*VFORCE: **V**SIPL++ **for** **R**econfigurable **C**omputing **E**nvironments*

Miriam Leeser

Nicholas Moore, Albert Conti

Department of Electrical and Computer Engineering
Northeastern University, Boston MA

Laurie Smith King
Department of Mathematics and Computer Science
College of the Holy Cross, Worcester MA

Outline

- COTS Heterogeneous systems:
  Processors + FPGAs
- Overview of VSIPL++
- VForce framework
- Run-time resource management
- Current status:
  - hardware platforms, applications
- Future directions

Reconfigurable Supercomputing

Cray XD1

www.cray.com/products/xd1/

- Xilinx Virtex 2s paired with AMD Opteron nodes
- RapidArray interconnect

SGI RASC

www.sgi.com/products/rasc/

- Xilinx Virtex 2 blades for server acceleration
- Numalink interconnect system
- FPGAs have access to globally shared system memory

Heterogeneous Embedded Systems

Mercury PowerStream

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

- Interchangeable PPC and FPGA daughtercards housed in chassis
- Race++, RapidIO interconnect

SRC SRC-7

www.srccomp.com/MAPstations.htm

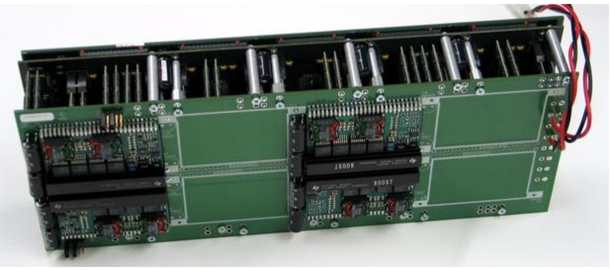
- Different sized systems with MAP components (FPGAs), processors, memory
- Custom configurations
- Proprietary Hi-Bar network

Heterogneous Embedded System and Supercomputer

Starbridge Hypercomputer

www.starbridgesystems.com

- Array of FPGAs coupled to a single microprocessor via PCI-X
- Different, fixed configurations:
  - 7, 11 FPGAs



HPTi cluster at AFRL

www.if.afrl.af.mil/tech/facilities/HPC/hpcf.html

COTS technology achieves high peak flops

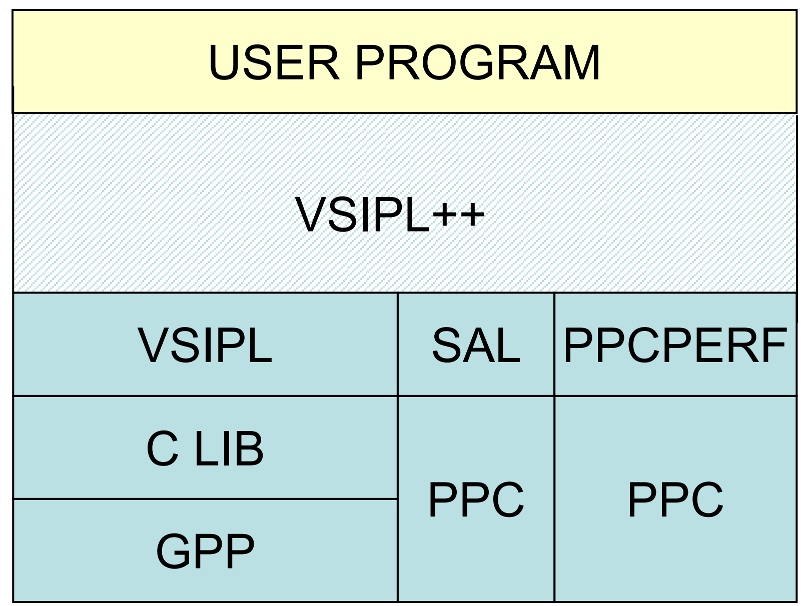
- Nodes combine dual Xeon Linux boards with Annapolis FPGA boards
- Myrinet interconnect, PCI to FPGA

Portability for Heterogeneous Processing

- All systems contain common elements
  - Microprocessors
  - Distributed memory
  - Special-purpose computing resources
    - FPGAs are our focus
    - also GPUs, DSPs, ...
  - Communication channels
- Currently no support for application portability across different platforms
- Redesign required for hardware upgrades, move to new architecture
- Focus on commonalities, abstract away differences
- Deliver performance

What is VSIPL++ ?

- An open API standard from HPEC-SI
- A library of common signal processing functions
  - Data objects interact intuitively with processing objects
  - High level interfaces ease development
- Implementation can be optimized for a given platform
  - run-time performance depends on implementation
  - Different implementations of VSIPL++ are available from different vendors



VSIPL++ Example: 16 point FFT

```
#include <vsip/signal.hpp>
using namespace vsip;

int main(int argc, char* argv[])
{
  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);
}
```

Why VSIPL++ for Reconfigurable Systems?

- Focus of VSIPL++ is
  - high performance
  - code portability
  - end-user productivity
- Parallel VSIPL++ specification support
  - mapping applications across distributed computing elements
- Object-oriented interface makes class replacements straightforward
- Functions available through VSIPL++ include proven candidates for reconfigurable hardware acceleration

VForce: Extending VSIPL++

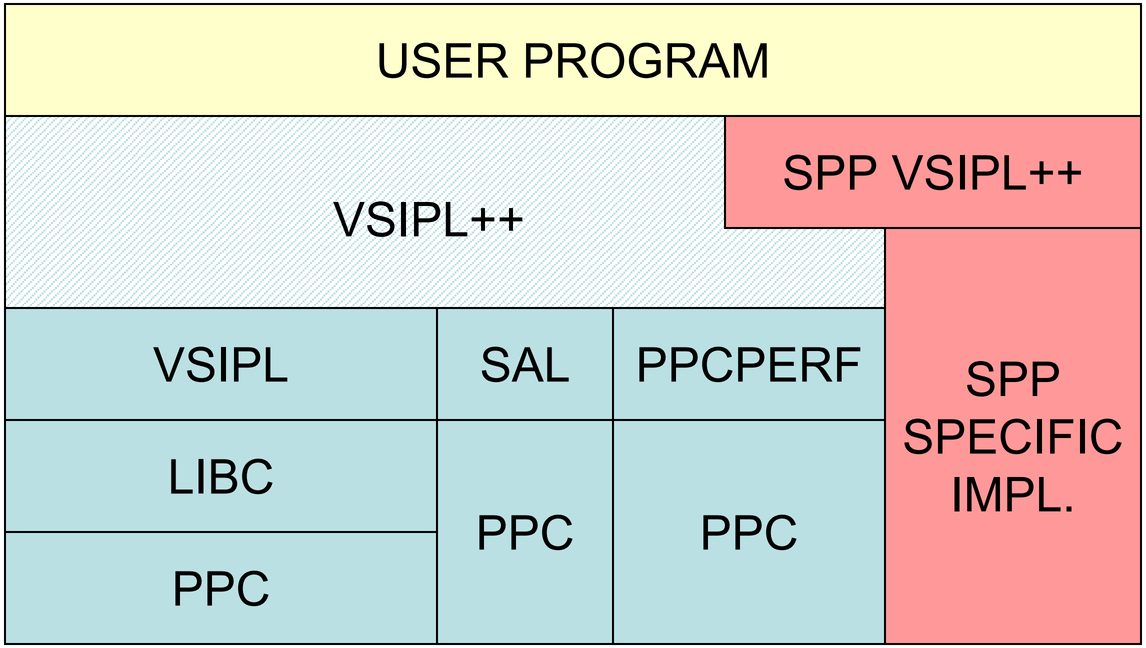
- VForce: a middleware framework
- adds support for special purpose processors (SPP) to VSIPL++
- Programmer uses VSIPL++ processing and data objects
  - Custom processing objects utilize a generic SPP object that interacts with VForce
  - SPP object uses SPP implementations when available (defaults to software)
- Standard API between processing objects and hardware resources

VForce Project Goals

- Existing VSIPL++ programs require little to no modification to use special purpose hardware
- Adding support for new hardware platforms is straightforward
- Hardware and software execute concurrently
- Hardware specific errors are hidden from user
- Platforms with multiple processors and special purpose processors are supported
- The framework is flexible to adapt to new developments in the VSIPL++ specification

VForce: Extending VSIPL++

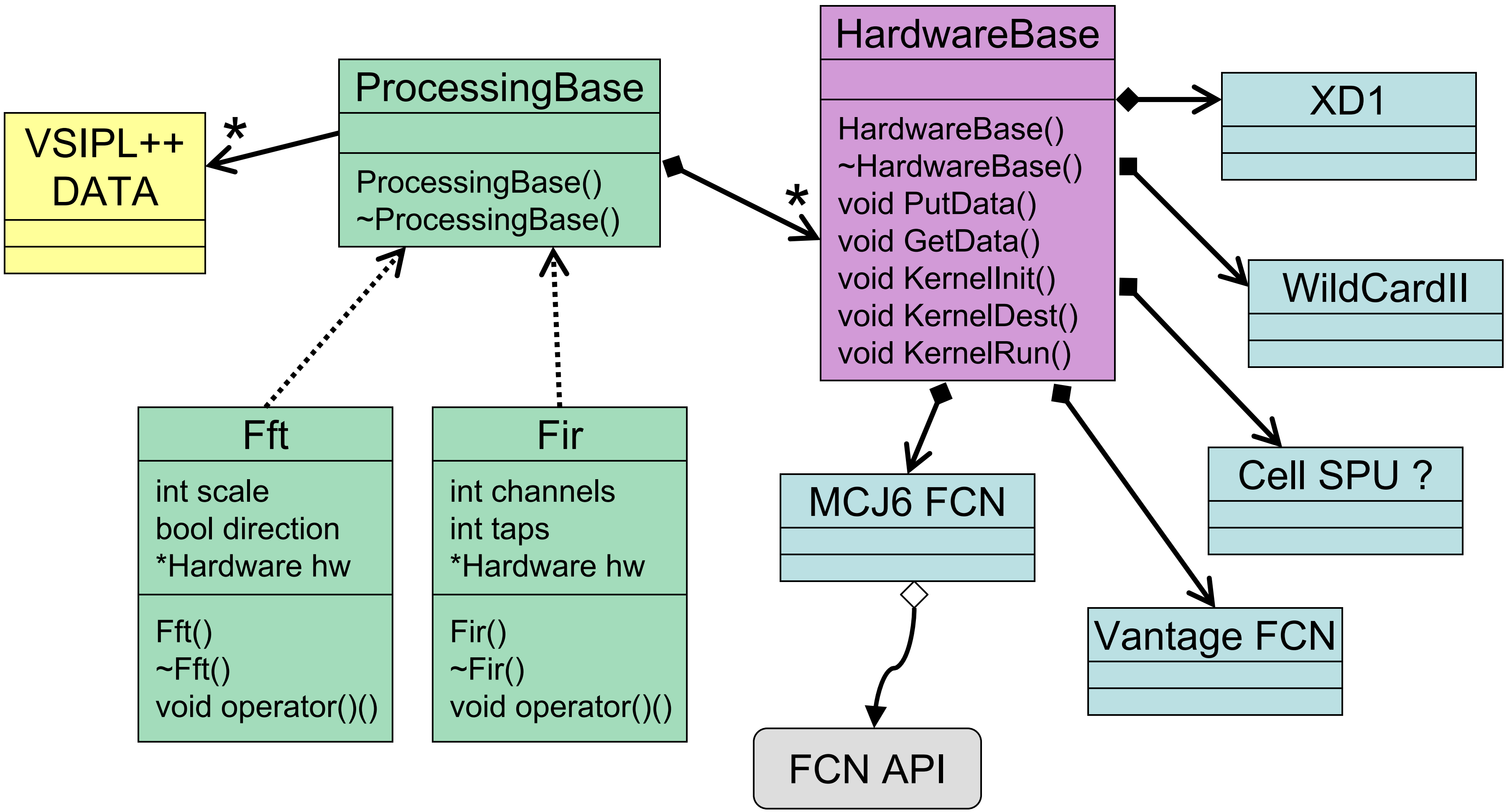
- Sits on top of VSIPL++
  - Implementation Independent
- Custom processing objects:
  - Overload a subset of VSIPL++ functions
  - Add new higher level functions → SPP's strength



VForce API

- Generic SPP object implements a standard API:
  - Move data to and from the Special Purpose Processor (SPP)
  - Configure algorithm parameters
  - Initialize and finalize SPP kernels
  - Start processing
  - Check interrupt status
- A processing object uses these hardware functions to interact with the SPP

High Level Class Diagram



Northeastern
UNIVERSITY

Dynamically Loaded Shared Objects (DLSO)

- Generic SPP objects are hardware independent
- Use dynamically loaded shared objects(DLSO) to control a specific SPP
- Each type of SPP requires a pre-compiled DLSO that converts the standard VForce API into vendor specific calls
- Separation of hardware concerns from user code and from binary until runtime
- Which DLSO and device?
  - Determined by a Run Time Resource Manager (RTRM)

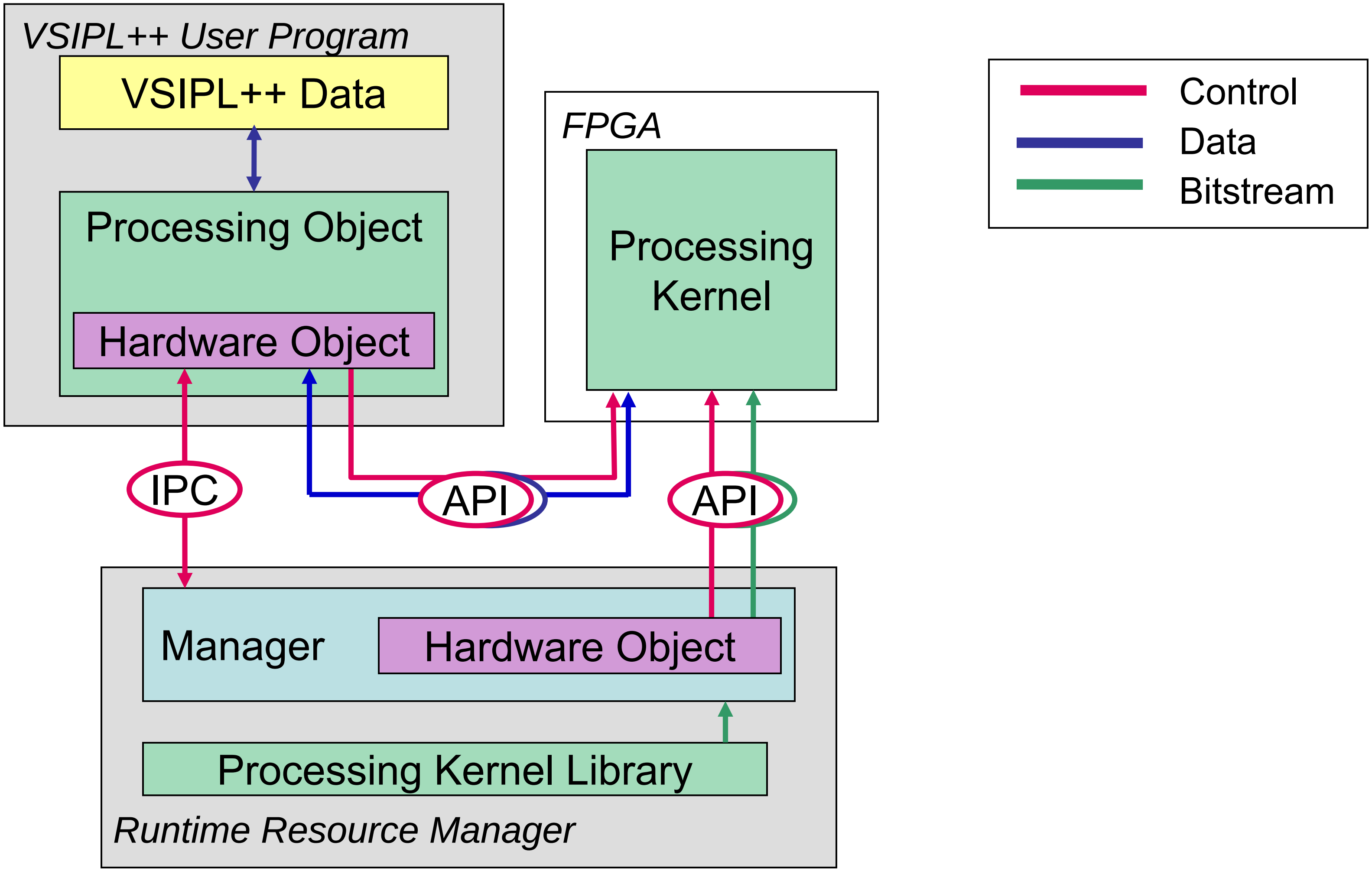
Run-time Resource Manager

- The Runtime Resource Manager (RTRM) encapsulates machine & vendor differences:
  - How many and what types of SPPs
  - What user code is running where
  - Knowledge of available bitstreams, binaries
- SPP object requests a hardware resource with a specified application kernel
- Manager returns device and DLSO to use

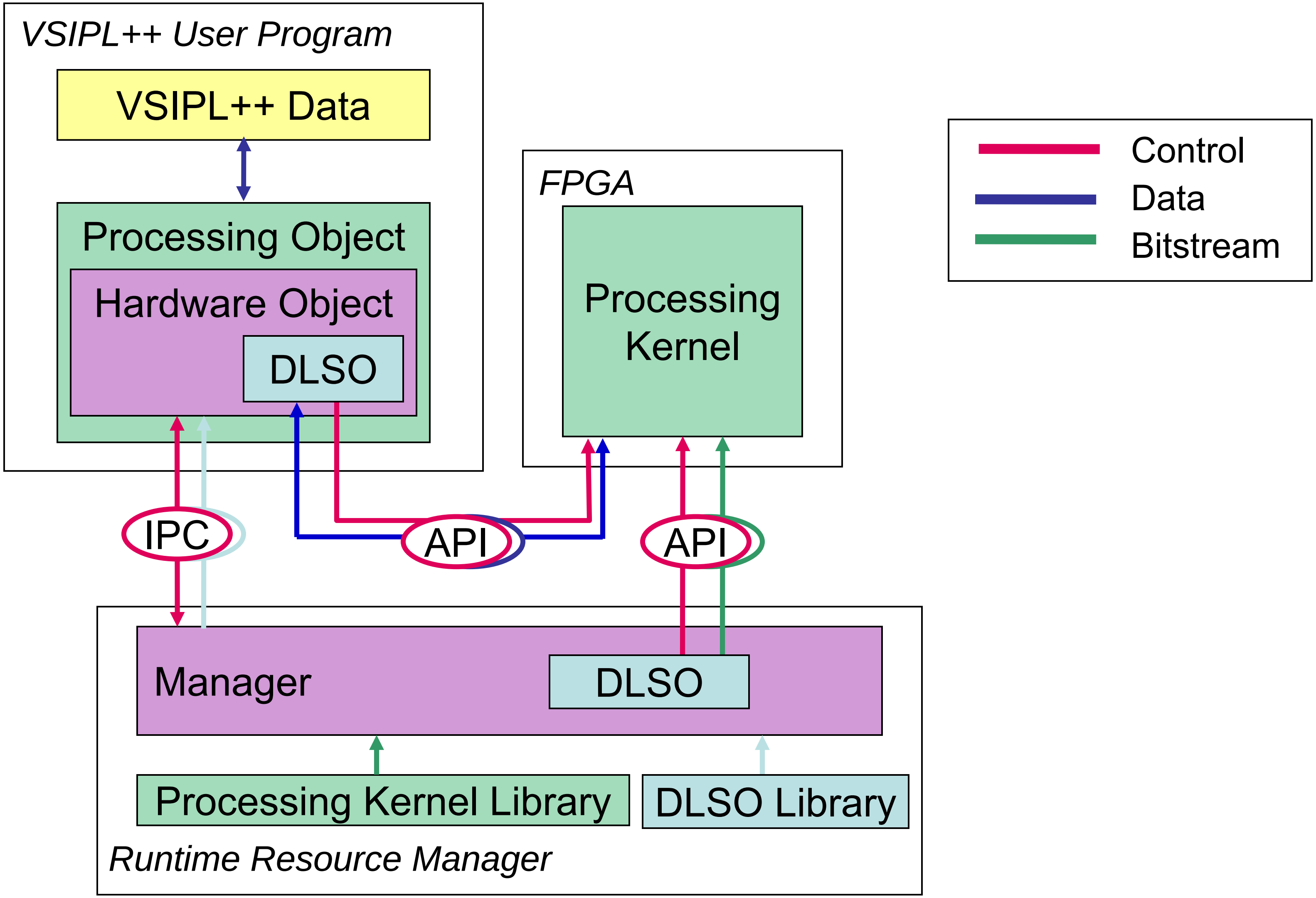
Run-time Resource Manager (2)

- The RTRM exists as a separate process
- Important: communication of data is direct from one processing element to another
  - Manager does NOT handle data
  - Manager does scheduling and allocation of tasks
  - Performance should not be impacted

Control and Data Flow



Control and Data Flow



Northeastern UNIVERSITY

VForce Framework Benefits

- VSIPL++ code easily migrates from one hardware platform to another
- Specifics of hardware platform encapsulated in the manager and DLSOs
- Handles multiple CPUs, multiple FPGAs efficiently
- Programmer need not worry about details or availability of types of processing elements
- Enables run time services:
  - fault-tolerance
  - load balancing
  - ...

VSIPL++ SPP Example: 16 point FFT

```
#include <hw_fft.hpp>
using namespace vsip;

int main(int argc, char* argv[])
{
   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);
}
```

Northeastern UNIVERSITY

Adding New Hardware to VForce

- Need to provide:
  - Hardware DLSO
  - Processing kernels (bitstreams) to run on the hardware
    - Drop in replacements for VSIPL++ functions
  - Manager must be aware of new hardware
- The new DLSO must implement the VForce standard API

Adding New Processing Objects

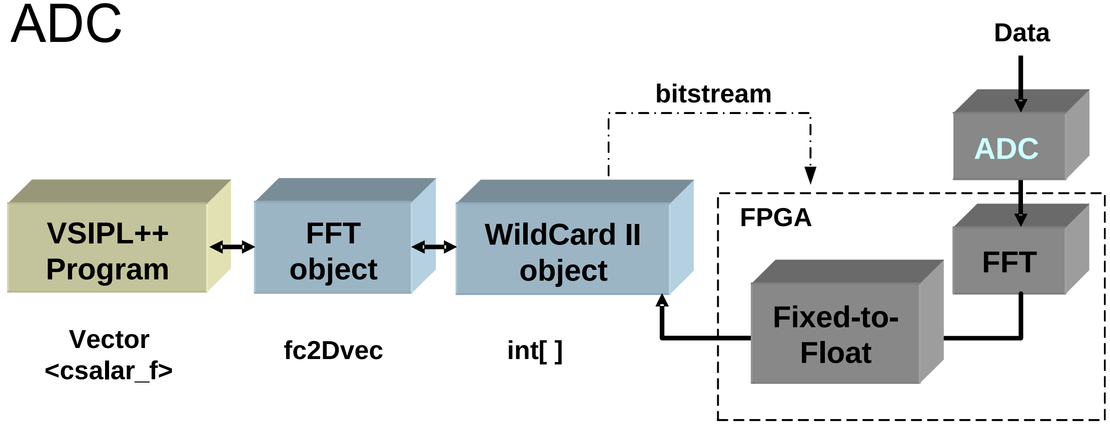
- New functions or implementations
  - Generate a new processing class
    - Use SPP object(s) to control hardware
  - Supply kernel with the means to execute the algorithm on the special purpose processor
- One-to-many mapping of processing objects to SPP executables
  - software
  - FPGA hardware: may be several versions
  - other types of SPPs

Platforms Currently Supported

- Annapolis WildCard II, Mercury VantageFCN
  - static scheduling mechanism and local run-time resource management only
  - Asynchronous calls allow for task-level parallelism
  - Exception handling hides hardware specific errors
- Mercury 6U VME
  - Static scheduling and local RTRM
  - Dynamic scheduling with remote run-time resource manager supervision
  - Manager allows for multiple VSIPL++ programs to run on the same system and share MCJ6 FPGA compute nodes
- Cray XD1
  - Dynamic scheduling with remote run-time resource manager supervision
  - Dynamic loading of shared objects
- No changes to code, 100% portability for applications

Completed Applications: 16 Point FFT

- Small FFT
  - A 16 point FFT for proof of concept
  - Complex single precision floats converted to fixed point for computation in hardware
  - FPGA kernel implemented for Annapolis WildCard II, Mercury VantageFCN and Mercury 6U VME
  - Software ported with minimal code changes
- Small FFT with A/D converter input
  - FFT kernel modified to sample data from the WildCard II's on-board ADC



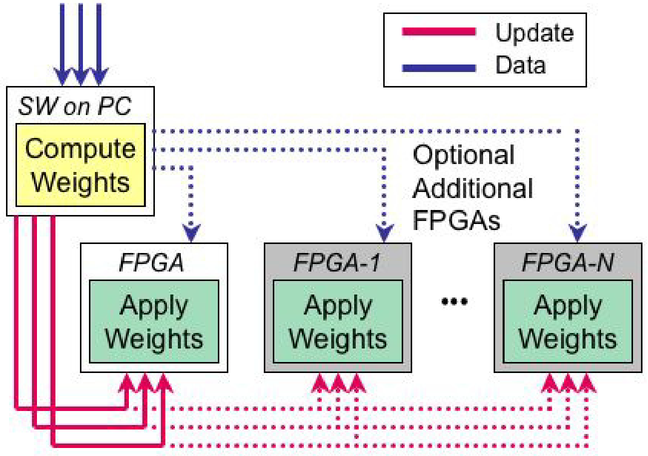
Parameterized FFT Implementation

- Uses parameterized FFT core from Xilinx Corelib
- 8 to 32k point FFT
- Matches functionality of the FFT class within VSIPL++
  - Scaling factor and FFT size adjustable after FPGA configuration
- Complex single precision floats in VSIPL++ converted to fixed point for computation in hardware
- Dynamic scaling for improved fixed point precision
- Implemented for WildCard II and Cray XD1

Beamformer

- 3-D time-domain beamformer, adaptive weights
- Single precision floating point operators at every stage
- Hybrid hardware/software implementation

- Weights computed periodically using QR decomposition
  - in software
- Multiply accumulate
  - in hardware
- Sensor data distributed to multiple FPGAs with round robin scheduling
- Implemented on Mercury 6U VME



Future Directions

- Support for new platforms
  - SGI RASC
  - SRC SRC-7
  - GPUs, DSPs, CELL SPEs
- Move beyond master-slave model of processing and communication
  - FPGA to FPGA communication not currently implemented
- Implement more complex processing kernels, applications

Conclusions

- VForce provides a framework for implementing high performance applications on heterogeneous processors
  - Code is portable
  - Support for a wide variety of hardware platforms
  - VSIPL++ Programmer can take advantage of new architectures without changing application programs

Thank you

- Contact: mel@coe.neu.edu
- VForce:

  http://www.ece.neu.edu/groups/rcl/project/vsipl/vsipl.html
- Reconfigurable Computing Lab @ NU

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

Northeastern UNIVERSITY