Implementing the VSIPL API on Reconfigurable Computers

Robin Bruce
Institute for System-Level Integration
Sponsor: Nallatech
What is VSIPL?

- Standard Software API
- Highly efficient and portable computational middleware for signal and image processing applications.
- High-performance computing
- Mostly defence-oriented & embedded
- Mainly floating-point functions
Why VSIPL?

Application Developers want:
- High Performance
- Portability & Maintainability
- High Productivity
- Open Standards

System Developers must:
- Abstract Implementation Details
- Conform to Standard Interfaces

VSIPL is the bridge
FPGA-based VSIP: Why?

- FPGAs have established advantages over CPUs for many floating-point algorithms
  - More FLOPS/W
  - More FLOPS/device
  - More sustained FLOPS
- Offer tight coupling to I/O devices
  - Good in embedded environments
- High Theoretical Data Bandwidths
- Very good for signal and image processing!
Implementation Environment

Hardware:

- Nallatech BenNUEY PCI FPGA Computing Card
  - Xilinx V-II 6000 Main User FPGA
  - 3 DIME-II Module Slots (up to 7 FPGAs)
  - 8Mbytes ZBT SRAM
Implementation Environment

- **Software:**
  - **DIME-C**
    - C-to-VHDL Compiler
    - Non-Cycle Accurate
    - Auto-parallelises and auto-pipelines subset of C
Implementation Environment

- **DIMETalk v3.1**
  - Creates Communication Network on FPGA
  - Acts as a Manual Linker of DIME-C Functions
Software Implementation of VSIPL

- Enhanced Reference Implementation of VSIPL (VSIPL/ERI)

  Naval Oceanographic Major Shared Resource Center

  Four Month Project to enhance TASP VSIPL

  Wrapped function calls to:
  - Basic Algebra Package (LAPACK)
  - Basic Linear Algebra Subroutines (BLAS)
  - Fastest Fourier Transform in the West (FFTW)
  - Math Library (math.h)

  These Libraries Do Not Exist for FPGAs
Comparing Performances

- High theoretical speedups possible when comparing FPGA computational kernels.

- Actual performance must be measured in a realistic setting.
CPU Function Call

- **Full execution time**
  - Standard CPU-based computing system
  - Includes fetching data and writing back data
  - Should represent best possible implementation on CPU system
Further to CPU call, must include:

- FPGA bitstream loading and system startup
- Data Transfer between host and reconfigurable PE
- Speedup estimates must be conservative and unbiased!
Investigation

Focussed Initially on:

**Single-Precision Vector and Matrix Operations**

- Matrix-Matrix Multiply, Vector Matrix Operations
- Unlimited Matrix and Vector Sizes make development of a single efficient architecture difficult
- Efficient implementations tied to memory structure of targeted reconfigurable computing platform

**Single-Precision FFT**

- FFT size – Limited to powers of two
- Efficient FFT implementation limited to 4096-point on XC2V6000 FPGA
Investigation

- **Elementary Functions applied to Vectors and Matrices**
  
  Example: `vsip_vexp_f`

  Exponential function applied to each element in array

  Highlighted lack of underlying math library cores

- **Focus of research is to address this**
  
  Need to develop “math.h” library

  Use standard HDLs to develop cores

  “Call” math functions from DIME-C

  “Calls” become instantiated cores
Example: Exponential Core

- Targeted at Virtex-4 devices (V4SX35-10)
  Operates on IEEE754 single-precision numbers

Resource Use:
- 232 Slices
- 3 blocks of 18kbit on-chip SRAM
- 8 DSP48s

20 cycles latency, pipelines at 240MHz

- Development is ongoing:
  Max Relative Error = 2.28E-07
  Average Relative Error = 5.20E-08
CPU Implementation: vsip_vexp_f

- VSIPL function to perform exponential function on a data vector of 32-bit IEEE754 values
- CPU execution time for 1M word data set
  - 3.2GHz Pentium D
  - 2 GByte RAM
  - TASP VSIPL, gcc -O3 compiled
- 79.3 ms execution time
FPGA Implementation: vsip_vexp_f

- FPGA execution time
  - Virtex-4 SX35 device on BenOne board
  - Core computation 1M cycles @ 150MHz = 6.7ms
  - Measured execution time of DIME-C function implementing exponential core
  - Data Transfer = 4MBytes/3.6GBytes/s = 1.1ms
    - Achievable with Today’s Machines
  - FPGA Context Switch ~ = 45ms
    - Time to transfer and load bitstream
    - Theoretical minimum of device

- 53.9 ms execution time
Exponential: FPGA vs. CPU

- Speedup Factor: 1.47x for 1M elements
  - 500k elements: 0.81x times speedup
  - 10k elements: 0.02x time speedup

- Raw Kernel Speedup Limit:
  - 24 Units at 240MHz on SX35 vs CPU:
    - 5.76 GOps/s vs. 12.5 MOps/s: **460x**!

- Bandwidth-Limited Speedup:
  - 3.6Gbytes/s = 0.9 GOps/s
    - 0.9 GOps/s vs. 12.5 Mops/s: **72x**!
Future Project Direction

- Focus on Developing Math Library
  - Cores should not be tied to specific FPGAs
    - Standard HDL portable between families and vendors
  - Cores have no external connections
    - Not tied to specific RC systems
  - Cores portable to other High-Level FPGA Tools
    - DIME-C used to integrate generic library cores

- FPGA “math.h” is an immediately useful foundation stone for FPGA VSIPL that represents good return on investment
De-emphasise focus on FPGA VSIPL until:

Technological improvements reduce FPGA function call time from milli- to micro-seconds

Virtex-5 introducing 32-bit SelectMap and multiboot

Building-Block Libraries exist:

- FPGA LAPACK
- FPGA BLAS
- FPGA FFTW
- FPGA math.h (project’s future)

Reconfigurable Architectures Stabilise
Reconfigurable Computing Languages Stabilise