The Morphware Stable Interface: A Software Framework for Polymorphous Computing Architectures

D. Campbell¹, D. Cottel², R. Judd², K. MacKenzie³, M. Richards⁴

¹Georgia Tech Research Institute, Smyrna, GA
²U.S. Navy SPAWAR Systems Center, San Diego, CA
³Reservoir Labs, Inc. New York, NY
⁴Georgia Institute of Technology, Atlanta, GA
Acknowledgements
Polymorphous Computing Architectures

- DARPA effort for high performance embedded platforms with strong, rapid, reactive in-mission configurability
  - Support dynamic and multi-mission requirements
  - Support collaborative, information-centric missions

- PCA will develop processing architectures that “morph”
  - Hardware and software resources reconfigure to balance resource requirements and availability
    - at multiple levels: micro-architecture, network, system
    - at multiple time scales: in-mission, between-mission
Generic PCA Microarchitecture

PCA chip

(replicated PCA tile)  (replicated PCA tile)  (replicated PCA tile)

(replicated tile)

(replicated tile)

(replicated tile)

P: Reconfigurable processor
M: Reconfigurable memory
C: Reconfigurable cache

---

Fixed communication

Configurable communication

High Speed off-chip I/O
Generic PCA Microarchitecture

**Tiled structure**
- Fully capable computing cores
- Configurable memory and cache
- Configurable data paths, network interfaces, and I/O
- Streaming and Threaded modes
- Methods to aggregate tiles into larger processing units

**Core projects differ in**
- aggregation mechanisms
- relative emphasis on processor, memory, or comm design

**Performance on the order of (per chip)**
- 4 – 64 GFLOPS / 4 – 16 GOPS
- 25 – 32 GB/s off-chip I/O
Increased hardware flexibility and complexity brings increased software complexity

- If we build target platform reconfigurability and performance info into the application, we lose scalability and portability
- If we don’t, the build and run-time systems will be entirely responsible for leveraging the platform capability, and we still lose fine-grain morphability
- Applications must be reactive to feedback from the hardware
  - resource collisions, SWEPT, faults

Solution: the Morphware Stable Interface (MSI)
The Morphware Stable Interface (MSI)

- Application Development Framework for PCAs
- Comprised of a software architecture and a suite of open standard APIs

Goals

- Dynamically optimize PCA resources for application functionality, service requirements, and constraints
- Obtain nearly optimal performance from PCA hardware
- Be highly reactive to PCA hardware and user inputs
- Manage PCA software complexity
- Leverage existing and already-developing technologies

Cross-project effort, developed in parallel with the hardware
Informal consortium of the PCA contractors and other selected participants

Organized and led by the Georgia Tech/SPAWAR team

Meets quarterly
  - interim meetings and activities as required

Propose, debate, develop, test, validate, document, and demonstrate standards that define the MSI
The Morphware Forum

~ Applied Photonics
~ Georgia Institute of Technology
~ George Mason University
~ IBM
~ Lockheed Martin Company
~ Massachusetts Institute of Technology
~ MIT/Lincoln Laboratory
~ Mercury Computing Systems
~ Mississippi State University
~ MPI Software Technology, Inc.
~ Northrup Grumman
~ Reservoir Labs, Inc.
~ Raytheon
~ SPAWAR
~ South West Research
~ Stanford University
~ University of Texas - Austin
~ University of Illinois
~ University of Pennsylvania
~ University of Southern California
~ Vanderbilt University
Stable API (SAPI) and Stable Architecture Abstraction Layer (SAAL) provide dual portability layers.

Application SW describes functionality, constraints, and performance requirements.

SAPI is PCA-aware collection of standardized language and service APIs.

SAAL is PCA-aware abstracted low-level machine representations.
Why SAAL?

- Traditional languages based on a machine model increasingly incorrect
  - Single program counter
  - One operation at a time
  - Data universally local

- All modern high performance computing systems battling this issue

- In order to exploit new hardware, core teams developed new languages not based on old model

- New languages based on similar models

- Formalize the models to make it explicit
Stream Languages

- Compute-intensive portions of many applications have characteristics of stream operations
  - fixed data flow graph
  - large, possibly infinite, data stream
  - functional kernels not data-dependent
  - functional kernels independent of one another
  - little or no retained data or state

- Representations that enforce these characteristics ideally suit PCA architectures, aid compiler in
  - Optimization
  - Scheduling
  - Resource allocation
  - Data Locality
Morphware Languages

Stable APIs (SAPI)
- StreamIt
- C/C++
- Brook
- Others…

Stable Architecture Abstraction Layer (SAAL)

Binary Executables
Traditional languages have an implicit SAAL layer

MSI has an explicit SAAL layer, a portable API that exposes the virtual resources typical of PCA systems
- Sacrifices some tool chain flexibility for simpler, more defined, more analyzable build chain
- Factors deployment of new languages and hardware
- Allows explicit consideration of model of computer
- Formalizes and augments existing model

Creates a two-stage compile process

Example constructs: kernel, stream, processor, etc
Morphware Compilation

Stable APIs (SAPI)
- StreamIt
- C/C++
- Brook
- Others...

High Level Compilers

Stable Architecture Abstraction Layer (SAAL)
- Virtual Machine API
  - Stream VM API
  - Thread VM API

Low Level Compilers

Binaries
- TRIPS
- M-Chip
- Smart Memories
- RAW
- Others...
Metadata in Morphware

- PCA Hardware is complex and changing
- PCA Missions are complex and changing
- Large amount of configuration, constraints, requirements, etc. information in addition to functional requirement

- Extracting and encapsulating this information
  - Increases portability, scalability
  - Facilitates Reconfiguration, Repurposing, Redeployment
  - Is an important goal of most modern software systems
Metadata System

- Metadata needed throughout the PCA system
  - Several contexts
  - Consistent method of representation & query preferred
  - Needed to enable processor and compiler developers to progress

- Current system stores metadata as XML
  - Metadata is expressed as relational, hierarchical object oriented structure
  - Instantiated as XML
  - Contexts are defined by a Schema and Documentation
  - Accommodates procedural or static representation queries
  - Accessible to wide range of API’s, tools, etc.
Use of Metadata

Stable APIs (SAPI)
- StreamIt
- C/C++
- Brook
- Others…

High Level Compilers

Stable Architecture Abstraction Layer (SAAL)
- Virtual Machine API
  - Stream VM API
  - Thread VM API

Low Level Compilers

Binaries
- TRIPS
- M-Chip
- Smart Memories
- RAW
- Others…
Reservoir

Platform Description Context

- Needed by HLC to improve VM output
  - Helps allow coarse grain partitioning of applications into appropriate sized pieces
- Nearly complete, minor fixes remain
- Describes target platform using common dictionary of virtual resources and attributes
  - Processors: type, frequency, max-IPC, latency…
  - Memories: type, size, cache-linesize, associativity…
  - Net-Links: senders, receivers, latency, bandwidth
Dynamic Configuration

- Model so far good for flexible resources, goals & constraints
  - Two level compile
  - Structured VM code
  - Well defined metadata
  - Good compilers

- Dynamic resources, goals, & constraints much harder problem
  - Builds have (nearly) infinite time to analyze & search the solution space, run-time changes must happen quickly
  - Static, configurable build parameters a hard, but tenable task
  - Support for dynamic criteria explodes the solution space
Alternate Monoliths

- Build with several parameters
  - Traverse build chain with a defined set of constraints, goals, resources expected
  - Deploy binaries for each set
  - Select the best-fit binary at run-time

- Benefits
  - Build chain sooner
  - Easier problem, faster builds
  - Known, testable states
  - Better optimization for known states

- Problems
  - Problems with unexpected hardware states
  - Not as flexible as the hardware
  - Only optimal for expected states
Component-Based Approach

- Flexibility & resilience gained by partitioning physical resources
- Configure each partition independently
- Build binaries for each partition in various states

Benefits:
- Smaller problem makes flexible build criteria more feasible
- Hierarchical approach factors the problem of resource management
- Able to match run-time needs more closely
- Able to achieve top performance in more situations
- More easily respond to hardware failures & changes

Problems
- Requires a more robust run-time system to fully exploit
- Many states possible – complicates testing
- Framework bloat
Component-Building

Stable APIs (SAPI)
- StreamIt
- C/C++
- Brook
- Others...

Component API

High Level Compilers

Stable Architecture Abstraction Layer (SAAL)

Virtual Machine API
- Stream VM API
- Thread VM API

Low Level Compilers

Components
- TRIPS
- M-Chip
- Smart Memories
- RAW
- Others...

Resource Subset Description
Goals/Constraints
Component Metadata
Morphware Forum Steps

_priority: End-to-End framework that allows an application that can reconfigure its platform

Immediate priorities:
- Finish TVM
- Finish HWMD
- Define HLC / LLC Interaction
- Determine run-time services
  - Load, unload, configure, measure, etc.
  - Consider component-based approaches

Continue regular activities
- Quarterly meetings, interims, draft documents, etc
The Morphware Forum web site provides some public information

- Selected public papers & briefings
- Links to PCA project sites and related links
- Link to DARPA PCA
- This paper and presentation, soon

In the future, it will provide one-stop public dissemination of MSI documents