Extending VForce to Include Support for NVIDIA GPUs using CUDA

Dennis Cuccaro, Nicholas Moore, Miriam Leeser
Department of Electrical and Computer Engineering
Northeastern University, Boston, MA

Laurie Smith King
Department of Math & Computer Science
College of the Holy Cross, Worcester, MA

Outline

- VForce Review
  - What is VForce?
  - Past Applications & Platforms
- Extending VForce to GPUs
  - Support for Nvidia CUDA
  - FFT Demonstration Application
- Future Work

Motivation 1

- A lot of new architectures
  - Many use “non-traditional” processor accelerators attached as co-processors
    - FPGAs, GPUs, and Cell SPEs
- For certain applications these accelerators offer a lot of potential performance improvements
  - Fine grained parallelism within accelerator
  - Coarser grained parallelism between processing elements

Motivation 2

- Drawbacks for adopting new architectures:
  - New architectures hard to use
    - Require specialized hardware knowledge
    - Vendor specific toolchains
  - Code is not portable
    - Vendor specific code mixed with application code
  - Short hardware shelf life
- Want tools to help deal with these challenges
  - Maintain performance
  - Reuse algorithm kernels
  - Maintain productivity

VSIPL++

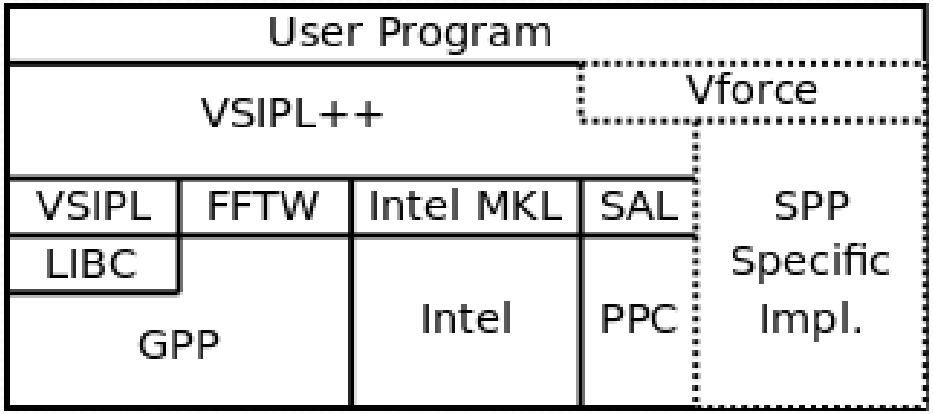
- C++ version of the Vector Signal Image Processing Library
- Open-standard API specification produced by High Performance Embedded Computing Software Initiative (HPEC-SI, www.hpec-si.org)
- Provides an object oriented interface to a library of common signal processing functions
  - Data classes specify storage, access, and distribution
  - Processing classes operate on data classes
- Particular implementation is responsible for performance on a given platform

VForce Overview 1

- VForce (**V**SIPL++ **for** **R**econfigurable **C**omputing **E**nvironments) is middleware for mapping VSIPL++ functions to special purpose processors (SPPs)
- Maintains VSIPL++ environment
  - Application programmer does not deal with accelerators
- Maintains VSIPL++ portability
  - No hardware specific code in compiled application
  - Applications do not need accelerators to run
  - Built on top of VSIPL++ API – implementation independent
- Compile Time and Runtime Components
  - Runtime binding to hardware
- Library based: use preexisting SPP kernels

VForce Overview 2

- Create new “processing objects” for acceleration
  - Function offload a decent match for accelerators
    - Granularity issues



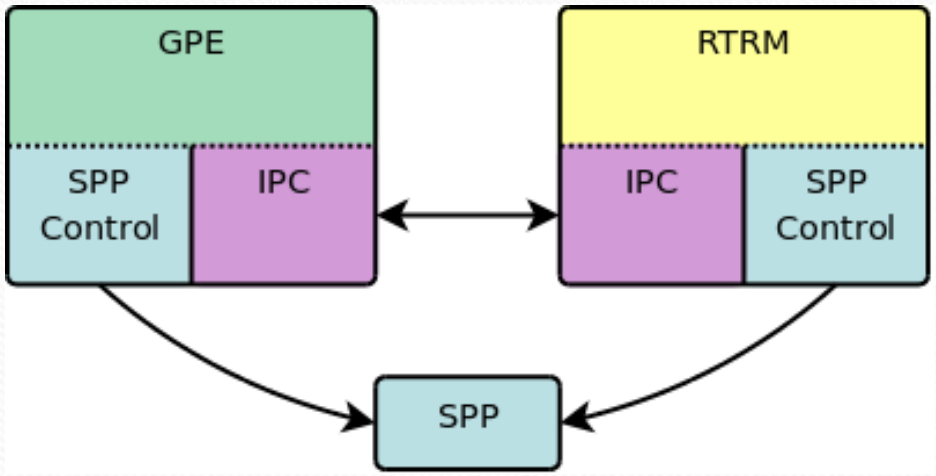
- Each processing object needs two implementations
  - Accelerated version
  - Software-only failsafe
- The accelerated version uses the generic processing element (GPE) to control
- Whenever there are no accelerators or an error default to software – no user programmer interaction

Generic Processing Element

- The Generic Processing Element (GPE) exposes a generic set of accelerator operations
  - Kernel execution control
  - Data transfers
- Supports non-blocking operations
- GPE contains no accelerator specific code – loaded at runtime
- GPE uses two internal VForce interfaces
  - Request/surrender accelerator hardware
  - Accelerator control interface

VForce Framework

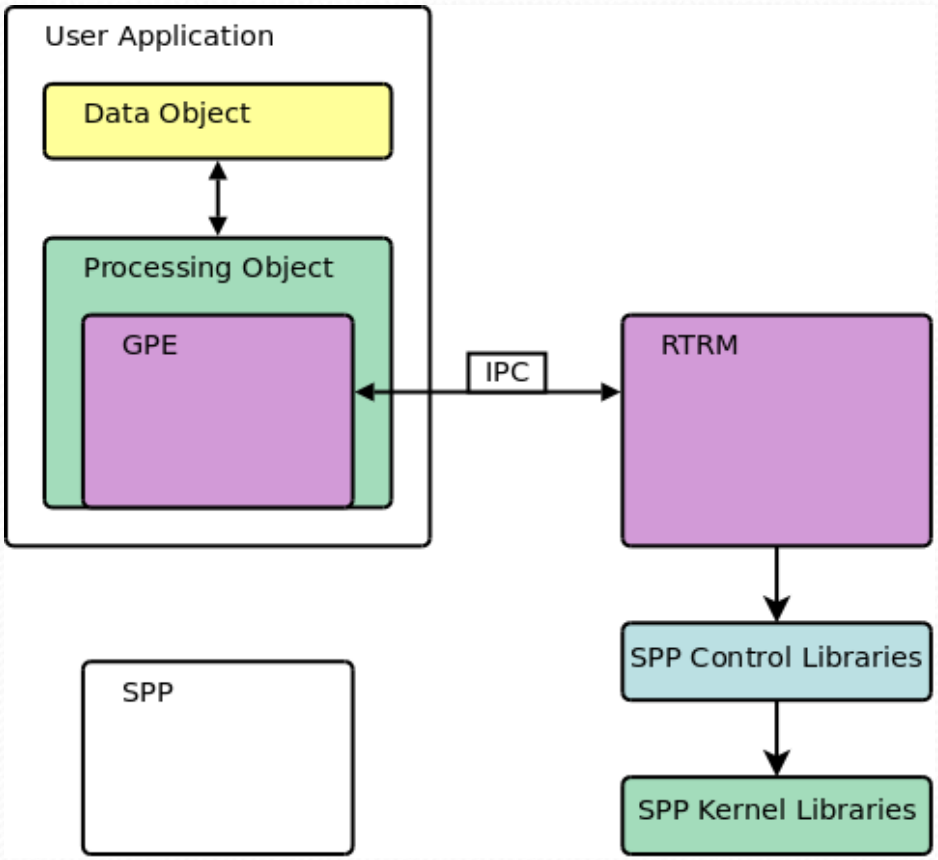
- GPE could bind to platform specific interfaces directly
- Currently gets hardware from system-wide Runtime Resource Manager (RTRM) via IPC



- RTRM manages HW and makes accelerator allocation decisions – completes abstraction
- Opportunity for runtime services – not explored
  - Current implementation is first-come, first-served
  - Generic like GPE – runtime binding

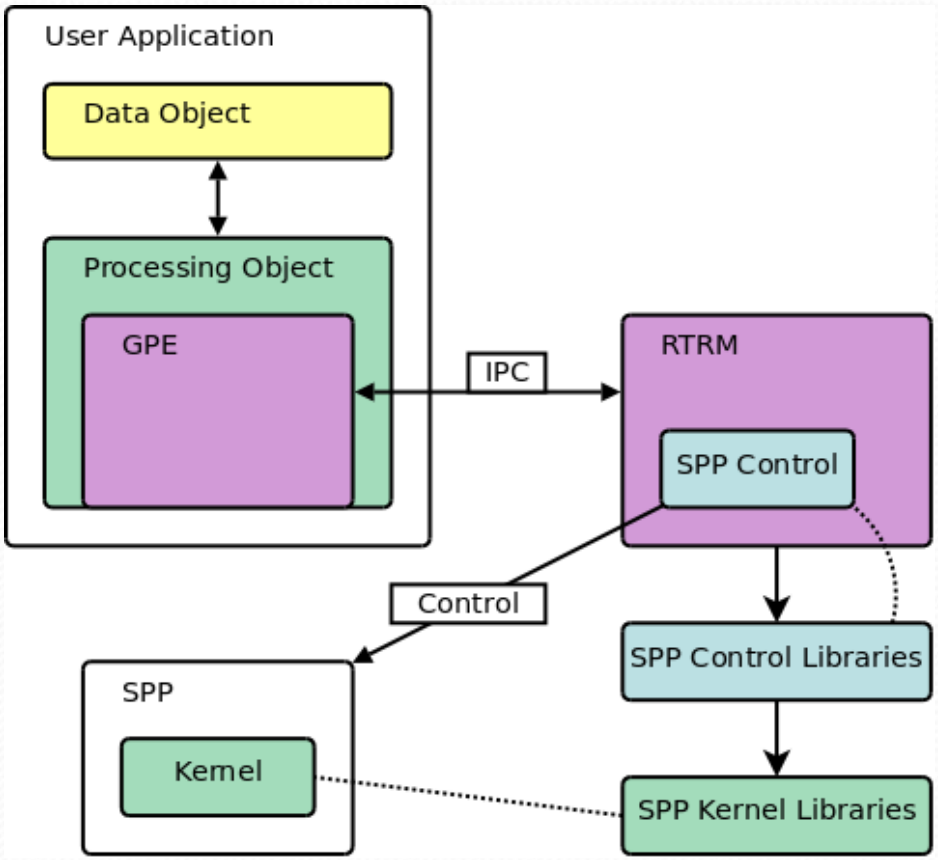
VForce Interaction 1

- During execution processing object tries to initialize a SPP
- GPE requests a SPP from RTRM via interprocess communication (IPC)
- Manager determines if there is an algorithm/SPP match
  - Optionally programs device with kernel
- Replies to GPE via IPC



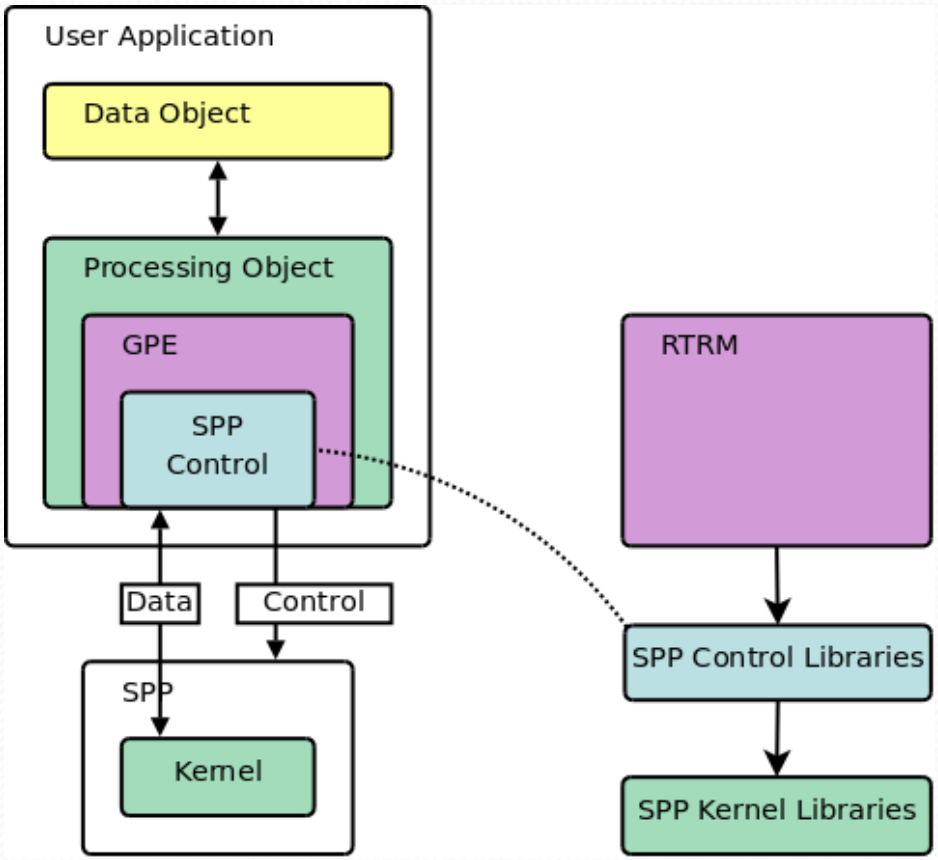
VForce Interaction 2

- During execution processing object tries to initialize a SPP
- GPE requests a SPP from RTRM via interprocess communication (IPC)
- Manager determines if there is an algorithm/SPP match
  - Optionally programs device with kernel
- Replies to GPE via IPC



VForce Interaction 3

- Hardware Available?
  - No: transfer to software implementation
  - Yes
    - Load the indicated SPP control library
    - Continue with the hardware/software implementation
- During execution communication and control direct – RTRM not involved



Previous VForce Work

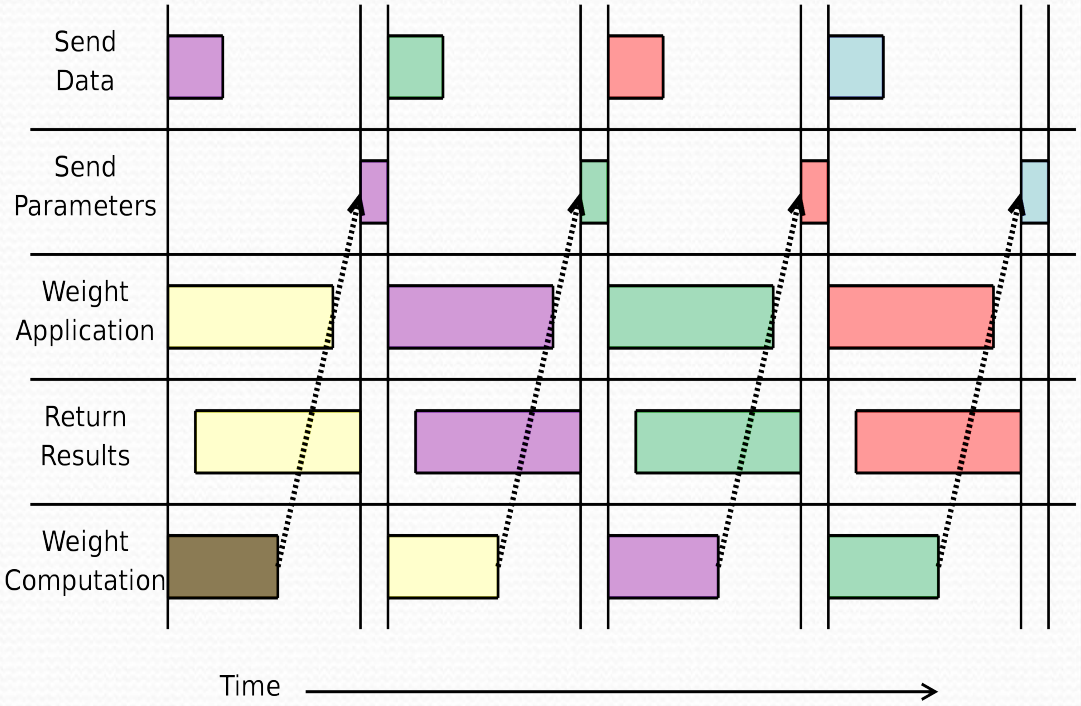
- We previously presented work on several FPGA-based platforms
  - *Vforce: Aiding the Productivity and Portability in Reconfigurable Supercomputer Applications via Runtime Hardware Binding*, HPEC 2007
  - *VFORCE: VSIPL++ for Reconfigurable Computing Environments*, HPEC 2006
- Early work on Annapolis WildCard II PCMCIA card
- Support for Cray XD1 and Mercury 6U VME systems
  - All Mercury development done by Albert Conti (NU MS 12/2006, Mitre)
  - FFT and time domain beamformer implemented for Cray and Mercury machines

VSIPL++ FFT Replacement

- Drop in replacement for VSIPL++ FFT
- FFT suffers from granularity issues for 1:1 function offload
  - Including data transfers always slower on Cray XD1
- Was used to look at VForce overheads
  - VForce software failsafe vs. VSIPL++
    - Included RTRM communication
    - Virtually no impact on performance
  - VForce hardware vs. Native C
    - Data copying from opaque views to DMA-able memory hurt performance (Future Work)

Beamformer

- Example large-granularity VForce function
- VForce supports asynchronous kernel control and data transfer
  - Important for getting max system performance
  - Used by XD1 beamformer to achieve additional speedup
    - Weight Application on FPGA concurrent with Weight Computation on CPU



Nvida Tesla and CUDA

- Tesla C870 GPU Board
  - Unified Shader Architecture
  - Higher ratio of transistors dedicated to arithmetic vs CPU
  - Massively parallel

http://www.nvidia.com/object/tesla_c870.html

- CUDA
  - General purpose development environment for Nvidia GPUs
  - Uses C-language extensions to express parallelism
  - Includes a toolchain (compiler, debugger, profiler), driver API, and libraries (CUFFT & CUBLAS)

Extending VForce to GPUs

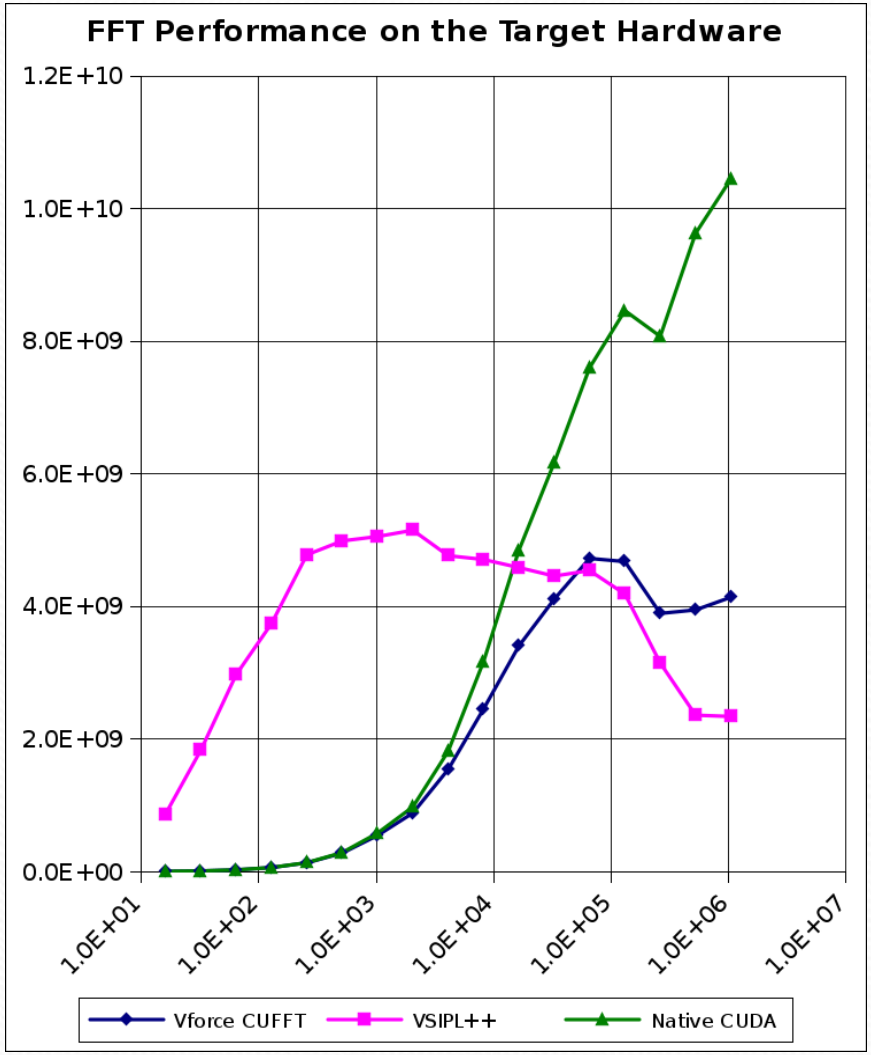
- Similarities to FPGAs
  - Data transfer to off-die accelerator
  - Pre-compiled kernels
- Differences in kernel execution
  - GPU kernels can be more flexible at runtime
  - Relatively small overhead for loading kernels vs FPGA
    - Allows executing multiple kernels & mixing and matching
- Differences in development
  - Tools still hardware specific
  - Fixed hardware, thousands of threads

VForce CUDA Support

- On FPGA platforms one SPP control library loads various FPGA bitstreams and handles all SPP control interface functionality
- RTRM search returns algorithm-specific control library
  - CUDA allows low-level bitstream-like functionality but not used
  - Higher-level method allows multiple kernels to be called if desired and the use of CUFFT and CUBLAS
- VForce tries to impose few HW requirements

FFT Results

- CUDA FFT uses CUDA libraries
  - CUFFT for FFT
  - CUBLAS for scaling
- Current results affected by data copying like XD1
- CodeSourcery VSIPL++ using FFTW on Intel Xeon 5110 (1.6 GHz dual core, 4 MB cache)
- Same exact application code as Cray XD1 FPGA FFT



Conclusions & Future Work

- User application code compiles unmodified between FPGA, GPU, and software only architectures
- Need more control over memory
- Support of new platforms: looking at Cell
- New applications

Thank You

Thanks to:

HPEC-SI

The MathWorks

Contact:
mel@coe.neu.edu

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