations on demand. In the following, we discuss different ways of reconfiguring an interconnection network using the existing ISA extensions of the Molen processor. Generally speaking, the reconfigurable FLUX network mechanism may not require additional ISA support to enforce the intended interconnection network (in case is needed, in the order of 1 or 2 instructions). A control processor is required to (re)configure the interconnection network and download the bitstream of the system. However, it may have an additional role. Assuming a master-slave parallel processing model, the control processor can be the master processor keeping sequential consistency of the program, controlling and synchronizing the PEs, and distributing the workload.

As described in the MOLEN programming paradigm [11], when it is needed to configure the entire network, then a SET $< address >$ instruction is necessary. The SET instruction utilizes an address to a memory location where the first element of the configuration bitstream is to be loaded from. This way, numerous different network configurations are allowed to be available in the configuration memory. The bitstream may include the configuration of the entire interconnection network or part of it (the partial set P_SET $< address >$ Molen instruction). Furthermore, the PEs configuration (including routing information) and possibly the initial data of each local PE memory (instructions and data etc.) may also be part of the bitstream. Figure 5 illustrates a possible reconfigurable FLUX network organization using a control processor. The control processor manages the reconfiguration of the reconfigurable multiprocessor system. An arbiter detects the SET instructions and subsequently activates the reconfiguration process utilizing the microcode unit. The bitstream is downloaded from the memory to the reconfigurable unit through the data load/store unit and the data memory multiplexer. When the reconfiguration is accomplished the microcode unit sends a signal to the arbiter and the following instructions are sent to the control processor in order to continue the execution of the remaining program. Finally, the synchronization of the PEs can be accomplished through the exchange registers bank and the MOVTX and MOVFX Molen instructions.

Figure 6 depicts an example of how the "SET" instruction (either partial or complete) can be executed with program routines in a sequentially consistent programming model. The "SET" instruction configures the network or the entire system before the execution of the next phase of a program. The SET instruction can point to an address in memory where the configuration bitstream of the new network is stored.

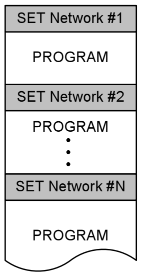


Fig. 6. `SET Network` before or during program execution.

Current synthesis and place & route tools might require significant amount of time to generate a bitstream for a design. Therefore, the bitstream of the interconnection network can be generated only during the compile time of an application, statically. However, a interconnection network could be automatically generated on the fly, and in case the tools allow a fast bitstream generation, then the "SET" instruction could perform a reconfiguration of a non-predefined interconnection. Consequently, the "SET" instruction would include as parameters the characteristics of the desired network. Apart from the PEs configuration (data and subprograms, if necessary), these parameters could include (without excluding others) the following:

- Switching techniques.
- PE routing algorithm (including routing policies, protocols etc.)
- PE addresses (PE numbering) and how they are connected with each-other (topology). Preferably this assignment is performed in a regular and extendable fashion, in order to be scalable for different number of nodes.
- Link width (word width).
- Number of PEs, etc.

What is new in the reconfigurable FLUX Networks: Our proposal differentiates from previous approaches such as [7], [8] in at least one of the following items:

- Our proposal allows the network configuration to coexist with common programming constructs via explicit calls of the physical network. Furthermore, the programmer and the designer have *complimentary* roles. The first chooses the desired topology while the second is responsible for the way this is going to be implemented on a physical reconfigurable infrastructure.
- Contrary to others [8], reconfigurable FLUX Networks can reconfigure the PE routers, changing switching techniques, routing algorithms, number and width of the links, etc. on demand instead of being prefixed.
- Reconfigurable hardware has a *unique* characteristic. Physical connections can match the logical connections of an application and support additional direct point-to point

TABLE I

IMPLEMENTATION RESULTS: 63 AND 127-NODES BT & 64-NODES MESH.

	link width (bits)	Routing Area (logic cells)	Freq. (MHz)	Diameter (delay ns)	Bisection Width	#nodes	#links	average links×2/node
127-node BT	32	38,712	200	12 (240 ns)	1	127	126	1.98
63-node BT	32	19,006	200	10 (200 ns)	1	63	62	1.97
64-node mesh	32	47,147	200	14 (280 ns)	8	64	112	3.5

connections not foreseen by the algorithm developer.

- Our approach can *dynamically* adapt to *arbitrary* topologies, while other solutions can only support *several* topologies and regular predefined structures [8].
- Other solutions require a second network to configure the switches, besides the one that interconnects the PEs [8], while our approach does not.
- In reconfigurable hardware, we set *raw* connections and the configuration time can be relatively small if a fine grain configuration can be supported by the technology. Furthermore, there is no local memory under each switch, to store the possible configurations for *every* supported topology. Other solutions employ routing elements to “reprogramme” the local memory introducing delay [8]. In essence, such networks are *programmable* rather than *reconfigurable*, adding extra interconnection overhead and delays.
- Router-based approaches are constructed by a network and actually “routers” [8], while the reconfigurable FLUX network is a network plus switching elements. Given that the “routers” are slower than the FPGA one “pass-transistor” crossbar delay, our approach can be more efficient and faster.

III. EXPERIMENTAL RESULTS

In this section, we provide evidence suggesting the viability of our proposal. We experiment using Xilinx Virtex2Pro-50, several synthetic benchmarks and interconnection networks. More precisely, we implemented a 2-D mesh and a binary tree network, and inject traffic patterns composed by several different phases more suitable for one of the above networks. Subsequently, we investigate whether reconfiguring the network to suit the traffic load is more beneficial than using rigid interconnects. We first present the design and implementation of our networks, and then describe the generated benchmarks. Finally, we present the performance results of the mesh, binary-tree and FLUX networks, and compare it with the theoretical best case of always using the best network without any configuration overhead.

Networks Design and Implementation: As indicated earlier, we designed and implemented a binary tree and a 2-D mesh network. The size of the networks is the maximum that can fit in a Virtex2Pro-50 FPGA device. We implemented a 64-node mesh which occupies 99% of the device and a 127-node BT which requires about 80% of the available resources even though has almost double number of nodes. Figure 7 depicts the datapath of a mesh node (BT nodes are similar having four instead of 5 ports). The left input and right output

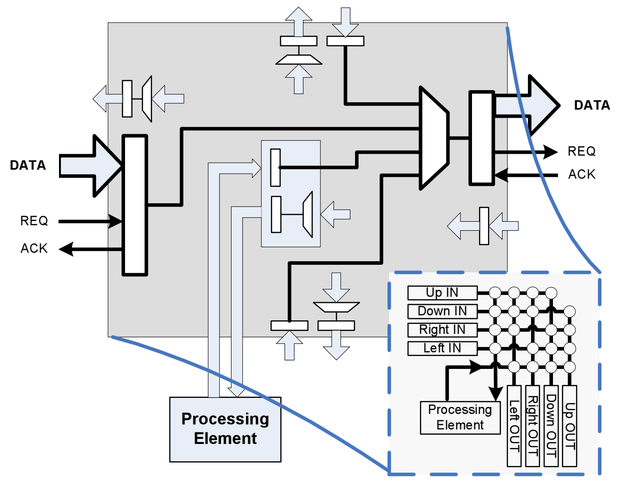


Fig. 7. Datapath structure of a mesh node. The left input and right output are illustrated in detail. There are 5 input/ouput ports (left, right, up, down and PE port). The connections between the inputs and the outputs are depicted on the bottom-left part of the figure.

are illustrated in detail. Additionally, the connections between the inputs and the outputs are depicted on the bottom left part of the figure. Both networks have 32-bit links and single flit (32 bits) input and output buffers. Packets are routed using wormhole routing [13]. The routing algorithms are *minimal*, that is, always a minimal path is followed, and *deterministic* in order to avoid deadlocks in the case of the 2D mesh [13]. The handshake between an output and an input is achieved using request-acknowledgment protocol in order to stall the flow of packet flits in case of contention. Consequently, each flit requires two cycles to be transferred to the next input or output buffer (one cycle REQ and one for ACK). The header of each packet requires one cycle to be decoded in the input and find its destination output, while a second cycle (at least) is needed to check whether the output is busy and transfer the flit to the output buffer. Finally, each output serves input requests in a round-robin fashion. Table II depicts the implementation results of the 64-node mesh the 127-node BT and also a 64-node BT. All networks can operate at 200 MHz. A 64-node mesh requires about 2.5× more resources than the 64-BT and 25% more than the 127-node BT. Finally, a 2D mesh node occupies about 750 logic cells, while a BT (not leaf or root) node requires about 550 logic cells.

Mapping a Binary-Tree into 2D Mesh and vice versa. As mentioned in Section I, rigid interconnection networks require mapping algorithms to map the logical network[1] into

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

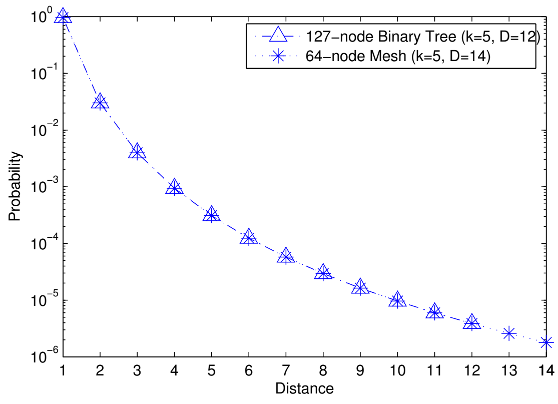


Fig. 8. Zipfian Distribution for k=5, and D=12 and 14.

the physical one, when they do not match. Therefore, in order to fairly compare rigid and FLUX networks, the former ones need to employ mapping algorithms to port a traffic pattern that does not match the network. We map the BT traffic into the mesh network utilizing the Lee & Choi algorithm [14]. However, the BT network has almost double number of nodes than the mesh. In order to resolve this problem, we map two BT nodes (based on the algorithm) into each one of the 63 mesh nodes, while the BT root node is mapped into the 64^{th} mesh node. We assume that each mesh node can handle the processing of two BT nodes and sequentially send their communication requests. This is the best case scenario for the rigid mesh network. Mapping a mesh into a BT is a NP-complete problem [7]. We invert the BT-to-Mesh mapping in order to map the mesh network into the BT. We assume that the mesh tasks cannot be parallelized in more than 64 nodes and consequently, only half of the 127-node BT is utilized.

Synthetic Benchmark. We generate different traffic patterns suitable for the 127-node BT and the 64-node mesh. Our synthetic benchmarks are composed by multiple phases (10, 100 or 1,000) of these traffic patterns. A new phase of requests cannot start before all the packets of the previous phase have reached their destinations.

We describe, next, the way we generate phases of 10,000 requests (packets) per node, suitable for BTs or meshes. In order to generate a traffic pattern suitable for a given topology X, a packet should have higher probability to have a destination where the minimal path is short (distance in terms of hops), having in mind the topology X. We propose a Zipfian distribution model for the packet distance, described by the following equation:

$$Distance(i) = Zipf(i) = \frac{i^{-k}}{\sum_{j=1}^{D} j^{-k}}$$

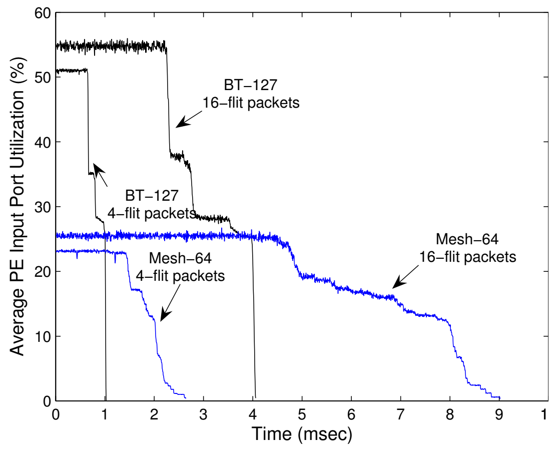
where $i = 1,\ 2,\ \ldots, network\ diameter$.

Consequently, the Zipf equation determines the probability for a packet to have a destination of distance i (number of hops). When the distance for a packet is decided, then, a destination of this distance is uniformly-random chosen. Figure 8 depicts the distance distribution for the 127-node BT and the 64-mesh (diameter D= 12 and 14 respectively), when $k = 5$. In both BT and mesh traffic patterns the probability of a packet to have a destination node one hop far from the source is over 95%. We generate benchmarks of either 4 or 16-flit packets (16 and 64 bytes). Each benchmark contains 10, 100, or 1,000 phases (each phase has 10,000 packets) suitable for meshes and BTs. The ratio of [Mesh:BT] phases varies between $[0:10,\ 1:9,\ldots\ x:10-x,\ldots\ 10:0]$. Finally, note that all benchmarks have all mesh-like phases first and BT-like phases next.

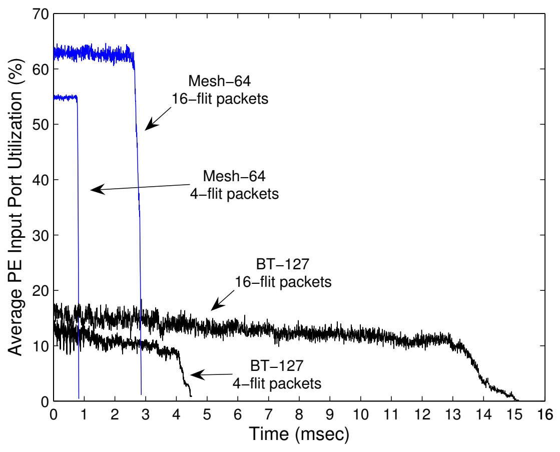
BT & Mesh Evaluation For Various Traffic Patterns. Before we evaluate the FLUX Networks performance compared to the rigid interconnects and analyze their reconfiguration overhead, it is essential to analyze the behavior of the BT and mesh networks on the traffic patterns described above. Figure 9 depicts the percentage of the Mesh and BT PEs which are busy receiving packets over time. The traffic patterns are either BT or mesh-like of Zipfian distributions of k = 5 and contain either 4 or 16-flit packets. When the traffic pattern is BT-like, we consider 127 PEs in both networks. For the BT network the mapping is one-to-one, while in the case of the mesh-64 the 127 BT PEs are simulated/mapped into the 64 mesh nodes and therefore the workload of about two BT PEs on average is handled by a single Mesh PE. Similarly, for the mesh-like traffic only 64 PEs are considered in both networks.

Clearly, for BT-like traffic, the BT-127 PEs are more than twice as busy receiving packets compared to the Mesh-64 PEs (50-55% vs. 24-26% peak utilization for 4 and 16-flit packets). Therefore, the BT requires less than half the time to complete the BT-like workload/phase compared to the mesh (1 and 4 msec vs. 2.6-9 msec for 4 and 16-flit packets respectively). When mesh-like traffic is injected in the two networks the performance drawback for the BT is substantially higher than in the previous case for the mesh network. The BT PEs input ports are clearly under-utilized compared to the mesh network (about 5-6$\times$ lower) and therefore require about 5.5$\times$ more time to complete a single mesh-like phase for both 4 and 16 flit packets. Finally, notice that the peak utilization does not exceed 56% and 66% for BT and mesh respectively. That is due to the irregular structure of the networks. For example, two BT leaf nodes, which have the same parent node receive only about $\frac{2}{3}$ ($\frac{1}{3}$ each) of the packets sent by their parent node. Similarly, the "border" nodes of the mesh receive less traffic than the internal ones and therefore reduce the overall throughput of the network.

Performance Evaluation. In our experiments, FLUX network can be configured as a 2-D mesh or a BT to suit the traffic pattern. We evaluate the performance of the FLUX networks, the 64-node mesh and the 127-node BT in terms of latency. The synthetic benchmarks, described above, are injected in the networks. Packets leave their source node sequentially without any intermediate delay. Therefore, the traffic load injected in the networks is maximal, *given the packet distribution and size*. Figure 10 depicts the performance results of the three networks for benchmarks of 10, 10^2 and 10^3 phases (of 10^4



(a) BT-like traffic (k=5)



(b) Mesh-like traffic (k=5)

Fig. 9. Percentage of the Mesh-64 and BT-127 PEs being busy receiving packets over time. The traffic patterns are either BT or mesh-like of Zipfian distributions of k = 5.

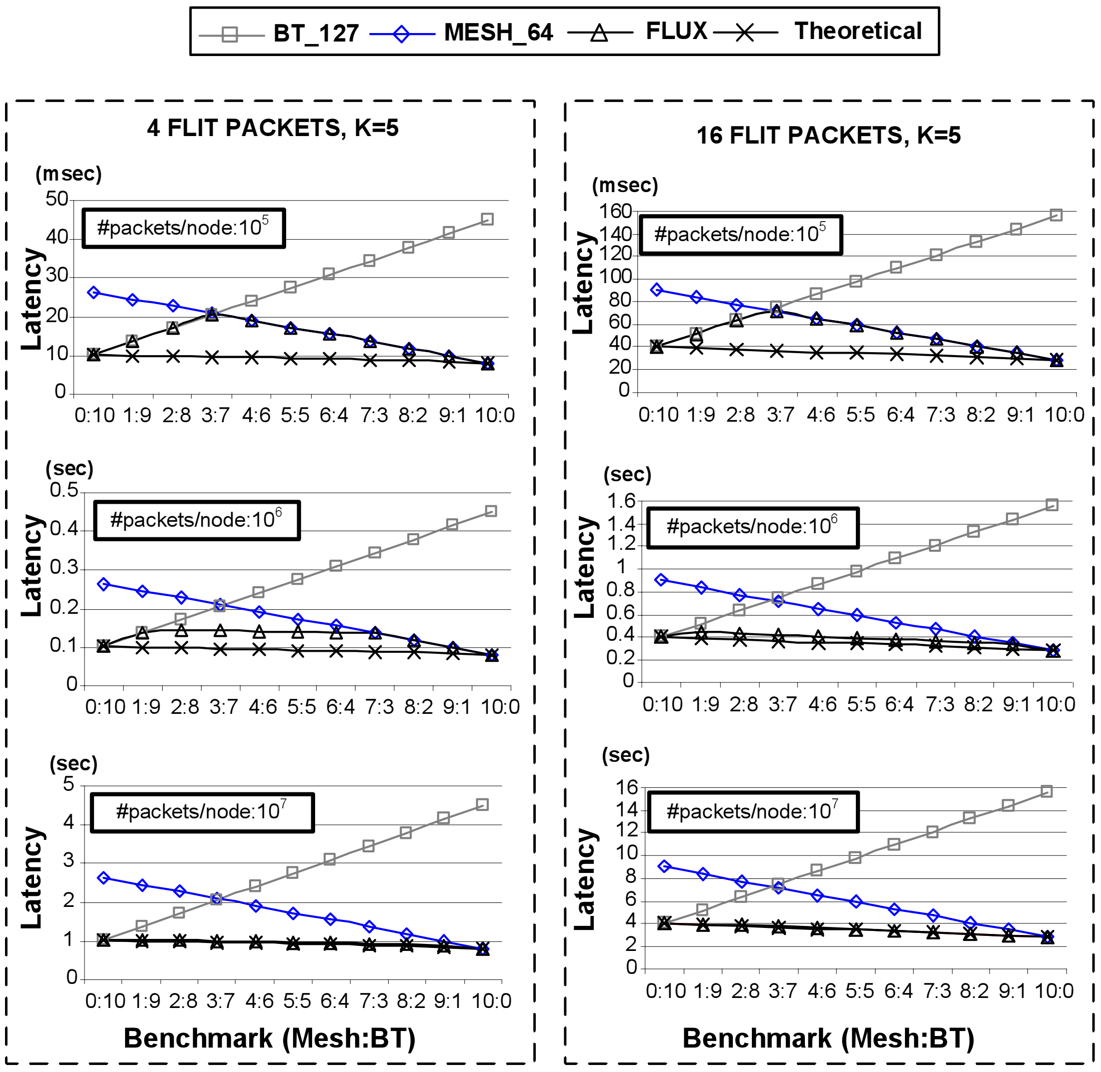


Fig. 10. Latency of the Binary-tree, 2D-Mesh and FLUX Networks for several Mesh/BT synthetic benchmarks.

packets per node), and packets of 4 and 16 flits. Additionally, the theoretical latency of a network that always chooses the best network setup without any configuration overhead is depicted. There are benchmarks that contain only BT-like traffic ([0 : 10]) both mesh and BT-like ([x:10-x], $\frac{x}{10}$ mesh-like and $\frac{10-x}{10}$ BT-like traffic), and only mesh-like traffic ([10 : 0]).

The FLUX Network is up to 2.6× and 5.5× better than the mesh and BT, respectively, for traffic loads of 4-flit packets. When the packets contain 16 flits, it is 2.2 and 5.4× faster compared to the mesh and BT networks. That is, due to the fact that each packet requires 4× more cycles (vs. the 4-flit packets) to arrive from the input of the destination node to the destination PE. Consequently, even if a packet has arrived to a destination node input, competing for the same destination PE is harder and some of the dilation[2] overhead is hidden. Network reconfiguration takes 47.55 msec in Virtex2Pro-50 [15]. Consequently, it is not efficient to reconfigure the FLUX network for benchmarks of 10^5 packets/node, since the overall latency is a few tens of msec (10-40 msec). In these cases, the FLUX network chooses to operate as a mesh or a BT and achieves at least 47% and 52% of the theoretical performance for 4 and 16-flit packets respectively. On the contrary, it is efficient to reconfigure the network on the fly for benchmarks of 10^6 packets/node. Especially, in the case of 16-flit packets, where the total latency is 400 msec, the FLUX network reaches at least 86% of the theoretical performance. For 4-flit packets, reconfiguration is efficient for the [2 : 8] to [6 : 4] benchmarks, and at least 65% of the theoretical performance is achieved. The network is also reconfigured, for benchmarks of 10^7 packets/node, and the FLUX network performance is in worst case the 95% and 98% of the theoretical one for 4-flit and 16-flit packets respectively. Figure 11 depicts a comparison between FLUX networks and the theoretical latency for different traffic loads. As the benchmarks contain more packets per node the FLUX network latency asymptotically approaches the theoretical one. Additionally, for larger packets the latency is closer to the theoretical. The suggestion is that, *the reconfiguration overhead is negligible, when a traffic load of specific characteristics runs for sufficient time*.

IV. Conclusions

In this paper, we presented the reconfigurable FLUX networks and investigated the efficiency of dynamically modifying the network configuration to suit the communication traffic. We showed that mapping one topology into another can result in substantial performance drawbacks that can be overcome only if interconnects are adapted into the desired topology. Our experiments show that the performance of a network drops when the topology is other than the appropriate one (up to $2.6-5.5\times$ higher latency). In addition, the reconfiguration overhead of the FLUX networks can be insignificant given that a specific traffic pattern runs for sufficient time. Finally, reconfigurable FLUX networks can asymptotically approach the theoretical latency of a network that always provides the most suitable topology without any reconfiguration overhead. 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.

[2]When embedding topology A into topology B, edge congestion is the maximum number of A edges, mapped onto any B edge.

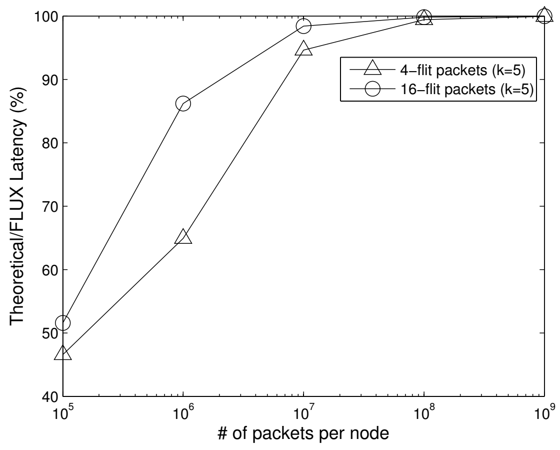


Fig. 11. Comparison between FLUX Networks and theoretical latency for different number of packets per node and packet sizes.

References

[1] S. Vassiliadis, L. A. Sousa, and G. N. Gaydadjiev, “The Midlifekicker Microarchitecture Evaluation Metric,” in *Proceedings of the IEEE Int. Conf. ASAP05*, July 2005, pp. 92–97.

[2] W. Bouknight, S. Desenberg, D. McIntyre, J. Randall, A. Sameh, and D. Slotnick, “The Illiac IV system,” *Proc. IEEE*, vol. 60, April 1972.

[3] I. Lee and D. Smitley, “A Synthesis Algorithm for Reconfigurable Interconnection Networks,” *IEEE Trans. Computers*, vol. 37, no. 6, pp. 691–699, 1988.

[4] F. T. Leighton, *Introduction to parallel algorithms and architectures: array, trees, hypercubes*. CA, USA: Morgan Kaufmann Inc., 1992.

[5] S. Ranka and S. Sahni, *Hypercube Algorithms for Image Processing and Pattern Recognition*. New York City, NY: Springer-Verlag, 1990.

[6] M. Reingold, J. Nievergelt, and N. Deo, *Combinational Algorithms: Theory and Practice*. New Jersey: Prentice-Hall, Inc., 1977.

[7] 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.

[8] L. Snyder, “Introduction to the Configurable, Highly Parallel Computer.” *IEEE Computer*, vol. 15, no. 1, pp. 47–56, 1982.

[9] 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.

[10] S. Vassiliadis and I. Sourdis, “Reconfigurable Fabric Interconnects,” in *International Symposium on System-on-Chip (SoC)*, November 2006.

[11] 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.

[12] 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.

[13] L. M. Ni and P. K. McKinley, “A survey of wormhole routing techniques in direct networks,” *Computer*, vol. 26, no. 2, pp. 62–76, 1993.

[14] 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.

[15] Xilinx, “Virtex-II pro Platform FPGA Handbook.”