

ADAPTIVELY RECONFIGURABLE CLUSTERS (ARC)

PROJECT DESCRIPTION

1. Introduction

Reconfigurable systems based on field programmable gate arrays (FPGAs) have been receiving rising attention. FPGA reconfigurable chips have been described by Scientific American as “the chips that can rewire themselves”. The typical chip has a number of hardware configurable logical blocks (CLBs). The function of each of these blocks as well as the way they are interconnected can be altered. This can be done, for example, using the VHDL hardware description language.

Adaptive computing architectures based FPGAs chips and general-purpose microprocessors have many synergistic properties, see Table 1 [GHA03b]. First, the modular structure of FPGA chips matches many fine grain applications such as signal and image processing operations. Secondly, their in-circuit programmability can provide hardware processing in a data-flow style for critical functions, while maintaining flexibility and allowing hardware reuse. Thirdly, the presence of general-purpose microprocessors can be leveraged for types of operations that are not easy to implement in FPGAs or do not lend themselves to this type of processing, such as floating point operations. In addition, having nodes that combine both reconfigurable and conventional processors allow reconfigurable architectures to scale by leveraging the advances in high-performance computing.

Table 1: Synergism between Reconfigurable and Conventional Processors

	Micorprocessors	Reconfigurable Processors
Parallelism Type	Coarse-Grain	Fine-Grain
Processing Flow	Control-Flow	Data-Flow
Scaling Applications Across Processors	Parallel Programming Tools Available	Hard, through hardware only

Microprocessors have recently achieved tremendous performance gains with clock ratings now in the Gigahertz. Yet, reconfigurable processing chips have demonstrated orders of magnitude of speed over microprocessors in many applications of importance to national security such as image processing and cryptography [FID02][GHA02][ARA03][NGU03][MIC03], and their uses have been expanding to many other critical areas such as bio-informatics [STE03]. This is because of the fact that they support many fine-grain styles of computing such as data-flow, systolic arrays, and pipelining. In addition, the performance of reconfigurable computing devices is improving at a much higher rate than that of the Moore’s law, when compared to microprocessors. Considering as a performance metric, (# of gates per chip) X (clock rate), figure 1 demonstrates that relative progress in performance [SRC02]. One reason for this is the fact that reconfigurable devices were introduced much later than microprocessors, therefore, they are still clocked at a much lower rate, hundreds of Megahertz, and have much more room to improve.

Reconfigurable computers based on FPGAs, however, are hard to program and application scientists must know hardware design and hardware description languages to use them. Furthermore, they suffer portability and scalability problems, as applications developed for one board cannot run on another and are difficult to extend to a system of more than one board.

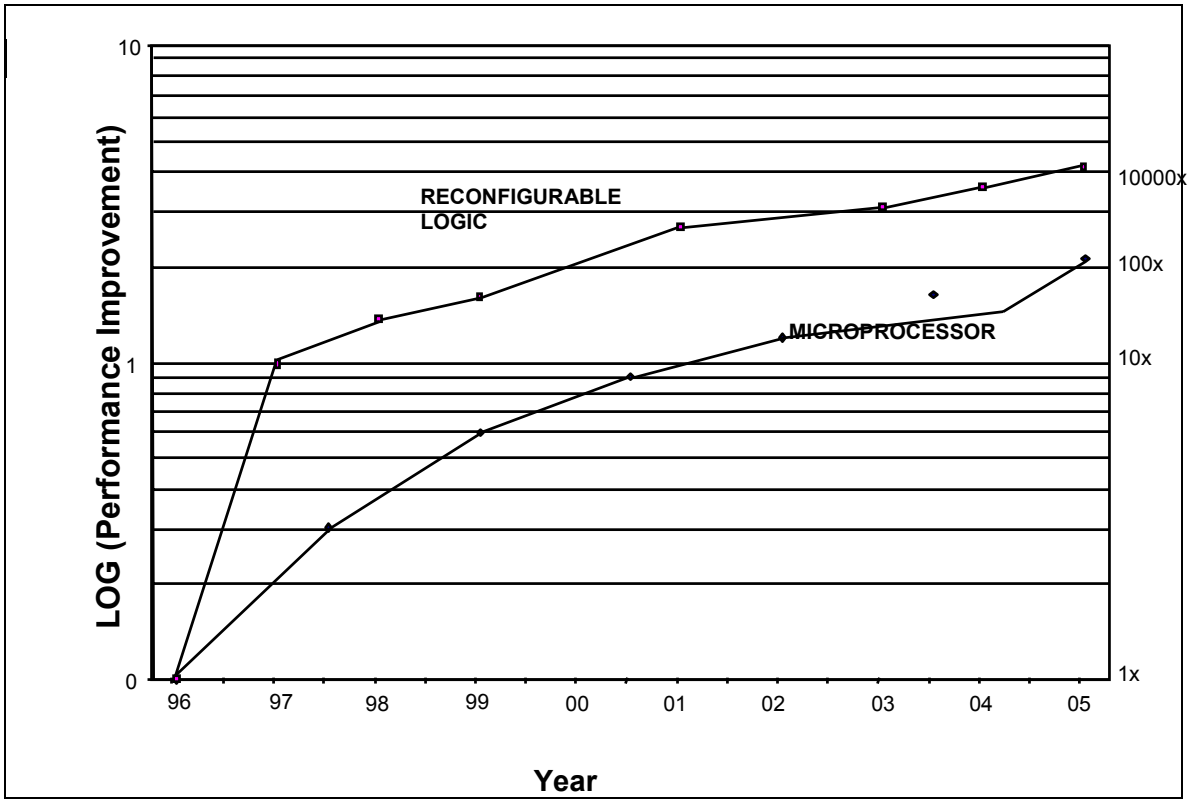


Figure 1. Relative Improvements of Reconfigurable Logic Devices and Microprocessors

This proposal addresses the need for a new architectural model that leverages the progress made in general-purpose high-performance and cluster computing and integrates them with recent advances in reconfigurable computing, in such a way that unlocks the synergism between these two paradigms. There have been already many clusters that have both conventional and reconfigurable computing resources assembled by researchers. However, we believe that there remains a real need for efficient execution and programming models for such clusters. Therefore, we propose our ARC model which embodies many new as well as extended concepts to provide: 1) ease of programming, and 2) adaptive and efficient run-time reconfiguragion and 3) a well understood execution model for reconfigurable clusters. From the architectural point of view, ARC uses a middleware referred to as the run-time reconfiguration manager (RTRM) to integrate these concepts into clusters of reconfigurable nodes, where each node is a conventional workstation equipped with reconfigurable computing boards as co-processors and the nodes are integrated via a dedicated interconnection network.

For ease of programming, the middleware will hide all hardware programming details and present the FPGA capabilities to the programmer as domain specific libraries that can be called from C or C++. Applications can be then scaled beyond the node via standard parallel programming paradigms such as the message passing interface (MPI). This will also ease the portability problem associated with reconfigurable systems as it decouples the application-specific high-level programming modules from the device-oriented programming. For adaptive reconfiguration, ARC utilizes a cache-memory-management-like model to manage reconfigurable computing

resources and allows, based on a cost function, an application to run on either the reconfigurable computing hardware or using the conventional microprocessors. This permits an application to dynamically adapt to run-time changes in processing requirements. ARC is based on a number of innovative concepts that have not been previously explored for reconfigurable architectures. These include processing locality and selective dual-track execution, as well as concepts extended to reconfigurable computing from advanced processor architectures such as scoreboarding, forwarding and chaining in reconfigurable hardware. For example the concept of processing locality has not been defined or considered before in this context. As in cache memory, processing locality characteristics will be used to discover at run time the workload processing needs and reconfigure the hardware accordingly providing a cache-memory-like resource management scheme for adaptive computing clusters. These principles have the potential for hiding reconfiguration latency, reducing its effective time orders of magnitudes by using dynamic processing patterns to anticipate and match application needs.

ARC defines the concepts for managing the execution in a modular architecture designed as a cluster of hybrid nodes, each of which consisting of a conventional microprocessor and a reconfigurable compute engine. The upper level of this architecture is a scalable fabric of general-purpose processors interconnected by a COTS high-speed network, such as Myrinet™. This level exploits coarse-grain parallelism and provides coarse-grain scalability with the ability to easily add nodes and integrate reconfigurable boards of any type. The lower level is a pool of reconfigurable FPGAs, providing fine-grain parallelism and high-compute density. The resource management policies defined by ARC can be easily implemented via a middleware, referred to here as the run-time reconfiguration manager (RTRM). ARC conceives a new framework for the relationship between applications, system-oriented high-performance computing clusters, and device-oriented reconfigurable computing, which leverages COTS conventional and reconfigurable hardware and software. If successful this research will provide the concepts and methodology for developing easy to program reconfigurable high-performance computing systems at a low cost and with a simple path to upgrading. Through the RTRM middleware, ARC hides the hardware reconfiguration details, reducing, the use of the reconfigurable hardware to calling a high-level language function component library. Thus, existing high-level paradigms including object-oriented languages (e.g. C++) augmented with message passing libraries (e.g. MPI) can be used. In addition, this function component library can be upgraded independent of existing applications as new reconfigurable computing boards are introduced.

The proposed research will develop the ARC model concept and determine its viability as the foundation for efficient, dynamically adaptive, easy-to-use, embeddable, and scalable systems. This proposal seeks to develop detailed concepts for the ARC architectural model, develop and study essential execution mechanisms, and establish a testbed to experimentally evaluate the potential of this approach.

2. Motivations and Objectives

2.1 Problems with the Current Approaches

Reconfigurable and general-purpose high-performance computing architectures have their own strengths as well as their own problems. Since the strengths are well known and the architectures have been proven successful, we focus on the limitations of these classes of systems. Fortunately, the limitations are quite complementary, which makes the notion of a hybrid architecture model that combines the two very attractive. This, in turn, creates a great need for an execution model that makes the two classes of architectures work together seamlessly.

Problems with reconfigurable architectures are centered around 1) reconfiguration flexibility, 2) scalability and 3) ease-of-use. Many recent systems and projects have recognized the compelling need for dynamic (or run-time) reconfiguration as the means to achieve better utilization and higher performance/cost ratio using reconfigurable hardware. However, current reconfiguration methods are not fully dynamic. Although reconfiguration in these systems does happen at run-time, it follows a fixed (static) schedule that has been determined off-line. These approaches can suffer in the general multitasking cases as well as in the cases of single tasks that are data dependent, where the actual processing needs depend on the dynamic and random field data or on the randomly arriving task mixes, and therefore processing requirements are not very deterministic.

While simple extensions of dynamic run-time reconfiguration based on static schedules are possible, such solutions would generally be limiting as they need to enumerate, compute, store, select and retrieve all possible complete schedules at run-time. Many clever ideas can be augmented with such extensions to make them better; however, the fact remains that such approach would be hard to implement and to extend to different applications.

The other two weaknesses of pure reconfigurable computing are scalability and ease of programming. In a reconfigurable architecture based on field programmable gate arrays (FPGAs), scalability beyond one board is typically hard to achieve. Due to their hardware programming approach, extending the reconfigurable hardware circuits across multiple boards in arbitrary ways to support the needs of different applications is a significant burden for the users. Furthermore, reconfigurable systems are hard to use for the majority of application developers, as the development of each new application involves creating a unique combination of host software and processing element (PE) hardware designs. The host software would be needed to send control signals and data to the FPGA(s) and read back the results. Substantial circuit design knowledge is always needed, since users would have to thoroughly understand reconfigurable hardware architectures and their APIs (Applications Programming Interfaces). This is necessary in order to specify the data flow within individual boards, among boards over the system bus, between boards and the outside world via I/O ports, and between boards and the host, which makes scalability difficult to obtain. Familiarity with CAD tools such as schematic capture or textual hardware description languages to specify the needed configuration for the FPGAs has been a must.

On the parallel systems side, designs have traditionally followed two alternative architectural paths. The first, fine-grain architectures, is to assemble a large number of simple computing elements that perform local simple functions, providing in the aggregate a powerful computing engine. This was the philosophy of the SIMD machines. Local operations in such architectures can be executed very efficiently, due to their inexpensive communications overhead for short local messages. In fact signal and image processing applications have substantially benefited from such fine-grain machines. Global communications for such huge number of small elements has not been very affordable, and a real limiting factor for scalability. Furthermore, there are no COTS fine grain processors and vendors had to develop their own end-to-end solutions. Reconfigurable computing is potentially positioned to provide such COTS fine grain parallelism for high-performance computing if its integration and use in these machines can be simplified.

The second high-performance computing approach uses a relatively smaller number of more powerful coarse-grain computing nodes, integrated with a capable interconnection network. While communication costs for the large messages in coarse-grain computations can be amortized over the message sizes, small local communications transactions are very costly due to the relatively high communications latency. General-purpose high-performance computers are more

easy to program, however, than reconfigurable computers, and can be used efficiently by application scientists and engineers without substantial knowledge about hardware.

Thus, a hybrid system based on combination of these two classes can conceivably embody the better of these two complementary worlds, where potentially the strengths of each of these architectural classes can be used to eliminate the weaknesses of the other.

2.2 Related Work

Many researchers have followed different routes to improve upon the programmability and the efficiency of reconfiguration of systems of hybrid reconfigurable and conventional processing capabilities. On the programmability side, current approaches relied on high-level to VHDL or hardware configuration compiling [HUT99][BAN00][GOK95]. Starting with code written in JAVA [HUT99], MATLAB [BAN00], or C [GOK95] the compiler produces hardware description. Athanas et.al. suggested a framework for translating C programs into dataflow representation and then mapping it onto multi-FPGA platform by dividing the graph into pools that can be implemented in hardware (FPGA) and others that must be executed in software (general-purpose processors) [PET96][ATH93a][ATH93b][WAZ93]. Similar framework that integrates both conventional and hardware compiler and scheduling functions onto the processing elements was suggested for MATLAB in [BAN00]. We believe that the success of such approaches can be limited for two reasons. First, due to conditional statements it will be hard to predict at compile time which configurations to load at run time, which can result in inefficiency. Further, dealing with different boards and integrating new ones into such compilers can be costly. Other work that focused on run time reconfiguration only considered determining needed schedules offline and performing such reconfigurations at run time [BRU01]. Such techniques also suffer many limitations due to data dependencies and non-deterministic task requirements. Therefore, there is a real need for exploring the design space of run-time reconfiguration, where reconfiguration decisions, or at least a significant part of them, are made dynamically. This again becomes a must when we consider data dependent tasks, multi-tasking, or multi-user scenarios, where the type and mix of tasks at any given point of time is not known offline.

In a different but related works, we have considered the problem of harnessing the unused cycles in networks of workstations that have reconfigurable coprocessors [REC][DEV03]. By extending job management systems we have demonstrated how reconfigurable boards can be recognized as resources and integrated for monitoring and allocation to provide a high-throughput distributed reconfigurable environment [Gaj02][ALX01]. In such environment it is assumed that many separate jobs need to be deployed over separate nodes in the network. Although we demonstrated hundreds of folds speed improvements for embarrassingly parallel applications, such as cipher breaking [MIC03], over conventional processors alone, this technique is not adequate for more general problems that would need tightly coupled reconfigurable clusters where multiple-grain parallelism and response time are of interest.

Therefore, we are proposing the ARC execution model and architecture as described below.

2.3 Goals of the new architecture

ARC is proposed as an architectural model that defines a programming view and an execution paradigm for reconfigurable computer clusters aimed at achieving the following goals:

1. **Ease of Programming:** Reduce the programming model to that of general-purpose conventional and parallel machines [C++, FORTRAN, and MPI for example].

2. **Seamless Heterogeneity and Interoperability:** Provide for seamless interoperability between reconfigurable hardware and conventional processors, and between different board and node types in multiple board/node solutions
3. **Portability:** Make application code independent of the specific reconfigurable boards or microprocessors used to build the hybrid nodes. Provide some portability between reconfigurable and conventional computers.
4. **Fixed and autonomous adaptability:** Provide effective model(s) for static and autonomous dynamic reconfiguration to enable reuse and maximize the utilization of the reconfigurable resources.
5. **Ease of Upgrading**
Just as in Beowulf clusters themselves, this open-architecture model will enable easy integration of newly developed reconfigurable boards and microprocessor based workstations
6. **Parallelism Support and Scalability:** enable the synergism between both high-performance general-purpose computing and reconfigurable computing, as well as the underlying application needs to
 - a. Exploit fine-grain parallelism through reconfigurable computing
 - b. Exploit coarse-grain parallelism through microprocessors
 - c. Optimize the overall use of reconfigurable hardware and microprocessors,
 - d. Leverage the progress in high-performance cluster computing to provide solutions that extend smoothly beyond one board or one node.

3.Approach- Concepts and Investigations

3.1 Hardware Architecture

The architectural view under the ARC system is one or more nodes, where each node contains a general-purpose COTS (Commercial Of The Shelf) microprocessor(s) and a reconfigurable engine of one or more COTS reconfigurable boards. Each reconfigurable board should typically have a number of FPGA chips and memory modules interconnected together. A minimum system would contain only one node composed of one microprocessor hosting one reconfigurable board. The system can be scaled by adding boards to this node and/or adding more nodes and interconnecting them, using a high-speed COTS interconnection such as InfiniBand or Myrinet™. Other system networks, such as Infiniband, can be also used. In addition to the operating system, the microprocessor on each node runs the middleware run-time reconfiguration manager (RTRM), which is the key behind hiding the hardware details and the system efficiency. The microprocessor is also used for configuring the FPGA chips. The architecture envisioned by ARC is heterogeneous in two different dimensions. At the intra-node level, ARC uses both conventional and reconfigurable hardware. The microprocessor runs the system functions and coarse-grain application computations while the FPGA boards are configured to perform the data intensive fine-grain processing. Hardware pipelining, systolic arrays, and SIMD architectures are two of the ways of exploiting such fine-grain parallelism at that level. ARC is also heterogeneous at the inter-node level, where multiple boards or multiple nodes of different type can be used. The system can be viewed at that level as a heterogeneous MIMD system.

3.2 Programming Model

Unlike other elements of this proposal, the thrust of the programming model is not to provide new innovative paradigms to the user, that they need to learn and master. The important issue at hand is to maintain the familiar view seen by application scientists and programmers of conventional and high-performance computers, for example the use of C, C++, FORTRAN, and MPI. This also reduces the development life cycle of applications developed for reconfigurable computing platforms. This is due to the fact that the RTRM hides the hardware reconfigurations presenting

the user with a simple application library interface. The RTRM also carries out the data handling processes, such as argument passing and run-time binding, transparently. Hence, both sequential and parallel processing at the intra-node and the inter-node levels can leverage familiar sequential and parallel programming paradigms and tools developed over the years. To this end, a program can be developed in FORTRAN or C++. If more than one board or nodes are used, the application developer can use a standard message passing paradigm, such as the message passing interface (MPI). At the reconfigurable hardware level, seasoned engineers can develop and continually upgrade the function library, which contains the fine-grain processing basic building blocks (such as FFT, edge detection, or Wavelet decomposition for example) independent of the applications. Applications only deal with the application program interface (API) for the library. In this library, hardware pipelining, SIMD, and systolic array architectures will be used to exploit the available fine-grain parallelism. A simplified view of the programming flow, is presented in the Figure 2. Besides the parts shown in the figure, compilers for general purpose programming languages are used for the conventional code.

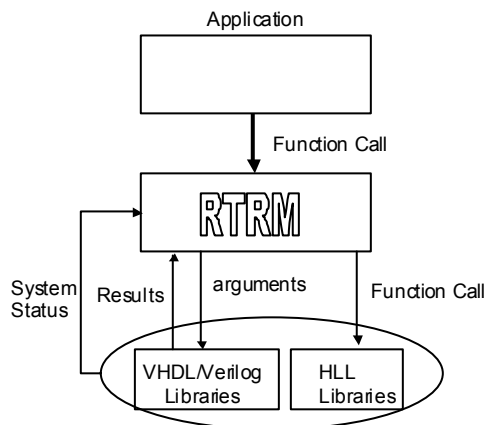


Figure 2: Software Architecture

The input to the system is a high level program written in sequential language like C, or parallel programming language like MPI, with calls to some functions from a predefined domain-specific library. Each function in the library has two implementations, one pure software that runs on general purpose processors and the 2nd is a hardware implementation for FPGAs.

The RTRM job is to get the arguments from the function, allocate the necessary hardware (FPGA or general purpose processor), and get the results. The RTRM then passes the result back to the function. By allocating the required resources, the RTRM must take into account the load balancing, the available resources, etc. This is done by using a superscalar-like method of scheduling (like the scoreboarding for example), as will be shown later.

3.3 Operation and Execution Model

One major objective of this project is to develop a complete and consistent low-level execution model that establishes the operational relationship between applications, the RTRM, and the hardware, both general-purpose microprocessor(s) and reconfigurable device(s). The operation and execution model includes internode and intranode operations in the static, dynamic and scheduled modes, as well as the enabling concepts and techniques.

The fact that applications utilize the reconfigurable hardware by calling an application library of basic functions, where the underlying operations for such functions are implemented using the reconfigurable hardware, means that such library can be also implemented using high-level programming languages. The presence of such high-level language mirror library can allow application portability between the reconfigurable and conventional computers.

Operational Modes: The operational modes for such reconfigurable clusters and the enabling concepts and techniques are discussed below. It should be noted that since the RTRM reduces the node programming to that of a standard high-level programming language, the inter-node operation follows directly the high-performance MIMD scheme and the use of message passing paradigms such as MPI becomes straightforward. Therefore, we only describe the node operation.

The key component of this project is the run-time support system (RTRM). RTRM allows seamless integration of the reconfigurable engine and the general-purpose microprocessor, and hides the hardware reconfiguration and utilization details from the user. Under the ARC system, when an application calls one of the processing library functions that are implemented using FPGAs, the RTRM receives such a call. The RTRM then manages the delivery of the input parameters to the corresponding FPGA configured device as well as it receives and returns the result to the calling application.

Static Node Operation

Static node operation is well suited for domain-specific executions. In the static mode, all configurations are done at machine initializations and remain in place till execution stops. Given the fact that not all functions may fit into the available FPGA devices, static profiling [BAL94][MAT95] of benchmark suite(s) from the application domain, including the applications themselves if known, will be used. After profiling, the best static configuration for each class of applications is used. Profiling can be performed on HLL versions of the benchmark to understand the function requirements. Profiling of some of these workloads in reconfigurable hardware form will be necessary in some of the cases to parameterize the application requirements in this form.

At run time, the designated configuration is applied during machine initialization. However, dynamic resource management is still needed in order to manage the FPGAs and processors and assign modules to them. This dynamic management is necessary because of the changing workloads across applications, and the fact that not all hardware library components may be instantiated. The RTRM keeps track of the machine resources, and using scoreboarding-like technique, assign modules to either FPGAs or processors, and manages the forwarding of data between modules. Superscalar processor techniques such as scoreboarding and Tomasulo's algorithm [SHE03] will be investigated and extended (or used to inspire new techniques) to keep track of which functions are assigned to which FPGA or processor, which data is needed by each function, ensure load balancing, as well as the availability of resources. Based on these data, RTRM will manage the assignment of functions to processing cores. Previous work concentrated

on totally static methods, using the compiler [BAN00] or the user defined schedules, no dynamic management of resources was done.

Scheduled (Phase-Based) Node Operation

In order to improve the utilization of the reconfigurable hardware on each node and reduce cost, one FPGA hardware resource can be configured successively to cater to a sequence of functions. Such hardware reuse will be clearly possible if the execution of such functions was known to follow a fixed and simple enough schedule. The presence of such opportunities for hardware reuse in the underlying application(s) is determined from application(s) profiling or expert knowledge about the application. For example in image registration applications we know that the processing sequence starts with thresholding, then edge-detection, trans-rotation, correlation, and finally correlation peak detection. Systems currently present are using totally static schedules. These scheduled operations are either user defined, or compiler generated [BAN00].

In our scheme we propose using dynamic in-flight scheduled operation. We still assume a general-purpose setting, where applications will change dynamically, however, we assume here that there will be many processing locality patterns that can be detected and exploited. For in-flight scheduled operation, the RTRM will detect the start of such sequences and configure the target FPGA chips accordingly before they are needed, perhaps alternating between the chips to hide the reconfiguration latency. Here, we need to borrow prefetching techniques [VAN97] used essentially in cache memories, modify them and use them to prefetch the configuration before needed [ZLI02]. In order to help prefetching, static profiling is needed.

From profiling, one can collect statistics and perform data mining tasks to construct relationships among the functions that can be configured and establish functional working sets and execution phases. Using these information we can develop representative signatures for such working sets and execution phases to be able to recognize them easily at runtime [ZOM99][DHO02]. These statistics, mined rules and signatures can be then used to guide how the workload must be distributed among general purpose processors as well as reconfigurable cores in a way to make use of the strength of both schemes as well as maintain load balancing and hide reconfiguration latencies.

A way of doing static profiling which is pertinent to our goal here consists of dividing an application into *meta-phases* [LAU04]. In [LAU04] it has been found that several program execution intervals exhibit similar behavior. Hence these intervals can be grouped into meta-phases. We want to extend the work in [LAU04] by profiling an applications and dividing them into meta-phases not only based on some performance metrics (such as memory behavior, registers usage, etc.) but particularly based on hardware requirements. This means, each meta-phase is characterized by the amount of general processors needed, the types of reconfigurable cores needed, etc. Using these meta-phase requirements, several static schedules are formed at offline. At run-time, the RTRM will detect phase changes, determine in which meta-phase they belong, and start managing and allocating resources based on the static schedule of each meta-phase. To this end, we will investigate several ways of detecting phase changes. For example, we are considering extending the work of Smith and Dhodapkar in [DHO02] and using frequency vectors (histograms) about the frequency of execution of a specific function and whether it needs one or more FPGA or only a general purpose processor. Figure 3 shows a possible extension of the working set signature presented in [DHO02] by including frequency vectors. As indicated in the figure, each entry in the bit vector table contains a bit string representation of the amount of hardware needed, etc. This table is built during offline profiling, or can be built dynamically through online profiling. Many issues need to be considered here, such as, how to represent the

information efficiently in bit strings, what is the minimal amount of information to collect to accurately perform phase detection, etc. The phase change detector, shown in the figure, stores the bit strings of the previous phase, and based on the current phase bit vector, it detects whether there is a phase change. In case of phase change, the bit vector is used by the RTRM to perform the required configuration. Torellas et.al. presented a new way of detecting phases by monitoring not the behavior in past cycles but using the part of the program currently executed, hence they used positional approach instead of the conventional temporal approach [HUA03]. We are planning to investigate this method by extending it with hardware requirements for each part of the program. Moreover, we are planning to try to integrate both the signature scheme as well as the positional approach in order to get better phase detection.

Detecting phase changes, as well as the meta-phase each phase belongs to is the first step toward efficient execution. The next step is to make that part of the program in the specific phase into the adaptive architecture. Extensive work has been done both in academia and industry toward mapping applications into reconfigurable architecture. For example in [PUR99] the authors introduce the concept of temporal partitioning to partition a task into temporally interconnected subtasks. They present algorithms for temporal partitioning and scheduling data flow graphs (DFG) for configurable computers. Computations with logic area requirements that exceed chip size cannot be completely mapped on a configurable computer (using traditional spatial mapping techniques). However, a temporal partitioning of the data flow graph followed by proper scheduling can facilitate the configurable computer based execution. Scheduling assigns an execution order to the partitioned segments such that the computation would execute in proper sequential order as defined by the flow graph. The authors in [PUR99] describe various algorithms for partitioning the DFG and mapping it to configurable hardware such as “Level Based Partitioning” and “Clustering Based Partitioning Algorithm”. We plan to enhance on the previous work in the area of mapping programs in to configurable hardware by making use of current system status, run-time statistics (such as load balancing for example), to help the RTRM doing more efficient mapping and adaptation.

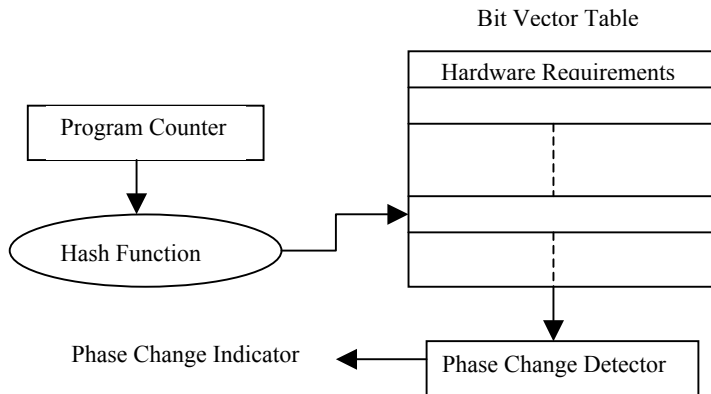


Figure 3: Phase Change Detector

Dynamic (Autonomous) Node Operation

In the dynamic mode, the ARC system becomes fully adaptive and all hardware configuration and re-configuration is decided by RTRM at run-time, to optimize utilization and speed.

This mode is useful in scenarios such as multi-user multi-tasks, in which case profiling cannot be done easily. RTRM must become aware of the current resources and capable of anticipating the upcoming processing requests with a high degree of accuracy. In this case, the RTRM will

manage the reconfigurable resources via a cache-memory-management-like model. By keeping track of *processing locality*, RTRM will anticipate the next function to be executed on an FPGA based on the current functions. This temporal behavior helps starting the reconfiguration before it is needed, hence helps in hiding reconfiguration time. Furthermore, the spatial locality of functions, helps the RTRM assigns the functions that are dependent near each other, in this case forwarding of data between functions will be done in a fast and smooth way. This locality characteristic also helps in deciding which module to replace in case another module needs to come in. Replacement will consider issues such as most frequently used and most-recently used older module. The locality characteristic will be exploited by extending cache-like techniques [JEO03] and will consider those existing strategies only as starting point.

Therefore, RTRM is equipped with some intelligence using a number of innovative concepts. Some of these concepts are extended from cache memory, and others from superscalar and high-performance processor design and used for the first time in reconfigurable hardware, and some are defined here for the first time, such as processing locality and dual-track execution. Some of these concepts are described as follows.

Selective Dual-Track Execution and the Role of the Microprocessors: ARC dynamic adaptability calls for adapting the use of both reconfigurable computing and conventional microprocessor resources at run-time to achieve the best possible performance. One aspect of this adaptability is to allow the conventional processor to elect at run-time to perform some of the functions that are intended for execution on the reconfigurable engine, when such arrangement can enhance the overall performance. This concept will be implemented and experimented with. Important cases in which such feature would be advantageous are:

1. First call: when a function has just been called and it is not configured yet on the reconfigurable engine, it would be of interest to perform the first execution on the conventional processor, while the reconfiguration proceeds in the background for subsequent calls. This feature is clearly used if for instance the execution on the conventional processor takes less time than the time for reconfiguration.
2. Load Imbalance: when the reconfigurable engine is heavily loaded and the conventional processor has free cycles to use, it would be helpful to shift some of that load to the microprocessor.

In addition to these scenarios in which the microprocessor could be selected by the RTRM to perform computations instead of the reconfigurable engine, the **microprocessor** always perform the following two additional **roles**:

3. Special Functions: in cases where the use of the conventional microprocessor could be more adequate than using the reconfigurable engine, the microprocessor will be always used to perform specific computations. Examples of such cases are floating point computations that could be faster on conventional processors, and some high-level decisions that are not data intensive or lack data parallelism.
4. Running the RTRM Code: The conventional microprocessors last responsibility is running the run-time reconfiguration manager middleware.

Selective dual-track, as explained in items 1-3, will be decided by the RTRM at run-time based on parameterized performance cost functions. Cost function parameters will be determined from the workload characterization of our selected pool of applications.

Processing Locality: Locality of references has been used to provide high average memory bandwidth by predicting future accesses and ensuring that the majority of them are satisfied at the cache level. Here, we define a parallel concept, which can be very useful in the context of reconfigurable adaptive processing. We call this processing locality and we define its temporal and spatial elements as follows. Temporal processing locality would refer here to the fact that functions that have been used in the past would be needed in the future, which arises from repetitive programming constructs such as loops. Spatial processing locality would arise from functions that are typically used together in a given application, such as openings and closings in morphological image processing, or convolution and decimation in Wavelet decomposition. Using these concepts, the RTRM can manage the reconfigurable computing resources as if it were managing a cache memory, making sure that most processing requests to the function library would be satisfied by the reconfigurable hardware. In our case functions currently executing represents cache memory slots. Strategies for discovering and exploiting such localities at run-time can be extended from cache and memory management techniques[JOS99][PAT95][SMT82], as well as from data mining[ZOM99]. For example, the replacement algorithms can be extended with measurements such as load balancing, frequency of function usage, etc. Therefore, when all reconfigurable cores have already been reconfigured, and a new function needs to be executed, a replacement algorithm needs to be applied [JEO03]. Also reconfigurable cache blocks will be designed to maximize processing spatial locality by considering how functions associate with one another based on the profiling.

Forwarding and Chaining in Reconfigurable Architectures: Forwarding and chaining are concepts that have been used heavily in vector supercomputers. Internal data forwarding refers to replacing memory accesses with register transfer operations. In ARC, internal data forwarding is similar in concept, but will forward data from one functional unit to the following one, without having to place the intermediate data back into the memory of the host processor. Contemporary reconfigurable computing boards such as the WILDFORCTM and WILDSTARTM have off-chip memory, such as the Mezzanine card in the WILDFORCETM, which can be used by RTRM for forwarding data from one configuration to another. Chaining, in pipelined processors, refers to feeding the output of one pipeline to the input of another, to increase the pipeline depth and therefore the throughput. The scoreboard information can allow RTRM to cite and exploit opportunities for such optimizations.

Load balancing: Unlike superscalars, multiplicity of functional units in reconfigurable hardware can be determined at run-time in order to balance computational requirements. RTRM can determine such a multiplicity factor based on the available resources and the current processing requirements.

Mixed Node Operation

In addition to investigating the aforementioned node operation modes, we will also investigate the issue of integrating these concepts and the rules governing their synergistic interplay by identifying what kind of tasks should be statically configured, which ones should be scheduled based on execution phases and which ones should be configured always dynamically. We will be exploring in this the right mix of such scheduling activities by RTRM and the system such that performance/cost can be maximized.

3.4 Interconnection Network

The interconnection network of ARC can be either fixed network, or *reconfigurable*. The reconfigurable network must be built with the following criteria in mind:

- It can be configured dynamically to reflect data flow between nodes
- Reconfiguration must be done smoothly without much overhead and hidden as much as possible. These requirements raise some issues, such as: what adaptation algorithm to use? Does the algorithm need to be centralized or decentralized? When is the reconfiguration initiated? Profiling comes into play here too. The communication pattern has to be captured in order to be able to decide the best interconnection, based on several criteria like data flow, contention, etc. Furthermore, several techniques can be borrowed from ad hoc network [MCD99]. Although ad hoc networks are introduced for wireless channels, similar techniques can be applied in ARC to direct dynamic interconnection and also to reach better reliability in case node becomes faulty or un-available.

4. Research Plan

4.1 Basic Investigations

The ARC architecture is based on a number of new concepts that have not been previously explored for this heterogeneous reconfigurable framework. These include processing locality, scoreboarding and forwarding and chaining in reconfigurable architectures, and the selective dual-track execution. For example the concept of processing locality has not been defined or considered before as it is of relevance only to our cache-memory-like reconfigurable resource management in adaptive computing. These elements constitute the core ideas behind the reconfiguration manager. Thus, in the basic investigations of this research project:

1. Mechanisms for recognizing and exploiting spatial and temporal *processing* localities will be investigated. While they are expected to have some correlation to memory management techniques, processing is quite different and its different requirements will be studied. Correct and efficient algorithms and data structures will be also investigated and developed for implementing these mechanisms. Augmenting a priori information available about the application will be also considered.
2. Information content and data structure of the scoreboard in such heterogeneous architecture will be studied and developed. The pertaining control mechanisms and procedures to be invoked by the run-time reconfiguration manager will be investigated and constructed.
3. The general software architecture of the run-time reconfiguration manager middleware will be abstracted and developed. Relationship to operating systems and compilers will be thoroughly investigated. Implementation trade-offs will be identified and evaluated.

4.2 The ARC Experimental Testbed

A testbed for the ARC project will be constructed by leveraging one of our currently existing clusters and fitting it with reconfigurable board coprocessors. This recently constructed Beowulf cluster has 18 2MHz P4 processors in 9 dual processor nodes, interconnected via a Myrinet switch. With the equipment funds from this proposal if awarded will be able to acquire addition

Firebird boards from Anapolis Microsystems to populate 8 nodes with reconfigurable hardware for the creation of a testbed for the experimental part of our study.

Applications

We currently have many image processing and cryptography applications for reconfigurable platforms. We will be augmenting this application base with additional ones from NASA remote sensing applications through our collaborations with NASA Goddard Space Flight Center. Currently our effort with NSA involves the development of a suite of FPGA applications, most of which can be leveraged to help our proposed plan

Measures of Success

Evaluate the performance of the cluster without the FPGAs for the underlying workloads and compare with those when FPGAs are used. Provide some performance/cost measures as well.

Project Schedule

Six month	Identify target applications. This includes leveraging/adopting existing and creating complementary applications. Build Experimental Testbed
12 month	Complete workload characterization and cost functions developments
18 month	Complete concept developments as specific algorithms and data structures to support RTRM
24 month	Implementation of RTRM functionality for static operational mode
30 month	Implementation of RTRM for functionality for dynamic operational mode
36 month	Full implementation and Evaluation of ARC

5. Management Plan

Professor. El-Ghazawi will be leading this research program. To this end, he will be supervising all research activities. Professor Alexandridis will focus on issues related to processor design concepts that are extended to FPGA platforms such as scoreboarding and forwarding and chaining.

Two graduate students will conduct their graduate research under this program. Funds are also allocated to support undergraduate student(s) in the second and third years of the project to conduct their two-semester senior graduation project in reconfigurable clusters.

This research program and the equipment that will be acquired under this program will be used to support our High-Performance Computing Lab (HPCL), which is co-directed by Professors Alexandridis and El-Ghazawi. HPCL, hpc.gwu.edu, has already significant reconfigurable computing and cluster computing resources and is designed to provide our graduate and undergraduate students with research experience. We are continuing to build this lab, which has now over a dozen of graduate and undergraduate students conducting degree research or senior design projects, to become a center of excellence for Computer Engineering research and education. At the undergraduate level, Professor Alexandridis teaches, among other courses, senior design project classes. Through his work with this class he brings a steady flow of undergraduate students to do their senior project work using the HPCL facility and in the context

of its external projects with government and industry. Should this award be made, Professor El-Ghazawi will be proposing a graduate level class in Reconfigurable Computing for the computer-engineering program.

The department of Electrical and Computer Engineering Department has been actively involved in the areas high-schools to encourage more students to pursue degrees in our area. The outcome from our research projects is typically an attraction to those prospective students. We plan to focus some of those visits and relations to schools that are highly populated with underrepresented groups. Dr. El-Ghazawi has also an ongoing collaboration with Universit e De Technologie Belford-Monbleard in France. Under this program undergraduate students from France conduct internships in HPCL. Dr. Alexandridis has a similar agreement with several universities in Greece. Through these arrangements we expect that some of those international students will be conducting their internship in the context of the project when they visit.

Our lab, HPCL, has had very strong relationships with NASA GSFC, NSA and other local labs and industry. Our work with NASA and NSA has greatly overlapped the areas of reconfigurable computing and cluster computing and we believe that those communities will benefit greatly from our results. These results will be disseminated directly to our collaborators in those communities and to the general community through publications and an extensive web site, which will house both research results and educational material. Many of the reconfigurable applications developed under our previous and current efforts from other sponsors will be also leveraged in this program.

6. Significance and Broader Impact

If successful the proposed research will develop concepts and foundation for efficient, dynamically adaptive, easy-to-use, embeddable, and scalable reconfigurable cluster systems based on commercial off the shelf (COTS) modules. This enables reconfigurable computing to move to the mainstream computing, where it can ultimately deliver supercomputing performance at a fraction of the cost, and play an important role for industry and government labs by providing an advanced subset of the Beowulf class architectures.

This project, can also enhance undergraduate education with exciting senior design project experiences that can entice them to pursue graduate studies. I will also help incorporate reconfigurable computing systems into graduate curriculum in a way that emphasizes the interplay between hardware and software issues. This helps reconfigurable computing to become a mainstream component and produce graduates who can see the big picture of how reconfigurable computers can fit into larger architectures and the many roles they can play. The graduate course that we are considering and the senior design projects will emphasize that aspect and the resulting research and educational material will be disseminated through the project and the teaching web sites at GWU. In turn, this can help fuel the industry with well-equipped graduates who can address such emerging issues.