Software Architecture for Morphing in Polymorphous Computing Architectures

Daniel P. Campbell
Georgia Tech Research Institute

Mark A. Richards
Electrical & Computer Engineering

Dennis M. Cottel, Randall R. Judd
Space & Naval Warfare Systems Center San Diego

Georgia Institute of Technology, Atlanta, GA 30332

1. Introduction

The Polymorphous Computing Architectures (PCA) program is a Defense Advanced Research Projects Agency (DARPA) effort to develop new embedded computing platforms with very strong, rapid in-mission reconfigurability. Target applications range from military platforms that must adapt to rapidly changing mission parameters, to embedded network controllers whose optimal configuration of hardware resources will change in response to the traffic and environmental conditions they face.

The PCA program "core projects" working to develop microprocessors that implement polymorphous capabilities include Smart Memories, Raw, TRIPS, and MONARCH; references for all are available in [1]. The chips under development in these projects have several characteristics in common. These are typically tiled structures composed from replicated, fully capable computing cores, reconfigurable memory and cache elements, and a rich set of reconfigurable data paths, network interfaces, and I/O paths. Each can operate in streaming or threaded modes. Each has mechanisms for aggregating individual processor tiles into larger compound processor units. They differ in their approach for aggregating processors and in their emphasis on processor, memory, or communication design.

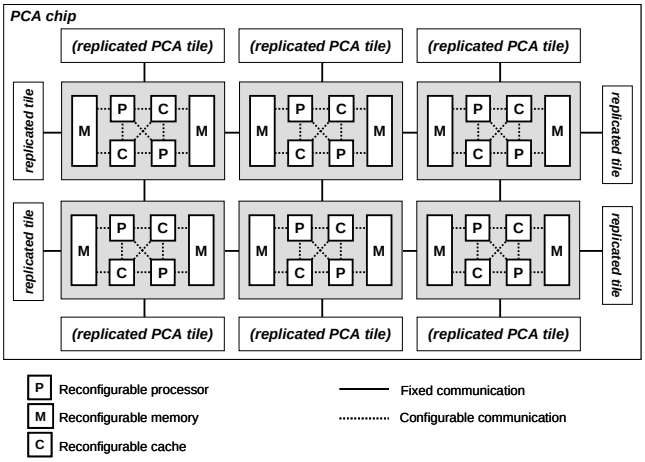

Figure 1. Generic PCA chip micro-architecture.

Figure 1 illustrates a generic PCA microarchitecture.

This ability to aggregate varying numbers and types of elements on a PCA chip means that the chip can be effectively partitioned into multiple processors of similar or different types, with each partition assigned to a different portion of the application program or even to different programs. For instance, one portion of a PCA device could be optimized for stream processing and dedicated to a sensor processing dataflow computation, while another is configured for conventional thread processing and allocated to conventional control processing. Furthermore, the number of chip tiles dedicated to each processor type could be varied based on expected loads.

To exploit the capabilities of PCA hardware while retaining as much end-user portability and performance as possible, the Morphware Forum (www.morphware.org), an informal consortium of the PCA contractors and other selected participants, is creating an application development framework, called the Morphware Stable Interface (MSI). An overview of the MSI is available in [1].

A key capability envisioned for PCA systems is *morphing*, the reconfiguration and re-allocation of PCA hardware resources within a chip in response to various events.. Morphing is fundamentally enabled by the reconfigurable hardware microarchitecture of PCA chips, but is made accessible to the programming environment through the MSI. Thus, a major software design issue for the PCA program is how to structure the MSI so as to support morphing while maintaining portability across the various PCA targets.

2. Types of Morphing

The MSI envisions a component-based application software architecture. Components provide natural and intuitive boundaries for run-time reconfiguration of hardware. In general, multiple implementations of various units of functionality (*e.g.*, a fast Fourier transform) will exist as different components, each offering different trades of performance and system requirements, and capable of being compiled to differing amounts of hardware resources. Morphing then implies changing either the particular component implementations in use, the resources assigned to the components, or both. Different types of morphing can thus be classified based on three orthogonal characteristics:

- whether the application code directly makes an application programming interface (API) call to initiate morphing, or it is done invisibly to the user by either the run-time system or the compiler;
- whether the component continues to execute or is reloaded or replaced with an alternate component; and
- whether the resources allocated to the application must change or stay the same

The Morphware Forum has developed a taxonomy of morph types, summarized in Table 1, to describe the various situations.

Table 1. PCA Morphing Taxonomy

	Run-time System		Application Programmer		Compiling System	
	Components continue	**Components change**	**Components continue**	**Components change**	**Components continue**	**Components change**
Resource allocation doesn't change	*Type 0a*	*Type 1a*	*Type 2a*	*Type 3a*	*Type 4a*	*Type 5a*
	Run-time environment changes transparently to the running application.	Run-time system changes components to reconfigured but equivalent set of resources.	Application makes API call to make suggestions.	Application makes API call to change processing mode but does so within existing resource set.	Compiler instructions reconfigure allocated resources.	Compiler switches to a different library able to use the same resources.
Resource allocation changes	*Type 0b*	*Type 1b*	*Type 2b*	*Type 3b*	*Type 4b*	*Type 5b*
	Run-time system changes resource allocation of a running application transparently to the application.	Run-time system configures resources and loads components at application startup.	Application makes API call to give up or gain some resources.	Application makes API call to add or replace one or more components using different resources.	Compiler requests different resources to meet change in performance specified by metadata.	Compiler switches to a different library that uses different resources.

3. Morphing in the MSI

3.1 Compilation Architecture

Portability across alternative PCA target devices is obtained in the MSI by using a two-level compilation architecture, as shown in Figure 2. The application program is partitioned by the user or by tools yet to be developed into streaming units and non-streaming (conventional) units. The former are expressed in a specialized streaming language such as Brook or StreamIt, while the latter are in conventional C or C++ code. The *high level compiler* (HLC) takes in this user source code as well as a *machine model* (MM), a metadata description of the resources available on the PCA devices to the compilation unit and their configuration.

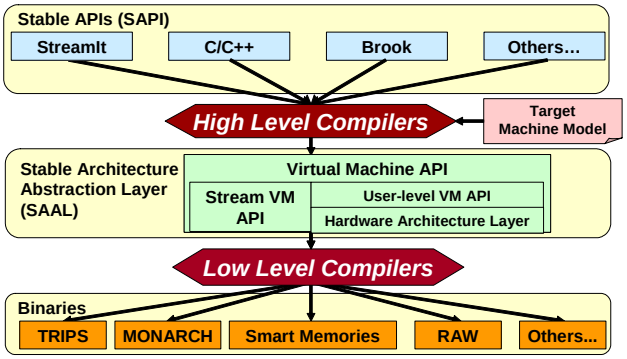

Figure 2. MSI Compilation Architecture.

The HLC compiles the streaming input units to a *stream virtual machine* (SVM) description. The HLC utilizes the information in the particular MM provided with the source code to optimize the coarse-grain parallelization of the streaming program unit. Thus, the same application code will produce different SVM codes, depending on the machine model description of the available resources. This mechanism provides the basic capability for portability across multiple target machines, as well as the capability to vary the amount of resources assigned to a functional unit within the same machine.

Threaded code is expressed in terms of a *thread virtual machine* (TVM), in turn composed of a *user-level VM* (UVM) and a *hardware architecture layer* (HAL). Other than expressing the output in these machine-neutral APIs, the HLC largely passes threaded code to the output without optimization. The machine-specific *low level compilers* (LLCs) then compile the VM code for their particular target PCA machine, performing further fine-grained parallelization and optimizations as appropriate.

3.2 Morphing Mechanisms

As seen in Table 1, some morphing operations are defined by the compiler, while others are controlled by the run-time system, most likely in the form of a yet-to-be-defined resource manager. Thus, morphing is actually implemented by various levels of the MSI, depending on the type of morph. Compiler-directed morphs can be the result of changes in the machine model for the target hardware, of coarse-grain optimization decisions made by the HLC, or of fine-grain configuration decisions made by the LLC. Morphs that change the executing components must be initiated and controlled by the run-time system of the PCA machine.

Several methods for representing the various forms of morphing have been proposed, including modeling morphs as program branches, explicit control of variables representing machine state at the SAPI or SAAL levels, and marking sections of code with performance and resource constraint expressions. The candidate approaches to date will be described and compared, considering such issues as the level of the MSI at which they are implemented; granularity; and visibility to and controllability by the programmer Selecting and vetting the appropriate interfaces to represent morphing is currently a primary focus of the Morphware Forum.

4. References

[1] The Morphware Forum, "Introduction to Morphware: Software Architecture for Polymorphous Computing Architectures", version 1.0, Feb. 23, 2004. Available at www.morphware.org.