A Reconfigurable Distributed Computing Fabric Exploiting Multilevel Parallelism

Charles L. Cathey, Jason D. Bakos, Duncan A. Buell
Dept. of Computer Science and Engineering
The University of South Carolina
{catheyc, jbakos, buell}@cse.sc.edu

Abstract

This paper presents a novel reconfigurable data flow processing architecture that promises high performance by explicitly targeting both fine- and course-grained parallelism. This architecture is based on multiple FPGAs organized in a scalable direct network that is substantially more interconnect-efficient than currently used crossbar technology. In addition, we discuss several ancillary issues and propose solutions required to support this architecture and achieve maximal performance for general-purpose applications; these include supporting IP, mapping techniques, and routing policies that enable greater flexibility for architectural evolution and code portability.

1. Introduction

Most widely-used architectures for high-performance computing consist of shared-memory multiprocessing architectures, message-passing cluster computers, and hybrid architectures. These architectures exploit course-grain parallelism at the processing node level while at the same time riding the wave of microarchitectural advancements that exploit fine-grain parallelism at the instruction level. The individual processing nodes in both models consist of general purpose microprocessors, making it relatively easy to write and compile programs for these architectures. Field Programmable Gate Arrays (FPGAs), when used as the processing nodes in a parallel computing architecture, offer the benefits of application-specific processing while their ability to be reconfigured allows them to be used for a wide range of computations.

The proposed architecture is one where all the FPGAs of a reconfigurable supercomputer have themselves been networked independent of the control processors to which they are tied. This is a departure from the classical high performance reconfigurable computing (HPRC) methodology which has considered the FPGAs as a subordinate application accelerator (co-processor). In this new model, the role of the CPU and FPGA are inverted. CPUs are tasked with support operations such as the DMA of data to and from disk and the execution of codes which do not translate well into the systolic operations, while the FPGAs perform the majority of the computation. The CPUs are also required to provide control and coordination of the FPGA network.

The end product of this new architecture is a scalable fabric of hardware computers that are networked to stream data, providing partial (fine-grain processed) resultants to other downstream processing elements. Through this tunneled approach, much larger computations can be accommodated than currently fit in the limited resources of a single FPGA.

The proposed interconnection network employs the newly available multi-gigabit transceiver (MGT) blocks of current-generation FPGAs. Using these on-chip devices as the backbone of the network reduces the cost of implementing such a network and has the added benefit of allowing data to be sent directly into the FPGA for processing. This negates the need for an external ASIC router and I/O penalty resulting from the traversal of the slower parallel pads of the FPGA.

In conjunction with the hardware architecture, we describe facilities which must be implemented in the FPGA core. These facilities include the router logic, which is assigned to a partially programmable column of the FPGA, and a core logic interface, providing a standard communication interface. Covered as research topics, we address additional considerations which must be analyzed and solved for such a platform to perform at its fullest potential.

The remainder of this paper describes the architectural design and considerations of such a system. Section two provides a background on HPRC. Following this primer, section three details the open research topics for our architecture. After these introductory considerations, section four examines the computing architecture and section five describes a

14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06)
0-7695-2661-6/06 $20.00 © 2006 IEEE

candidate network architecture which satisfies the HPRC space. Section six then addresses algorithmic compilation and mapping techniques utilizing graphical analysis. Finalizing this paper, section seven dictates the research roadmap, providing a path to the resolution of the presented open research topics.

2. Background

Almost all modern HPCs have employed one variety or another of von Neumann CPU. These devices exhibit many different problems which have been patched over the years. Recent processor technologies such as Symmetric Multi Threading (SMT) and SuperScalar architectures have attempted to diminish the shortcomings of the underlying serial instruction stream. As can be seen with these new devices, the added complexity of patching the original architecture comes at the cost of price, power, and area. After accounting for the use of instruction and data caches, pipeline buffers, branch predictors, instruction reorder units, etc., an increasingly disproportionate amount of the modern CPU power and area budget is spent supporting these performance enhancing architectural patches without achieving a correspondingly proportional speedup. In defense of the current designs, microprocessors have enjoyed a degree of flexibility and ease of programmability and thus do not require changes to the way they are programmed. However, if we are to continue to push the envelope of processing throughput based on the exploitation of parallelism (the common characteristic of HPC applications), a back to basics approach will be needed to combat the growth of these parasitic trends.

The Field Programmable Gate Array (FPGA) is a device which accepts a binary stream (bitstream) that programs the state of its internal *gates* implemented as look up tables. In the FPGA domain, this binary stream can be considered a program. After the FPGA has been configured by the program, the resulting hardware is the physical implementation of the program, commonly referred to as a *hardware-task*. Entities such as Mitron and SRC have successfully advanced the field to a level of abstraction which allows the hardware to be programmed in a sequential semantic language much like C. The techniques of translating this C-like language to a concurrent semantic Hardware Description Language (HDL) (and then to a synthesized bitstream) comprise a significant sum of their work. Advantages of using such an intermediate language in opposition to the classical VHDL/Verilog design model are portability and the translation of legacy code written for general purpose processing environments. Both concerns are of the utmost importance to these companies' end-users.

In the case of currently implemented HPRCs, such as the Cray XD1, SRC 6, and the forthcoming SRC 7, the FPGA subsystem (the SRC MAP® is an example) is used to farm highly parallelizable and deterministic code segments to dedicated reconfigurable hardware computers. The motivation for this methodology is attributed to physical constraints including the relatively slow speed of FPGAs in comparison to CPUs, the bandwidth limitation of the interconnection bus of the FPGAs and the CPU control processor, and the inability for FPGAs to adequately handle nondeterministic workloads.

3. Open Research Topics

The reconfigurable processing element of current commercially available HPRC systems is used as a coprocessor to perform highly deterministic work upon a static selection of data. This does not lead to the required flexibility needed by general-purpose processing. First, the limitation of data injection into the reconfigurable element presents a time penalty, and second, the decoupled approach does not lead to efficient algorithmic decomposition. As such we need to address three key concerns that span distinct classical disciplines: distributed computing, compilers, and system-on-chip/network-on-chip.

Priority one is the creation of a fast and scalable interconnection network for the FPGA side processing fabric. Bandwidth is the greatest limitation observed in the reconfigurable computing space. Since FPGAs perform best when implementing a systolic operation, the need for one FPGA's partial result to feed into another will grow as problem sizes and complexities increase. In most current systems, data transfers from the host to the FPGA coprocessor are performed via a DMA from system memory to memory mapped on-board RAMs. Vendors such as SRC, as of the Carte 2.1 release, have attempted to improve this transfer by implementing libraries that allow for streaming to these memories with varying success [19]. However, a much more efficient technique must be developed that bypasses the memory hierarchy of the von Neumann backend.

Priority two is the need to adequately partition algorithms into parallelizable and non-parallelizable code segments as well as distribute the segments across multiple processing elements. This will allow a hybrid CPU/FPGA peer processing model to achieve greater speedups over the classical distributed processing model. It is currently the responsibility of the programmer to define this boundary based on a set of loose guidelines, often requiring repetitive testing and rewrites. Such a method is utilized in the SRC and Mitrionics approaches, where the programmer

writes the CPU-side application in C and the FPGA-side subroutines in either MAP-C or Mitrion-C. However, this decoupled approach will become cumbersome as our applications grow to span an increasing number of FPGAs and CPUs.

Priority three is the need to map a partitioned algorithm's segments across the distributed reconfigurable processing fabric to achieve large grain parallelism. Currently, most HPRC systems are intended to be used in a single user batch processed model. However, HPRC systems of the future will use more reconfigurable processing elements, such as the Cray XD1 and SRC crossbar extension for multiple MAPs, necessitating the ability to run multiple applications concurrently as a means of achieving greater utilization. To execute these multiple processes without contention, a mapping strategy and supporting routing policy must be developed that reduces the blocking behavior of the interconnection network both for single and multiple process scenarios.

Only with these problems solved can HPRCs be perceived as a viable alternative to CPU based solutions for an equally wide sampling of HPC applications.

3.1. Processing Element Network

We propose a new generalized system architecture for HPRC systems where multiple FPGAs, acting as processing nodes, are interconnected in a direct network. Interestingly, much of the research in the System-on-Chip and Network-on-Chip fields is directly analogous to developing a feasible HPRC architecture. However, translating these techniques to the needs of HPC requires the addition of some basic networking capabilities. Namely, we must provide a standardized processing element interface for all reconfigurable processing devices allowing the abstraction of the network from the processing elements.

The stringent requirement of network abstraction can be split into two main considerations. The first is the need to allow the underlying network structure of the global architecture to change. Numerous networking parameters such as routing policies, router design and features, and network topologies should remain alterable, while not invalidating the processing element interface. If this abstraction level is not provided, the architecture will be tied to a specific network structure which will hinder the flexibility of the overall architecture as it evolves. The second consideration centers on processing element flexibility. Only with a middle layer, such as the interface, can a single processing element be instantiated any number of times, at any location in the network. With this added flexibility, larger designs will become easier to realize.

3.2. FPGA Reconfiguration

There are currently two different forms of FPGA programming. One involves the reprogramming of the entire FPGA where normal reprogramming periods are on the order of seconds. This produces a significant penalty for systems which change *jobs* (the hardware-tasks being executed) at a relatively high frequency. The other form of programming involves the partial reprogramming of a segment (column) of the unit. Current FPGA densities have not yet warranted the use of partial reprogramming in many real world applications. However, the technology does currently exist and will be relied on by the on-FPGA router required for our interconnection network.

Marescaux et al. [1] have begun an investigation and early prototyping of a partially reconfigurable FPGA device which utilizes a wormhole routed network [11] to interconnect the tiles of the FPGA. By providing such a fabric, the development team succeeded in dynamically changing the operating mode of the FPGA, enabling a low-power highly efficient portable multimedia device capable of handling interactive workloads. Their work demonstrates that such a device is conceivable using current technology. However, due to the constraints of the Xilinx Virtex II FPGA, the chosen two dimensional mesh network was collapsed into a single row. Other shortcomings of the design include the inability to clock the interconnection network at speeds greater than the reconfigurable fabric. This results in the creation of an artificial bottleneck. If these techniques are to be further developed, an alternative structure must be realized.

Our method of addressing these problems is to translate the techniques employed by Marescaux et al. to a larger grain interpretation of a virtual hardware component. Rather than attempting to place an entire network internal to the FPGA (as is done in ASICs with a NoC) we view each FPGA's user logic (omitting the router column) as the atomic unit of reconfiguration. Using many FPGAs interconnected by an MGT network, we are able to increase the speed of the network by using their integrated high-speed transceivers. We can more easily reconfigure the atomic devices since only the router must be pushed into a partially reconfigurable column, and the need to compress or collapse the network structure is not required since the network topology and structure is created external to the FPGAs by board-level traces.

Additionally, since jobs execute for extended periods of time in the HPC domain, we can amortize the reprogramming costs. However, if we wish to

have a flexible processing environment, the ability to process nondeterministic workloads is required. In this sense, the term nondeterministic workload means the ability to handle, variable sized data, branch conditions, exceptions, and data-dependent loop structures. Classically, these problems have been avoided by the various FPGA HPRC vendors since it is a complex problem outside the scope of their work.

3.3. Algorithmic Partitioning

At this time, FPGAs are generally limited to a maximum logic speed of 250 MHz [18] whereas an Intel Pentium 4 is capable of operating at a pipeline frequency of 3.8GHz. Although both devices utilize a 90nm fabrication process, FPGAs suffer from longer wiring traces and increased area required for its reconfigurable elements [8]. This overhead results in lower logic densities and precludes the FPGA from rivaling the clock speeds of a full-custom ASIC of equivalent data path depth.

To address this hindrance, FPGAs must be *programmed* to employ sufficiently wide data paths such that the resulting parallelism produces greater throughput verses the sequential computation of the faster microprocessor. Currently, this problem is solved by constructing HDL which exhibits the configurable hardware as a concurrent device or using other high-level languages and their corresponding compilers. However, neither HDL, nor any of the currently available HLL compilers, allow their languages to be implicitly compiled across multiple hardware units. The determination of which segments of a large algorithm should be partitioned into atomically reconfigurable units, as well as their placement in the interconnected network is a problem not currently addressed.

This introduces a new concern stemming from the multi-level parallelism that such a partitioning and mapping strategy would produce. Regardless of whether the partitioning is performed by hand or an automated process, the placement of the resulting bitstreams within the reconfigurable network will need to be optimized. If a naïve mapping is performed, the resulting contention for network resources will degrade the throughput potential of the computing device.

4. Computing Architecture

Our proposed architecture employs a serial routed direct network of interconnected FPGAs as shown in Figure 1. In this system, messages containing intermediate results, address location, and function unit instructions may be set from any source node to any destination node. This network forms a "*collaboration network*," similar to that used in the TRIPS multi-core architecture [10].

Each FPGA is interconnected using the MGTs of the current generation FPGAs. Both Xilinx and Altera have released implementations of their flagship FPGAs (Virtex II Pro/X, Virtex 4, Stratix GX, and Stratix II GX) with this functionality [15][16][17][18]. Note that the routers are drawn internal to the FPGAs. The router logic is encapsulated into the FPGA as a partially programmable column. This router logic allows data to be routed across the many available channels of the FPGA (the Stratix II GX includes twenty 6.375Gbps bi-directional channels).

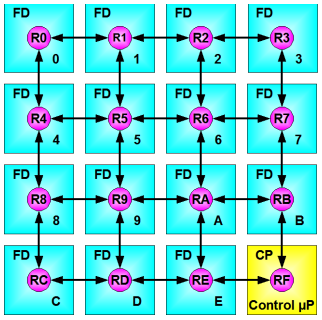


Figure 1: FPGA Array With Control Processor

4.1. Definitions & Terminology

Before an adequate discussion of the Processing Element Interface (*PEI*) can be made, a general outline of the greater computing system as shown in Figure 2 must be described as well as the names assigned to each physical and logical element in the computing fabric.

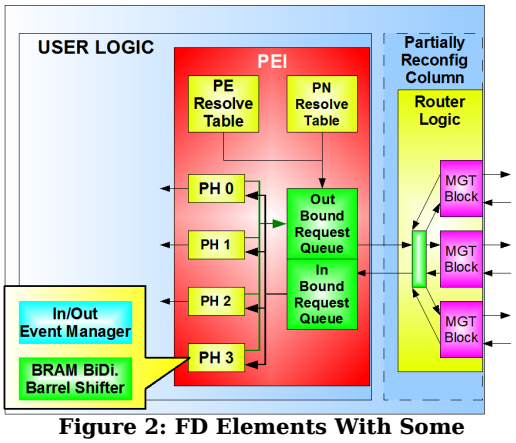


Figure 2: FD Elements With Some Omissions

A physical FPGA device in the fabric will be called an *FD* (FPGA Device), the bitstream used to *program* an FD shall be referred to as a *PE* (Processing Element), and the routers shall be called an *R* (Router). For the purposes of our discussion the FDs, PEs, and Rs shall be appended with a numerical value to indicate which physical/logical device is

being referenced.

Algorithms in the realm of this device are implemented via a control and mapping description (*CMD*) and a series of one or more PEs. The CMD is used by the control processor (*CP*) to determine how many of each type of PE must be loaded and in what order. This allows the CP to dynamically *program* available FDs for use as a specified PE (look-ahead can be utilized to ease the time penalty of programming).

Since data must traverse the network, additional PE housed logic must be instantiated to support the PEI. The atomic element required by the interface is the Port Handler (*PH*). A port handler facilitates the composition, injection, and ejection of packets to and from the router on its designated port. This is an event-based (asynchronous) device which ensures that blocking behavior is arbitrated.

4.2. Processing Element Interface (PEI)

To enable bitstream reuse, it is imperative that a standardized network interface exist across all bitstreams which is amenable to the architecture. We have chosen a ports system, where a given PE has a series of Port Numbers (*PNs*) which reference local Port Handlers within the PE. To assist our discussion we will first examine the packet structure of the network. Following this examination we will discuss the two primitive operations, injection and ejection of packets via the PEI.

4.2.1. Packet Structure. The packet structure is shown in Figure 3. Each packet in the underlying *collaboration network* is composed of a routing header and a payload.

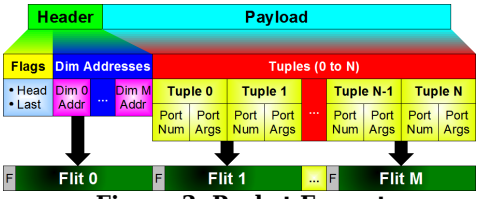


Figure 3: Packet Format

The payload is composed of a variable number of tuples which contain a port number field and an argument field. The port number is a numerical designator which references an internal PH. The argument of the tuple is the data which is to be copied to the internal array which the PH services. The arguments are of variable word count which is set at synthesis time. However, each argument that traverses the network must be the same word count as the BRAM array it is to be stored in.

4.2.2. Packet Ejection. When a packet is received from the *collaboration network*, the arguments are forwarded to the PH as designated by the PN of the tuple. The PH then buffers the argument in the local BRAM referenced by the port number, shown in Figure 4.

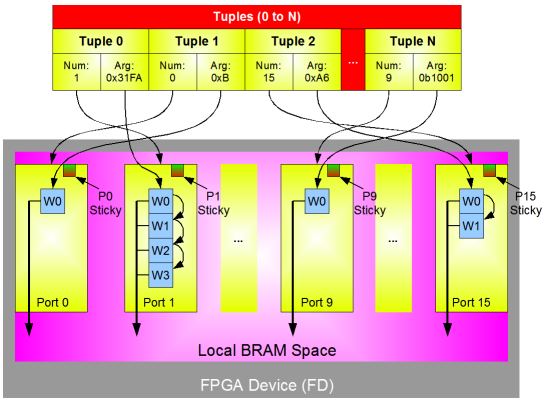


Figure 4: Tuple to Port Handler Ejection Interface

This structure introduces overhead in the form of BRAM utilization, reducing the available local memory for the computations provided by a PE. However, the PE may not be capable of processing all data as it arrives, resulting from physical computation restrictions and data dependencies. By buffering the arguments that a given PE requires, issues stemming from out-of-order argument arrival and blocking behaviors, such as multi-party sources, can be circumvented.

4.2.3. Packet Injection. Packet injection is accomplished by a series of PH's being scheduled for out-bound sends in conjunction with the addition of a header formulated by the Packet Constructor.

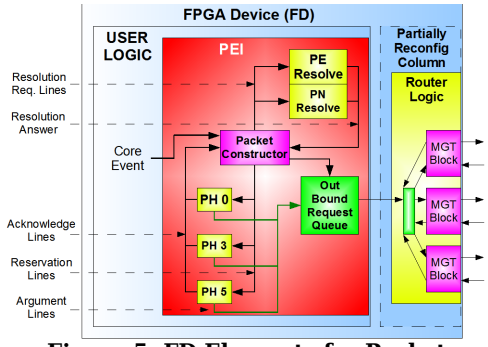


Figure 5: FD Elements for Packet Injection

A thorough functional description of each element shown in Figure 5 is necessary to understand the operation of the PEI when constructing and injecting a new packet into the collaboration network.

When a core event occurs (computation complete etc.) that requires a packet transmission, the event is passed to one of the packet constructor blocks of the PEI (multiple packet constructors may exist). Not

shown is the existence of a FIFO internal to the Packet Constructor to service multiple events by order of arrival or priority.

The number of packet constructors that may be placed in a PEI is a determined by the number of possible packet transmissions and the port handlers which comprise the arguments of the payload. It is easiest to think of a constructed packet as a vector of destinations (address) and a vector of packet handlers (payload). If two packets' PH vectors are disjoint, meaning that they share no port handlers, then they may belong to separate packet constructors, since the reservation of the packet handlers will not result in a deadlock. However, should more than one packet share port handlers, these dependent packets must belong to the same packet constructor to avoid PH reservation conflicts.

The packet constructor is partitioned into two stages: setup and execution. The setup stage is responsible for resource allocation and resolution; the execution stage places these resolved addresses and port number-argument tuples onto the injection bus and deallocates the resources required for the packet after they have been used.

When a core event occurs, the packet constructor is activated and the setup stage begins. Internal to each packet constructor is a lookup table which instructs the packet constructor as to which non-resolved processing element(s) and port number(s) this packet (spawned by the core event) is to be delivered. This information is set at synthesis time. Since the synthesized information may need to be updated based on runtime criteria (adaptation), these non-resolved PE and PN entries are then sent to the PE and PN resolution tables as indices. The results of the lookups are the physical network addresses and port numbers. This flexibility allows the dynamic placement of PEs within the FD fabric without requiring the re-synthesis of all PEs used in a given application.

In parallel to the resolution lookup, the packet constructor also lodges a reservation request to all the PHs that are part of the subject packet's PH vector. The PHs may have been reserved previously by an ejected packet or a core logic component. Only when all PHs have been released and send the ready acknowledge signal back to the packet constructor will the packet constructor lodge a request in the out-bound request queue. This queue is a FIFO, which means that some additional latency may be observed as other packet constructors (already in line) are serviced. The out-bound request queue reservation signals the end of the setup stage.

Once the packet constructor under examination reaches the head of the line, the packet constructor is sent the go signal beginning the execution stage. Afterward, the packet constructor places the resolved PE lookups onto the injection bus. Following the exhaustion of the PE lookups, it then inserts a resolved PN from the PN vector and then instructs the corresponding PH to barrel shift its data onto the injection bus. This process is repeated until all PN and PH tuples (the payload) have been exhausted. After each PH barrel shifter rolls its data onto the bus, it resets its reservation and is considered reservable by another packet constructor event, core logic component, or ejection operation event. After the last PH has unloaded its data, the packet constructor may then service the next core logic event in its queue.

As long as the inputs and outputs of the PE are serviced by different port handlers, the PE can stream data in and out simultaneously on the multi-dimensional MGT network.

5. Network Architecture

The collaboration network is composed of an MGT-interconnected topology of FPGAs. The choice of topology and routing policy is an open ended matter. Preexisting topologies and policies have emerged over the past years in different machine architectures. The examination of Dally's wormhole routing and the later extended virtual cut-through routing was of great interest in the late 80's to early 90's for distributed multicomputer architectures. The simplicity of wormhole routing and the multitude of routing policies that have been developed following its introduction have also made it a good candidate for current NoC research [4, 5, 6, 16, 18]. Building on these previous networks, aspects of the proposed interconnection topology and routing policy will be discussed and elaborated.

5.1. Topology Selection

At this time, all HPRC vendors have chosen a crossbar switch solution to network multiple *accelerators*. In the case of the Cray XD1, each accelerator is directly connected to the host processors of each blade and the other blades of the cluster via their proprietary *RapidArray Interconnect* (a product of OctigaBay). Under the hood, the *RapidArray Interconnect* is a very large crossbar switch. Similar to the Cray efforts, SRC has employed a similar approach, the *Hi-Bar* switch, to network multiple *MAPs*.

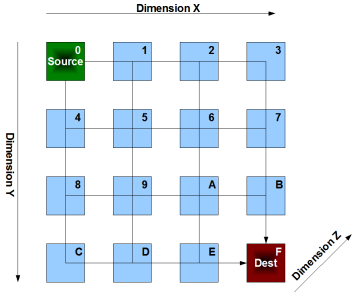
However, the crossbar will eventually meet its end, since its ability to scale with the number of networked elements is $O(n^2)$. A much more desirable growth pattern would be one that scales linearly, O(n), with the number of networked elements while preserving the ability to route data to all destinations from any given source. Many such topologies exist

including the mesh, torus, and alike. Other more exotic topologies such as the k-ary n-cube, butterfly, and fat-tree also exhibit admirable traits, but do so at the cost of added complexity.

It is the opinion of the authors that a simple 2D bi-directional mesh or torus network topology will result in the best possible throughput. This should not be interpreted as meaning that one topology is better than another for randomized traffic. Rather, the statement is intended to mean that if adequate precautions and pre-calculations are made, this simple topology will render near if not exactly identical performance results while avoiding the need for greater hardware complexity. Of course, a proof of this statement will be required and is part of the research roadmap as stated at the conclusion of this paper.

5.2. Routing Policy Selection

There is a large number of routing policies (algorithms) that exist for the virtual cut-through routers that we are targeting for use in our system architecture [3][7][12][13]. All these routing policies can be classified into one of three sets: deterministic, fully-adaptive, and semi-adaptive.



$$Shortest\ Path\ Count = \frac{((d_x + d_y)!)}{(d_x! * d_y!)}$$

Figure 6: Possible Paths on a (2D Bi-Directional Mesh)

An example of a deterministic routing policy is the dimension-ordered shortest path algorithm. In this policy, a topology is viewed as a *m*-dimensional space. In truth it is an m+1 dimensional space, but the last dimension (injection to and ejection from PEs) is implicit. Figure 6 shows the 20 possible minimum paths that could be taken from the designated source to the destination.

However, if using the deterministic dimension ordered policy, all of the lowest order dimension will be traversed, followed by the next lowest order dimension (LOD) until completion. This results in only one possible path from 0 to F by vectors:

< 0 1 2 3 7 B F > in the case that X is the LOD,
< 0 4 8 C D E F > in the case that Y is the LOD.

One advantage to such a routing policy is that if we examine the possible flight vector, only *m*-1 turns must be made where *m* is the number of dimensions (omitting the implied dimension Z). One disadvantage to such a policy is that if link 3-7 is busy due to a current transmission, the policy cannot route around the contention (this is called a block). This means that the available bandwidth of the network is not being used to its full potential.

To address the bandwidth utilization problem, a number of fully-adaptive policies have been invented over the years that attempt to route around blocks by taking any minimum routable path from the source to the destination. One disadvantage to such policies is their inability to guarantee freedom from deadlock without a sufficient number of virtual channels (requiring additional buffers, arbitration logic, larger internal crossbars, and allocation logic). However, a discussion of such policies is beyond the scope of this paper.

The last classification of routing policies is semi-adaptive. Like the fully adaptive algorithms, this policy class allows packets to traverse multiple paths from source to destination. Unlike the fully-adaptive algorithms, the semi-adaptive policies enforce a rule set (turn restrictions) to ensure that the network is deadlock-free. The advantage to this class of policies is their ability to provide comparable performance without the need for virtual channels maintaining the router's simplicity and efficiency. One such algorithm is the Odd-Even routing policy [13]. The O-E policy, at this point in our research, appears to hold the most promise. However, even at this early phase, we have made observations that could greatly improve the effectiveness of the policy such as the implementation of a turn-predictor cache, pipelining, and other various improvements.

What must be understood about these policies and the networks they were originally intended to drive is that there is a distinct difference in the pseudo-random traffic they were analyzed with and the NoC/FPGA fabric traffic that we wish to optimize. This fundamental difference is the existence of predicable traffic patterns.

5.3. Semi-Deterministic Traffic

Semi-deterministic traffic is traffic whose patterns exhibit some degree of predictability. Often in the past, routing policies were analyzed based on randomized traffic since it offered a good indication of the network's capabilities in the general case. Unlike this scenario, after a given PE has been synthesized, we have complete knowledge of all transmissions which it *may* send to any other PE.

However, real algorithms are not completely

static. Many operations in real code are based on branch conditions and loop invariants that are data dependent. As such, unless a characteristic data set is provided to a synthesized PE, knowledge of the frequency and periodicity of these dynamic aspects will not be known. What should be taken away from this realization is that, although we might not know the periodicity or frequency of transmissions that may be housed in a static or dynamic segment of the design, we do know of their existence, the source and destination pairs, and the volume. Based on this *a priori* knowledge a globally optimal placement of PEs within the network can be determined. PEs that communicate most frequently with the largest volume should be placed closer together, and PEs that never communicate with each other should be placed in disparate regions of the reconfigurable fabric to avoid blocking transmissions.

6. Algorithmic Compilation and Mapping

When describing all the pre-calculations which must be performed by the control processor, it is important to analyze the three main portions of the architecture that require it, the CMD, the relative addresses of PEs and PNs, and the placement of PEs within the reconfigurable fabric.

6.1. The Control and Mapping Description

The CMD describes all of the PE instantiations that are required for a given algorithm, as well as the interrelation of these PEs. If we look at an algorithm as a unified data & control flow graph of primitive processes, we can determine all of the data that must be supplied to a given region of the graph as well as all the resultants that are passed out of a region.

That said, a PE is a parallelized region (subgraph) of the algorithm graph. We can replace all of the vertices (primitive processes) of the subgraph with a single vertex and connect all of the incoming and outgoing directed edges from the subgraph to this subsuming vertex. If the algorithm graph has been decomposed into a series of PE vertices, each incoming edge to a given vertex is treated as an incoming port and all outgoing edges are treated as an outgoing port. Incoming edges transfer data that is ejected from the collaboration network (edges) into a PE, and outgoing edges transfer data that is injected into the network on the part of the PE.

A given vertex need only be knowledgeable of its downstream neighbors. If we look at an example graph, as shown in Figure 7a, we see that the Graph G has been partitioned into four distinct regions (subgraphs) in Figure 7b. Each subgraph is then collapsed into a single vertex as shown in Figure 7c.

For the architecture to function correctly, PE_0 need only know that PE_3 exists and that it is to forward some of its resultants to that Processing Element. To facilitate this local knowledge reflect on section 4.2.3's reliance on a PE and PN lookup table. When the PEs are loaded onto the FD fabric, it is the responsibility of the CP to adapt these tables to the physical mapping it performed. The only way that the CP could know to do this is through the use of the CMD, and its knowledge of the topology and routing policy of the FD fabric.

The CMD begins with a prologue that defines all of the PEs (and the ports of these PEs) used in its execution. Multiple PEs of the same type may be instantiated, so this is not a physical definition of all the PEs that will be loaded onto the FD fabric. After the prologue, a table describing all of the PE instances, and their downstream connections is defined.

For example, in Figure 7c, PE_1 has four downstream connections, two of its output ports are mapped to PE_0, one is mapped to PE_2, and another is not designated by the diagram. When the CP places this PE onto the FD fabric, it will be required to configure PE_1's PE & PN lookup table to point to the network and port address of all its downstream connections.

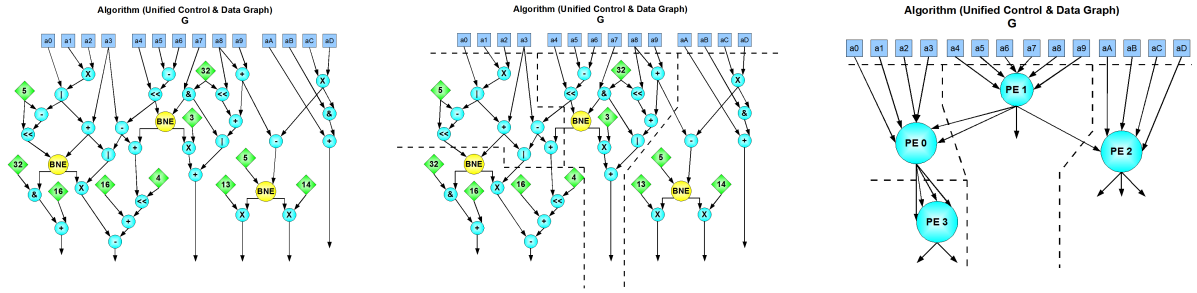


Figure 7: Graphical Algorithm Partitioning
a : Primitive Control & Data Dependency Graph
b: Partitioned Primitive Graph
c : Collapsed Subgraphs

6.2. Relative Addressing Example

The CP must use the CMD to determine the placement of PE instances on the FD fabric and update each PE's lookup tables to reflect that placement. An example will help make this clear. Using Figure 8, assume that the *CP* has made the following assignments:

PE_0 → FD_(0,0)
PE_1 → FD_(0,2)
PE_2 → FD_(3,3)
PE_3 → FD_(2,0)

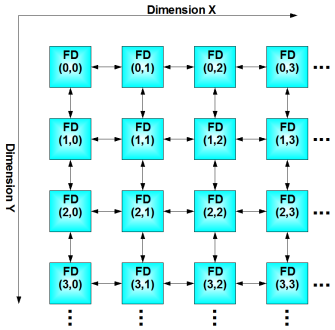


Figure 8: FD Fabric (2D Bi-Directional Mesh)

Since addressing in Wormhole and Virtual Cut-through networks is achieved by relative coordinate based offsets, each PE would need to have the its tables updated with the resolved relative address of its downstream neighbors. PE_0 would need the address of PE_3 which, relative to PE_0's placement, is (2,0). PE_1 would need the address of PE_0 and PE_2 which are (0,-2) and (3,1) respectively. PE_2 and PE_3's downstream neighbors are not defined in the previous example.

If at a different time, the mapping is not the same (perhaps for reasons of contention or a different topology), the ability to update, the PE Resolution Tables, allows the same CMD to be run with a different mapping. It is this ability to adapt the PE to runtime characteristics that enables modular reuse of pre-compiled PEs.

6.3. PE Mapping

In the previous section we assumed that the CP was capable of performing a mapping based on its knowledge of the FD fabric's topology and the CMD. However, this mapping is of extreme importance, and can make or break the architecture.

Recall that Wormhole and Virtual Cut-Through routing are blocking routing strategies, if a naïve placement of PEs is made, the blocking characteristics of the network may greatly reduce the throughput potential of a given mapping. The placement problem is NP-Hard (an instance of a constrained *quadratic assignment problem*) which means that heuristics must be used to perform the mapping determination in a reasonable amount of time, since it must be performed each and every time the CMD is executed [2]. To reduce this penalty, it is advantageous to append previously mapped layouts to a CMD to provide clues to subsequent executions.

7. Conclusions and Future Work

This paper describes a novel system architecture and application development framework for generalized reconfigurable distributed computing. The benefits of this architecture include its ability to exploit fine- and course- grain parallelism through its use of a data-flow processing model, linear interconnect scalability through its use of a direct network, and its ability to reuse pre-compiled processing elements through its use of pre-computed PE/PN lookup tables. This paper also outlines several open research problems that must be resolved in order for this architecture to be capable of executing traditional HPC applications.

To satisfy our hardware needs we are using a sufficiently similar unreleased hardware architecture to prototype our distributed reconfigurable system architecture. However, many of the research areas cannot be resolved by the mere existence of a physical system. Adequate simulation facilities must also be created to determine the feasibility of the PEI, CMD, Topology, Router Architecture, and Routing Policy. After conducting an evaluation of currently available network simulators, it has become evident that a more powerful cycle accurate simulator will be required. To address this need, we have started work on NoCsim available at http://sourceforge.net/projects/nocsim.

As an introductory use of this simulator, we are performing a wide survey of routing policies under semi-deterministic traffic patters. To automate the generation of these traffic patterns we have written a middle layer that allows the translation of task graphs from TGFF [9] into an XML traffic description. Using this traffic description we are able to experiment with our mapping heuristics. After formalizing and validating these mapping heuristics and policies under varying characteristic traffic patterns we will begin analyzing real applications.

The next major research effort will be the development of algorithm partitioning techniques. Real task graphs exhibit data-dependent branching and looping behavior. Both dynamic and static loops in a graph are strongly connected subgraphs. Search algorithms currently exist that are capable of determining these strongly connected regions. In an effort to reduce communication costs, one employed technique is a rule enforcing that strongly connected subgraphs must be collapsed into a PE vertex.

The next step is filling in the corresponding region around these strongly connected segments. Using the Kernighan-Lin min-cut algorithm [14], we can reduce the amount of communication required for each PE. We are providing an easement of primitive vertices around the designated strongly connected regions. The size of the easement is determined by an approximation of how much of the graph can be synthesized into one of the FDs. After creating the easements, we will allow the K-L heuristic to determine where we should draw our partitions. Unlike the application of this algorithm for circuit board design, we apply a weight to each edge in the graph that indicates its traffic density. This will allow us to cut more edges (create more ports) if the cost reduces the overall transmission density.

Conveniently, the SCALE research compiler [10] is capable of translating native C code into a unified control and data flow graph representation. The final directive of our research is the integration of our partitioning and PE mapping heuristics into this compiler. The goal is to create a compiler capable of accepting algorithmic descriptions in the form of C, graphically explode the code, make determinations of which segments should be housed in the FD fabric and CPU space, and create the CMD with synthesized PEs and CPU side code.

By taking these early steps we are already traveling down this path. Even without the full compiler, the mapping heuristics will be useful when placing hand-crafted logic cores onto the FD fabric of the machine. In addition, the evaluation of different topologies and router architectures will allow for greater flexibility and performance improvements, even in the general case.

8. References

[1] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins, "Interconnection networks enable fine-grain dynamic multi-tasking on fpgas," in *FPL '02: Proceedings of the Reconfigurable Computing is Going Mainstream, 12th International Conference on Field Programmable Logic and Applications*. London, UK: Springer-Verlag, pp.795-805, 2002.

[2] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based NoC architectures under performance constraints," in *Proceedings of ASP-DAC*, Jan. 2003.

[3] R. Cypher and L. Gravano, "Storage-efficient, deadlock-free packet routing algorithms for torus netoworks," *IEEE Trans. Comput..*, vol. 43, no. 12, pp. 1376-1385, 1994.

[4] B. Sethuraman, P. Bhattacharya, J. Kahn, and R. Vemuri, "Lipar: A light-eight parallel router for fpga-based networks-on-chip," in *GLSVSLI '05: Proceedings of the 15th ACM Great Lakes symposium on VLSI*. New York, NY, USA: ACM Press, pp. 452-457, 2005.

[5] N. Kavaldjiev and G. J. M. Smit, "An energy-efficient network-on-chip for a heterogeneous tiled reconfigurable systems-on-chip," in *DSD '04: Proceedings of the Digital System Design, EUROMICRO Systems on (DSD '04)*. Washington, DC, USA: IEEE Computer Society, pp. 492-498, 2004.

[6] N. Kavaldjiev and G. J. M. Smit, "A survey of efficient on -chip communications for SoC," in *PROGRESS 2003 Embedded Systems Symposium*, 2003.

[7] L. Schweibert and D. N. Jayasimha, "Optimal Fully Adaptive Wormhole Routing for Meshes," *In Supercomputing '93*, pp. 782-791, 1993.

[8] J. H. Anderson and F. N. Najm, "Power Estimation Techniques for FPGAs," *IEEE Trans. on VLSI Syst.*, vol. 12, no. 10, pp. 1015-1027, Oct. 2004.

[9] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Proc. Int. Workshop Hardware/Software Codesign*, pp. 97-101, Mar. 1998.

[10] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore, "Exploiting ILP, TLP and DLP with the Polymorphous TRIPS Architecture**"**. *In Proceedings of the 30th Int. Symp. on Computer Architecture*, pp. 422-433, Jun. 2003.

[11] W.J. Dally and C.L. Seitz, "The torus routing chip," in *Journal of Dist. Computing*, vol. 1, no. 3, pp. 187-196, Oct. 1986.

[12] H. Matsutani, M. Koibuchi, Y. Yamada, A. Jouraku, and H. Amano, "Non-Minimal Routing Strategy for Application-Specific Networks-on-Chips," in *Parallel Processing, 2005. ICPP 2005 Workshops,* pp. 273-280, Jun. 2005.

[13] G. M. Chiu, "The Odd-Even Turn Model for Adaptive Routing," in *IEEE Tran. Parallel Distrib. Syst..*, vol. 11, no. 7, pp. 729-73, 2000.

[14] B. W. Kernighan and S. Lin. "An efficient heuristic procedure for partitioning graphs," The Bell System Technical Journal, 49(2):291--307, 1970.

[15] http://www.xilinx.com/bvdocs/userguides/ug070.pdf, Virtex 4 User Guide v1.4 (UG070), Xilinx Inc., Sept. 2005.

[16] http://direct.xilinx.com/bvdocs/userguides/ug012.pdf, *Virtex II Pro and Virtex II Pro X FPGA User Guide v4.0 (UG012)*, Xilinx Inc., Mar. 2005.

[17] http://www.altera.com/literature/hb/sgx/sgx_handbook.pdf, *Stratix GX Device Handbook sgx5v1-1.1*, Altera Inc., 2005.

[18] http://www.altera.com/literature/hb/stx2gx/stxiigx_handbook.pdf, *Stratix II GX Device Handbook Preliminary Information siigx5v1-1.1*, Altera Inc., 2005.

[19] *SRC C Programming Environment v2.1 Guide (SRC-007-16)*, SRC Computers Inc., Aug. 2005.