Support for Programming Reconfigurable Supercomputers

Miriam Leeser
Nicholas Moore, Albert Conti
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA
Laurie Smith King
Dept. of Mathematics and Computer Science
College of the Holy Cross
Worcester, MA

Northeastern
UNIVERSITY

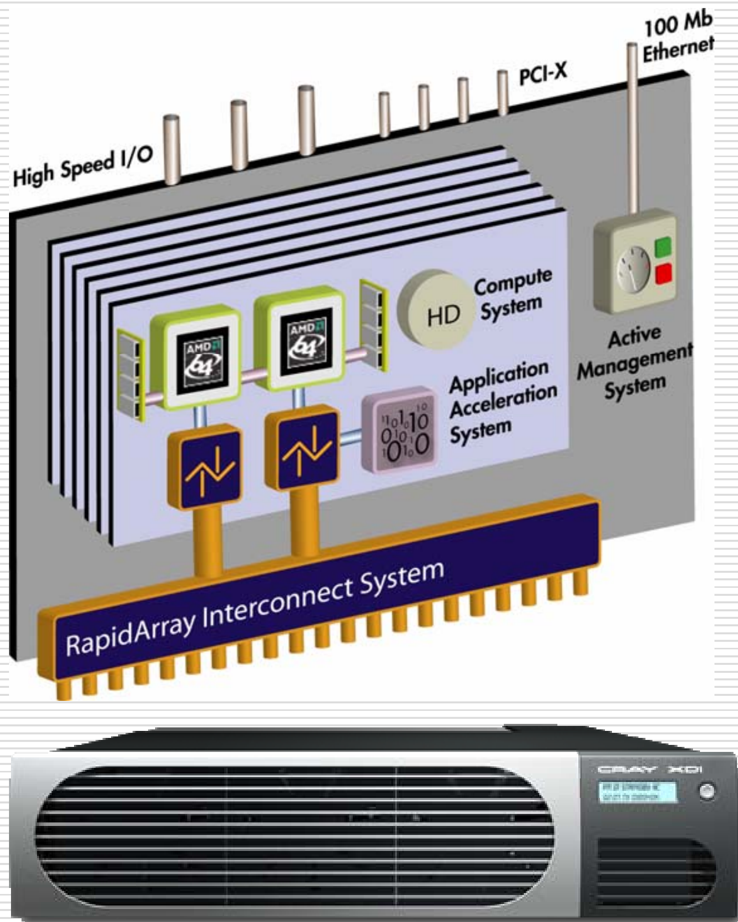
Outline

- Reconfigurable supercomputing (RS) architectures
- State-of-the-art in programming RS
- VSIPL++ and support for Special Purpose Processors (SPPs)
- The Run-time Resource Manager for supporting SPPs in RS
- Future Directions:
  - Run-time tools for RS
  - Support for “write once, run anywhere”

State of the Art: Development for Reconfigurable Supercomputers

- Multiple different hardware platforms, all containing:
  - Processors
  - Reconfigurable fabric
  - Memory: SRAM, DRAM
  - Interconnect
- Implementation designed for each hardware platform
- Customization done at design time
- Current research targets compilers to support *design time* paradigm

Cray XD1
www.cray.com/products/xd1

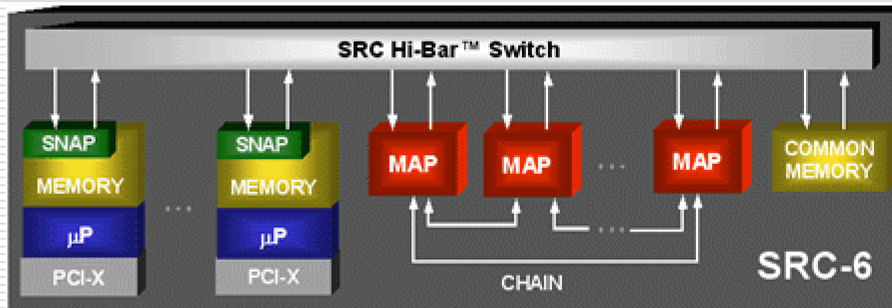


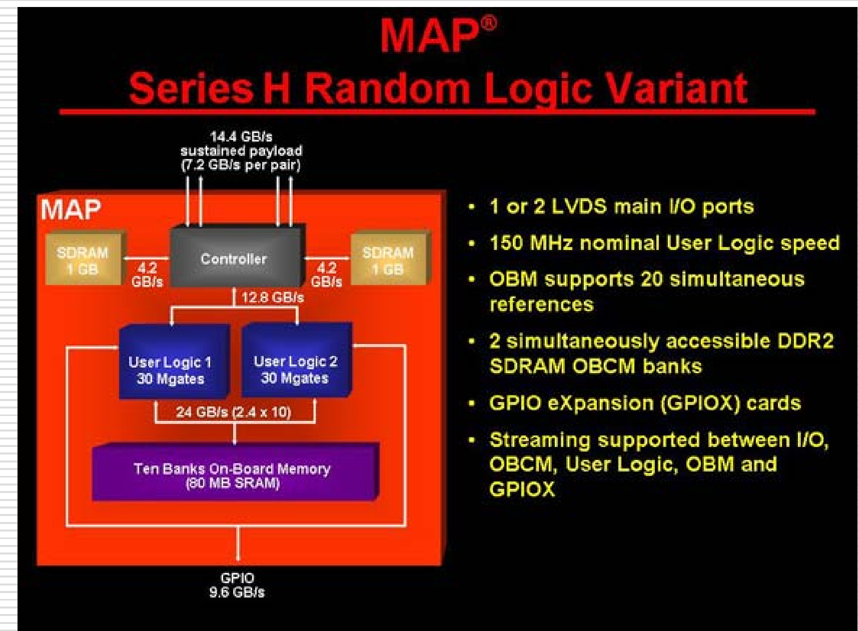
- Six nodes per chassis
  - Connected via RapidArray
- Each node:
  - Two 2.2GHz dual-core Opteron CPUs + One Xilinx Virtex4
    - Dedicated per-processor memory
      - Up to 8GB per Xeon + 16MB per FPGA
  - Two RapidArray interconnect processors
    - 3.2GB/s between same-board processors
    - 2GB/s node-to-node (per RA)

SGI RASC: http://www.sgi.com/products/rasc/

- SGI® RASC™ RC100 Blade
  - Dual Virtex 4 LX200 FPGAs
  - 80MB QDR SRAM or 20GB DDR2 SDRAM
  - Dual NUMAlink™ ports

SRC: http://www.srccomputers.com/





Mercury MCJ6 FCN Module

http://www.mc.com/products/boards.cfm

- FPGA board:
  - Two Xilinx Virtex II Pros
  - 8MB SRAM, 128MB DRAM per FPGA
  - 10 2.5GB/s links between FPGAs
  - 2 RACE++ ports per card, plus fiber, serial, and LVDS ports
- In 6U VME rack:
  - PowerPC G4+ compute nodes
  - Up to 512 GFLOPS in a chassis
  - RACE++ switch fabric
  - Aggregate bandwidth: 4.8 GB/s

HHPC at AFRL, Rome NY
http://www.rl.af.mil/tech/facilities/HPC/

- HPTi 48 Node Beowulf Cluster
  - Myrinet MPI interconnect
  - Gigabit Ethernet
- Each node
  - Dual 2.2 GHz Xeon processors
  - 4 Gigabytes of RAM
  - Annapolis Wildstar II FPGA boa
    - 2 x Virtex II 6000
    - 12 Megabytes of SRAM
    - 128 Megabytes of DRAM
    - 64b/66MHz PCI to host PC

Different Platforms: Similar Elements

- A variety of reconfigurable supercomputers
- All have:
  - Processors
  - Reconfigurable Logic
  - Memory
  - High Speed Interconnect
- Current state of the art:
  - Different design for each application and each platform

Support for all these architectures

- VSIPL++ library routines to specify functionality
- FPGA implementations of selected components
- An API for using the FPGA implementations from within VSIPL++
- We extend the model to other types of hardware: Special Purpose Processors (SPPs)

Our Goals

- Add support for special purpose processors (SPPs) to VSIPL++
- Leave the programming interface unchanged
- Maximize run-time hardware/software performance
- Facilitate new algorithm development
- Ease SPP integration
- VSIPL++ programmer should be able to write code that exploits SPP hardware without knowledge of the accelerator hardware

- *Note: We are NOT automatically compiling code to SPPs*

Why use VSIPL++ ?

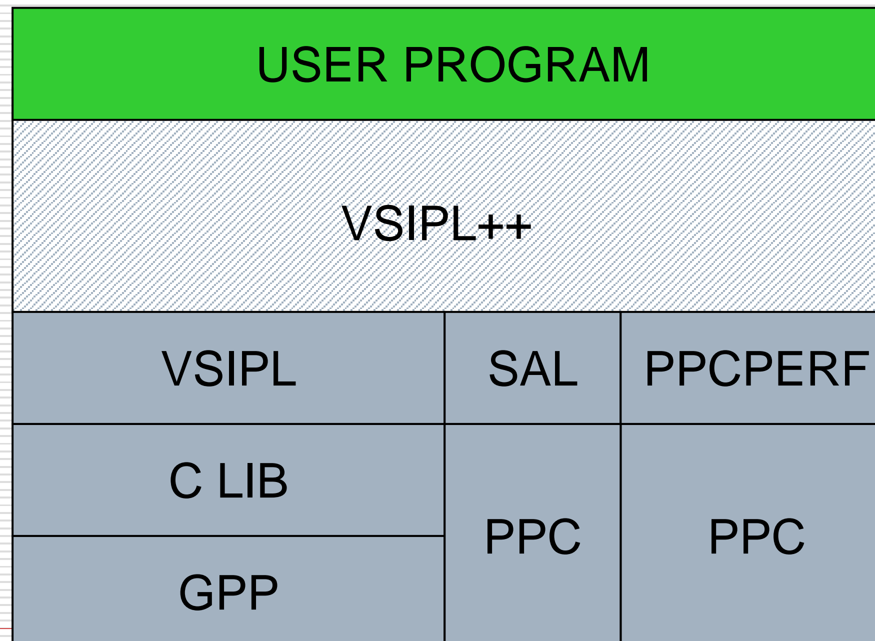
- An open API standard: http://www.hpec-si.org/ from the High Performance Embedded Computing Software Initiative
- Object oriented interface to a library of common signal processing functions
- Data objects interact intuitively with processing objects
- High level interface eases development, facilitates portability
- Run-time performance is left up to the implementation of the library
  - Optimized implementations for specific platforms

Software support for VSIPL++

- Reference implementation on top of C library

  or

- Optimized implementation for specific platform



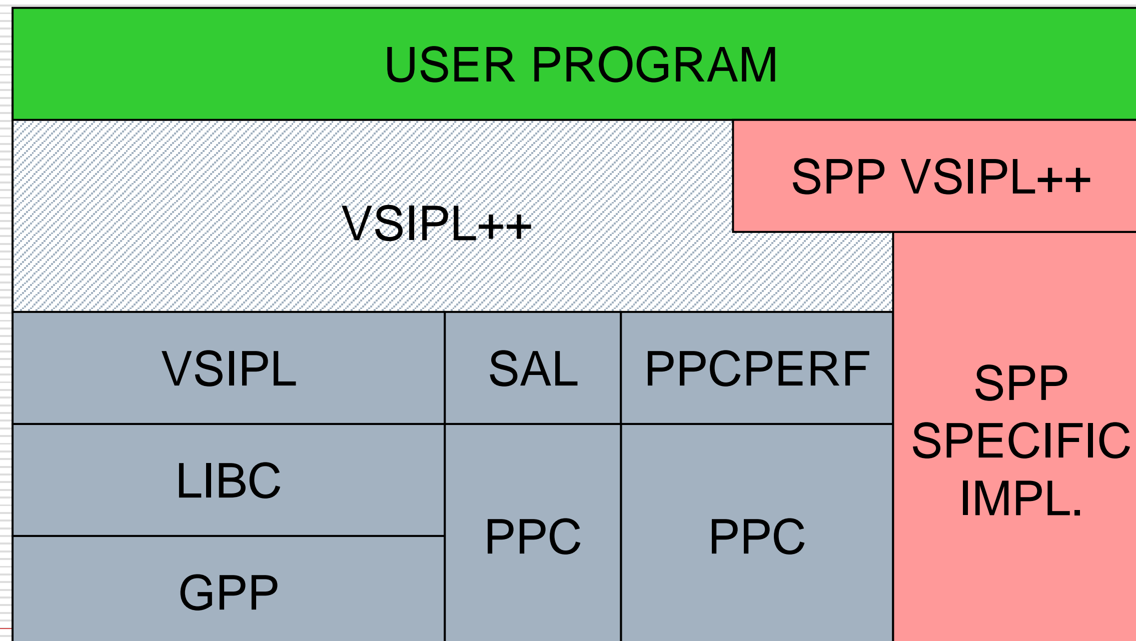
Northeastern UNIVERSITY

VSIPL++ Example: FFT

```
#include <vsip/signal.hpp>

int main(int argc, char* argv[])
{vsip::vsipl lib;
  Vector<cscalar_f> inData(16);
  Vector<cscalar_f> outData(16);
  Fft<const_Vector, cscalar_f, cscalar_f, fft_fwd>
    fft_obj(Domain<1>(16), 1.0);

  outData = fft_obj(inData);

  return(0);
}
```

Integrating SPPs into VSIPL++

- Software framework for VSIPL++ SPP support:
  - overload (intercept) a subset of VSIPL++ functions
  - add new (higher level) functions



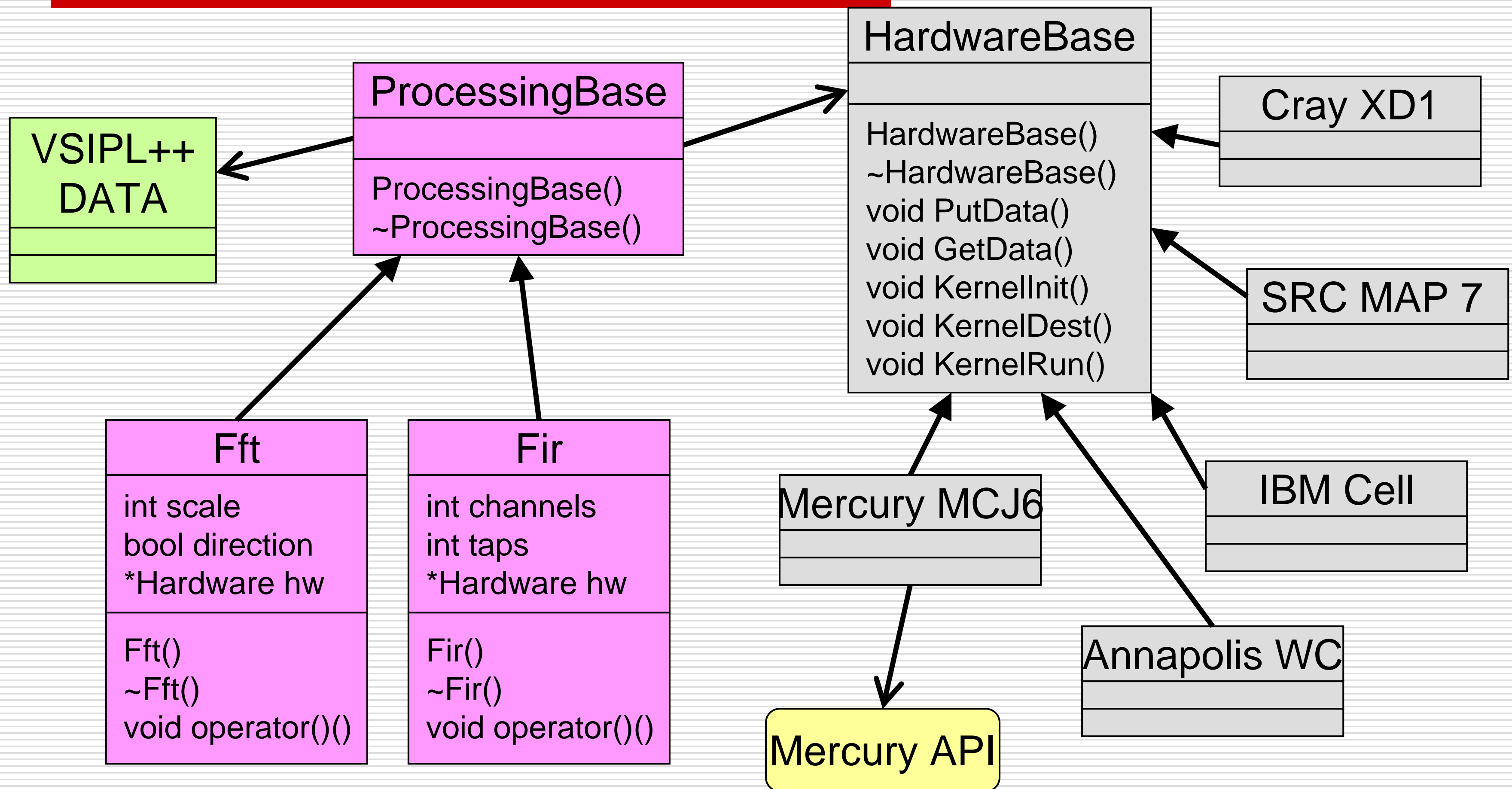
Northeastern UNIVERSITY

VSIPL++ Software Architecture

- VSIPL++ programs have
  - data objects and processing objects
  - instantiated in a user's program
- We add hardware objects to support SPPs

- SPP VSIPL++ uses VSIPL++ data objects
- Each function to be implemented with an SPP must have a drop in-replacement for the VSIPL++ processing object
  - Maintains VSIPL++ interface
- Hardware objects serve as middleware between VSIPL++ processing objects and the vendor API
- Processing objects access SPPs via abstract hardware object interface

Design for Modularity



Northeastern UNIVERSITY

Adding new hardware to SPP VSIPL++

- Need to provide:
  - Support for the hardware (hardware object)
  - Processing kernels (bitstreams) to run on the hardware
- Generate a new hardware class
    - Inherits from the abstract hardware base class
    - Implements abstract hardware base class virtual methods
    - Can add its own methods
  - Hardware objects are accessed by processing objects

Adding object code to library for SPP VSIPL++

- New functions or implementations
  - Generate a new processing class
    - Inherits from the processing base class
    - Implements processing base class virtual methods
  - Supply library with the means to execute the algorithm on the special purpose processor
    - One-to-many mapping of processing objects to SPP executables

SPP VSIPL++ Framework V1

- Version 1 supports a simple Processor - FPGA coprocessor model
- Processor responsibilities:
  - Manage the FPGA bitstream library
  - Master memory transactions
  - Program the FPGA
  - Poll FPGA for status and control

    ... All while running the VSIPL++ user program
- User passes a reference to the hardware object for the processing object to map to the FPGA

SPP VSIPL++ V1 Example

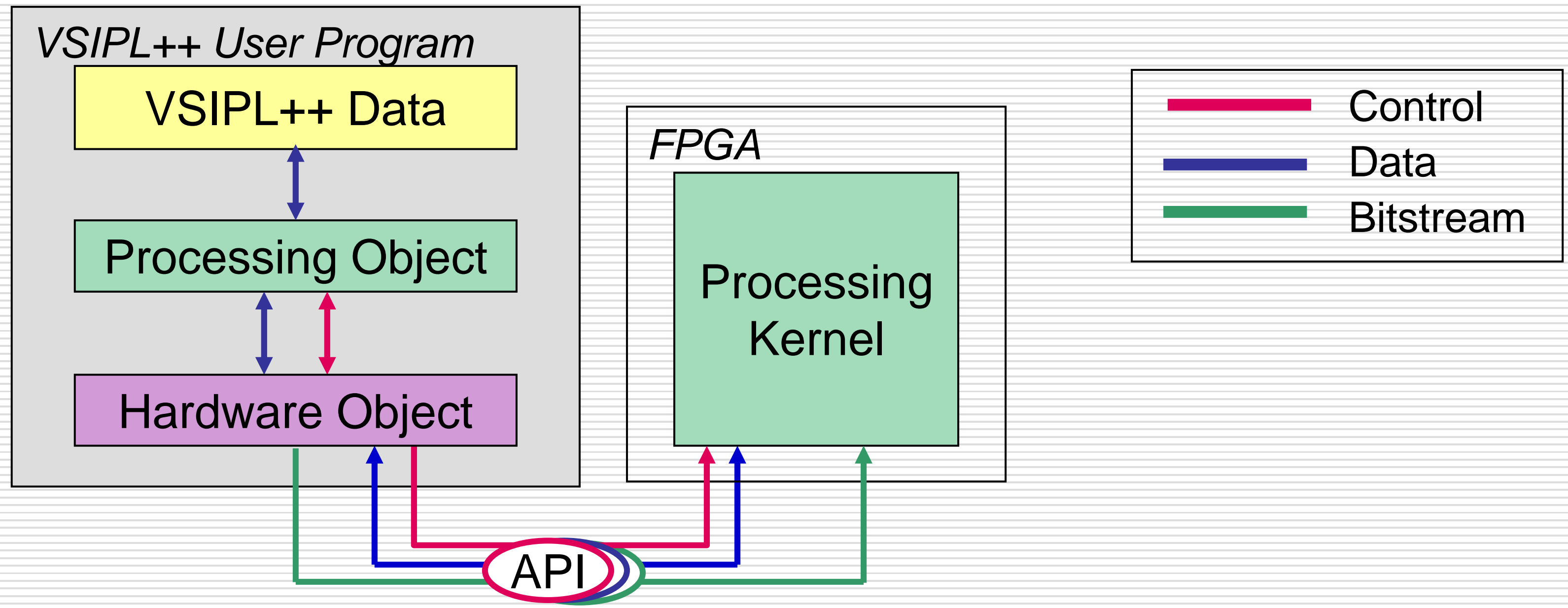
```
#include <vsip/signal.hpp>
#include <Atlanta.hpp>
#include <hw_fft.hpp>
int main(int argc, char* argv[])
{vsip::vsipl lib;
  Vector<cscalar_f> inData(16);
  Vector<cscalar_f> outData(16);

  Atlanta *fpgaBoard = new Atlanta;

  fft_obj = new Fft<const_Vector, cscalar_f, cscalar_f, fft_fwd>
    (Domain<1>(16), 1.0, fpgaBoard);

  outData = (*fft_obj)(inData);
  delete fft_obj;
  delete fpgaBoard;
  return(0);
}
```

Status of SPP VSIPL++ V1

- Hardware supported:
  - Annapolis Wildcard II
  - Mercury Computers Atlanta board

- FFT demo implemented on both platforms
  - Fixed point Xilinx core
  - Northeastern University Floating Point Library component for fix2float conversion

Control, Data Flow with V1



Northeastern UNIVERSITY

V1 Features

- Concurrency with single thread
  - Non-blocking methods within processing objects allow GPP to work asynchronously with SPP
- Exception handling
  - Automatic default to VSIPL++ software routines on errors
  - Invisible to programmer
  - Throws exceptions when VSIPL++ would

SPP VSIPL++ Framework V2

- Supports parallel heterogeneous systems:
    - Multiple GPPs, Multiple FPGAs
- Abstracts away differences in hardware platforms at the user program level
- Incorporates a run-time resource manager (RTRM)
- GPP responsibilities:
    - Running VSIPL++ user program
    - Requesting, releasing resources from/to RTRM
    - Interfacing with allotted resources
- RTRM responsibilities:
    - Maintain a list of SPPs and status of each
    - Handle requests for resources
    - Configure SPPs with processing kernels
- Add control layer without performance loss
- *Version 2 is under development*

SPP VSIPL++ V2 Example

```
#include <vsip/signal.hpp>
#include <spp.hpp>
#include <hw_fft.hpp>
int main(int argc, char* argv[])
{vsip::vsipl lib;
  Vector<cscalar_f> inData(16);
  Vector<cscalar_f> outData(16);
  // Ask RTRM to consider allocating a SPP for an FFT
  Spp *hardware = new Spp(Fft);
  // Map algorithm to SPP or run as RTRM dictates
  fft_obj = new Fft<const_Vector, cscalar_f, cscalar_f,
    fft_fwd>
     (Domain<1>(16), 1.0, hardware);
  outData = (*fft_obj)(inData);
  delete fft_obj;
  delete hardware;
  return(0);          }
```

Run-Time Resource Manager

- RTRM encapsulates platform specific information
  - runs on a general purpose processor
  - is a standalone application
  - Can be implemented in any language supported by GPP and the SPP-specific API (C, C++, VSIPL++, ...)
  - Communicates with hardware objects (VSIPL++) via InterProcess Communications (IPC)
  - Configures SPPs and checks their status via SPP-specific API
- RTRM spawn required before VSIPL++ programs make resource requests
  - Should VSIPL++ spawn RTRM or vice versa?

RTRM: Advanced Features

- RTRM enables multiple VSIPL++ programs to run in parallel and share resources on the same system
- Many choices left to the implementation
  - Scheduling algorithm
  - Pre-programmed knowledge of the architecture
    vs.
  - Discover available resources at run-time
  - Support for fault tolerance
  - Load balancing
- Anonymous brokering of services
  - User programs application tasks
  - RTRM handles the rest

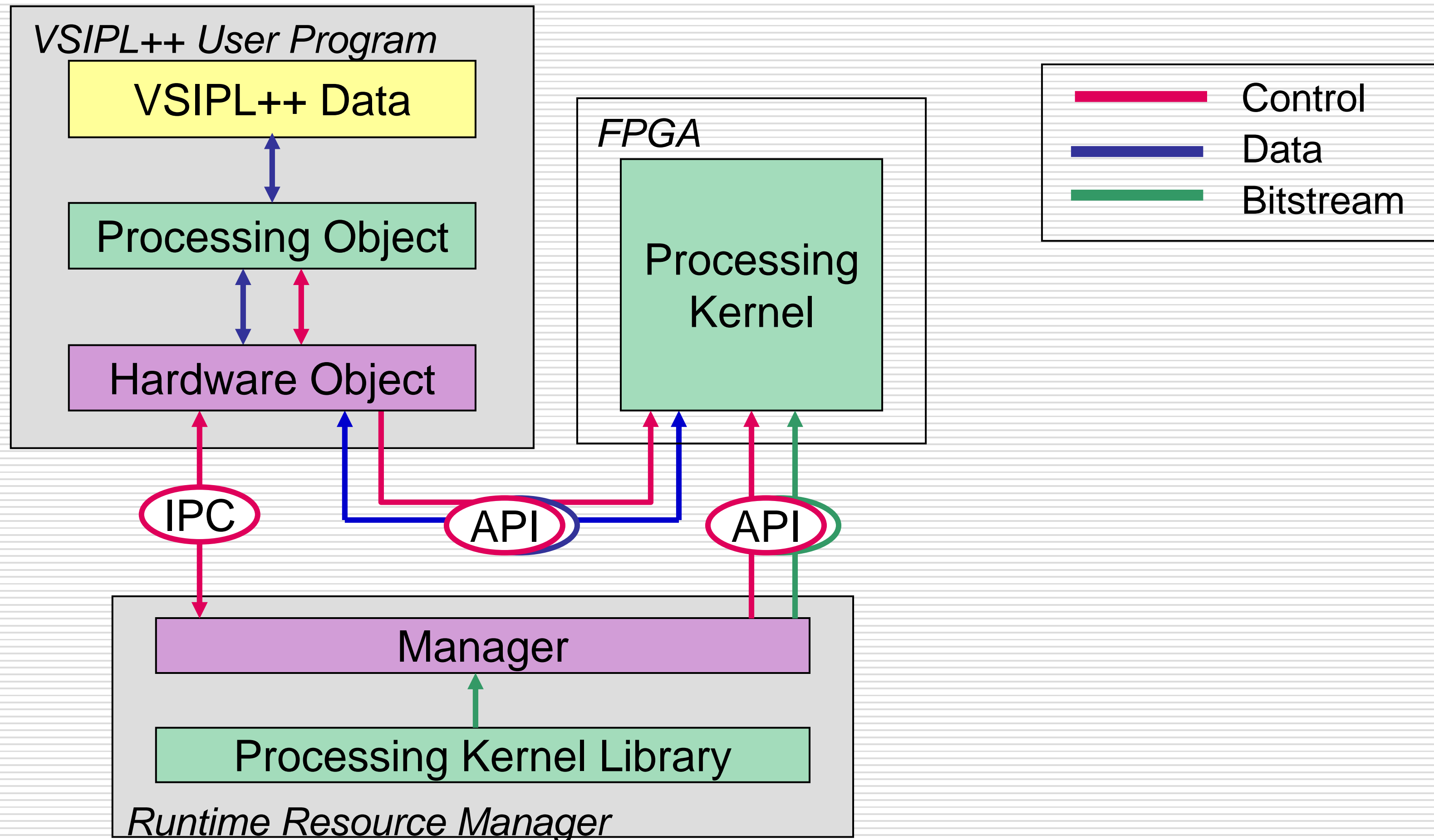
V2 Current Status

- Initial version of RTRM implemented for Mercury MCJ6
  - First come, first served scheduler
  - Knowledge of Mercury FCN SPP and GPP only
  - Two requests from board objects handled:
    - Initialize Kernel
    - Destroy Kernel
  - Run-time support for malfunctioning SPPs
    - Default to software
- Additional features are currently under development
  - Platform abstraction has yet to be implemented
  - Hardware instantiation and mapping is done at the user level
    - Syntax same as for Version 1
    - Goal: eliminate calls to specific hardware from VSIPL++ program (in Version 3)
- Manager for CRAY XD-1 is under development

V2 Example (Current)

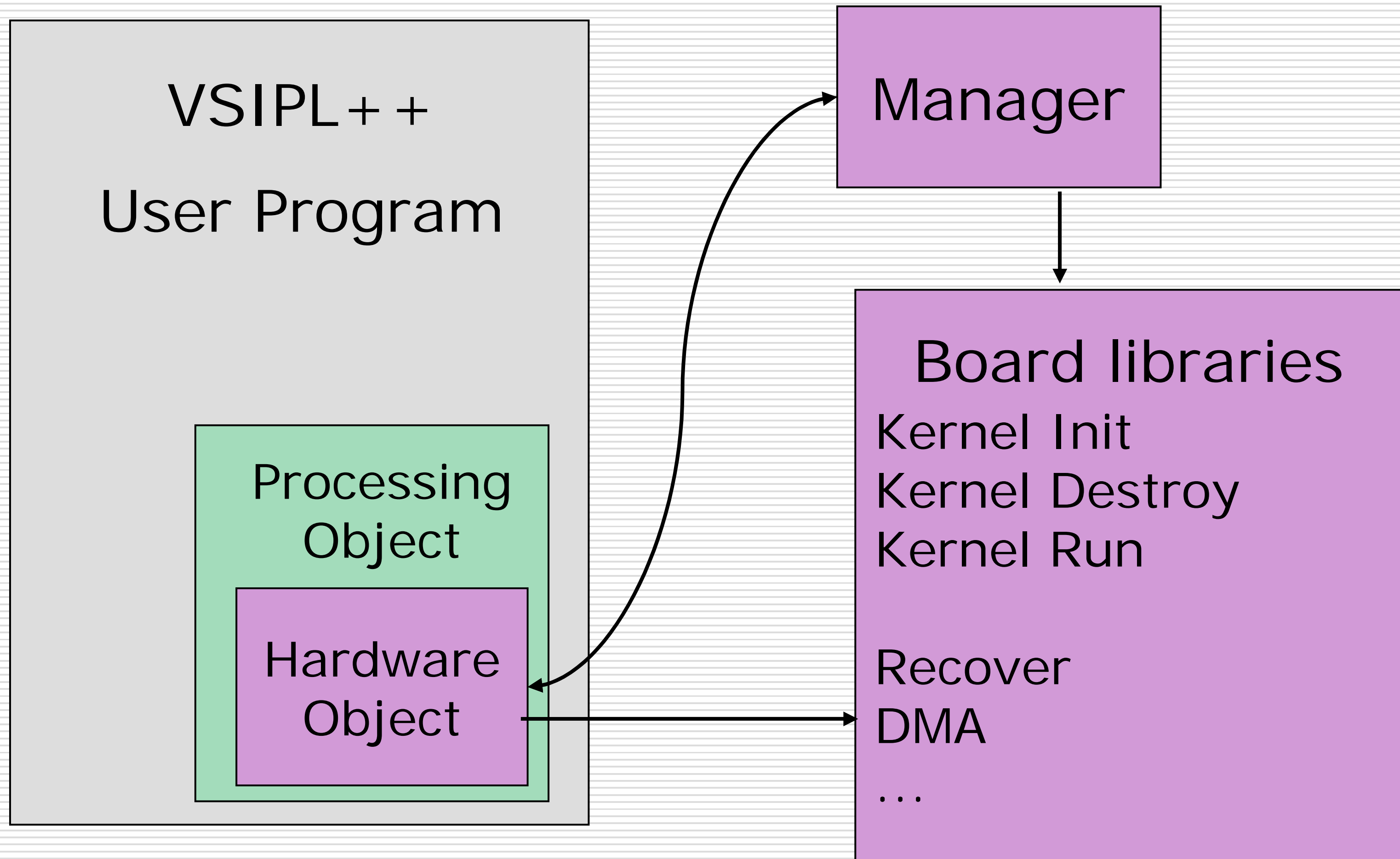
```
#include <vsip/signal.hpp>
#include <boston.hpp>
#include <hw_fft.hpp>
int main(int argc, char* argv[])
{ vsip::vsipl lib;
  Vector<cscalar_f> inData(16);
  Vector<cscalar_f> outData(16);
  Boston *hardware = new Boston;

  fft_obj = new Fft<const_Vector, cscalar_f, cscalar_f,
    fft_fwd>
    (Domain<1>(16), 1.0, hardware);
  outData = (*fft_obj)(inData);

  delete fft_obj;
  delete hardware;
  return(0);     }
```

Control, Data Flow with V2



Northeastern UNIVERSITY

V3 (under development)



Future Work

- More managers for more architectures
  - Cray XD1
  - SGI RASC
  - SRC
- More complicated applications
  - Beamforming
  - Pulse compression
- Support to break Master-Slave model of processing/communication
  - Currently communication is all between FPGA and GPP
  - Support FPGA to FPGA communication
- Look at new platforms
  - IBM Cell Processor, GPUs, …

Future: Run-time Design Tools

- Design once
  - Optimize for common processing elements
    - Compilers, design libraries
- Customize design for each platform at run-time
  - Automatically determine available hardware
  - Support for fault tolerance
  - Dynamic load balancing
- Challenge:
  - Provide high performance without writing application for specific platform
  - Write once, run anywhere … efficiently

Thank you

- Miriam Leeser: mel@coe.neu.edu

- Reconfigurable Computing Lab:

http://www.ece.neu.edu/groups/rcl/

- VSIPL++ project:

http://www.ece.neu.edu/groups/rcl/projects/vsipl/vsipl.html