Evaluation of the VSI PL++ Serial Specification Using the DADS Beamformer

HPEC 2004
September 30, 2004

Dennis Cottel (dennis.cottel@navy.mil)
Randy Judd (randall.judd@navy.mil)
SPAWAR Systems Center San Diego
VSI PL++ Demonstration

- HPEC-SI is moving VSI PL functionality to object oriented programming and C++: VSI PL++

- Goal of this demonstration:
  - Evaluate the draft VSI PL++ Serial Specification
  - Identify both advantages and problems with the VSI PL++ methodology
  - Suggest improvements

- Method
  - Port a DoD acoustic beamformer algorithm written in standard C to use VSI PL++ and C++
  - Measure and Evaluate (when compared to baseline code)
Deployable Autonomous Distributed System (DADS)

• DADS Goals
  - Develop and demonstrate deployable autonomous undersea technology to improve the Navy’s capability to conduct effective Anti-Submarine Warfare and Intelligence-Surveillance-Reconnaissance operations in shallow water

• Sponsor: ONR 321
DADS Concept

- Sensors, Arrays & Sources
  - Acoustic
  - Electromagnetic
- Communication Links
  - RF buoys & AUV gliders
  - Acoustic modems
- In-Node Signal Processing
  - Acoustic, passive & active
  - Electromagnetic
  - Sensor data fusion
- Master Node
  - Network control
  - Network data fusion
DADS Beamformer

• Signal processing program chosen for conversion is DADS multi-mode beamformer
  – Adaptive minimum variance distortionless response

• Current software is …
  – Sequential ANSI C
  – About 1400 lines of C source code
  – Pointer-ized -- no vectorization
Approach

- Establish test data and environment to execute and validate current code
- Analyze existing code and data structures
- Vectorize
- Rewrite module using VSI PL++
- Validate VSI PL++ version
- Report specification issues and code metrics

Used pre-release of CodeSourcery sequential VSI PL++ reference implementation which in turn uses the VSI PL reference implementation
Deliverables

• Metrics
  – SLOC
  – Lines changed if appropriate
  – Time to develop
  – Others

• Report results and lessons learned
  – HPEC-SI workshop
  – DADS Annual Program Review for ONR, project personnel, industrial partner (Undersea Sensor Systems Inc.)
Initial Steps

• Established testable code baseline
  – Wrapped module in executable program
  – Set up test data file and associated parameters
  – Set up validation procedures

• Analyzed baseline code
  – Figured out what algorithms were implemented
  – Mapped program data flow
Dual Implementations

• Starting from scratch based on analysis of original program
  – Insight, trial approaches to sub-problems

• Incremental modification of original program
  – Vectorization
    • Un-pointerize
    • Reorder tests within loops
    • Recast loops into vector and matrix operations
  – VSIPL++ -ization
  – This version chosen for final solution and metrics
Example of Typical Code

```c
frptr = fr;  // pointer to replica buffer (real)
fiptr = fi;  // pointer to replica buffer (imag)

for (ifreq = ibin1; ifreq <= ibin2; ifreq++)
    // produce one row of the weight matrix at a time
    for (iang = 0; iang < nang; iang++)  // loop over bearings
        for (i = 0; i < nh; i++)  // copy a row of the replica
            sr[i] = *frptr;
            si[i] = *fiptr;
            frptr++;
            fiptr++;

    for (i = 0; i < nh; i++)  // loop over hydrophones
        wr[i] = wt[i] * sr[i];
        wi[i] = wt[i] * si[i];

for (int ifreq = ibin1; ifreq <= ibin2; ifreq++)
    w = vsip::vmmul<0>(wt, replica.get_xy(ifreq-ibin1));
```
Code Metrics

- Number of files increased from 8 to 14
- SLOC for all source files
  - Counting semicolons:
    - Baseline: 887
    - VSI PL++: 630 -29%
  - Counting non-blank, non-comment lines:
    - Baseline: 1389
    - VSI PL++: 1018 -27%
- Heart of the beamformer calculation (all lines):
  - Baseline: 410
  - VSI PL++: 180 -56%
- Lines of code changed: Most!
Memory Size Metrics

• Binary program sizes (statically linked):

<table>
<thead>
<tr>
<th></th>
<th>HP-UX/PA-RISC</th>
<th>Red Hat/Pentium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>560 KB</td>
<td>700 KB</td>
</tr>
<tr>
<td>VSI PL++</td>
<td>1,800 KB</td>
<td>3,900 KB</td>
</tr>
</tbody>
</table>

• Memory footprint and usage:
  – Weren’t able to measure this
  – VSI PL++ programs might be expected to use larger structures
    • For example, N vectors become a matrix
  – For this program’s statically allocated structures and arrays, it should be a wash
Test Cases

- **64 input sensors, 64 output beams**
  - 64x64 covariance matrix
  - Forward FFTs 64 x 1024
  - Inverse FFTs 64 x 1024
  - Smaller data set
  - Fewer larger objects created, more computing per object

- **14 input sensors, 108 output beams**
  - 14x14 covariance matrix
  - Forward FFTs 14 x 2048
  - Inverse FFTs 108 x 2048
  - Larger data set
  - More smaller objects created, object creation amortized over less computing
Execution Time Examples

Baseline
VSIPL++

Execution time (seconds)

PA-RISC PowerPC Pentium PA-RISC PowerPC Pentium

64 sensors, 64 beams 14 sensors, 108 beams
Profiling Results for PA-RISC

64 sensors, 64 beams, 1024 point FFTs

14 sensors, 108 beams, 2048 point FFTs
Profiling Results for PowerPC

64 sensors, 64 beams, 1024 point FFTs

PowerPC, 1.25 GHz, OS X 10.3.4, g++ 3.3

14 sensors, 108 beams, 2048 point FFTs

PowerPC, 1.25 GHz, OS X 10.3.4, g++ 3.3
Profiling Results for Pentium

64 sensors, 64 beams, 1024 point FFTs

Baseline VSIPL++ on Pentium, 450 MHz, Red Hat 8.0, g++ 3.2

14 sensors, 108 beams, 2048 point FFTs

Baseline VSIPL++ on Pentium, 450 MHz, Red Hat 8.0, g++ 3.2
Object Creation

• Previous experience with VSIPL has shown
  – Object creation in inner loops is inefficient
  – Solution is early binding / late destroys

• VSIPL++ reference implementation uses VSIPL library as its compute engine
  – Observed similar inner-loop inefficiencies
  – C++ new() called to create subviews of data

• A purely C++ VSIPL++ implementation would avoid some of these problems
Overall Issues

• Additional data copying a potential problem
  – Improvements in reference library will remove some of this

• Memory allocation
  – A clever implementation might avoid much of this
  – Proposal to improve specification so implementation can avoid calls to C++ new() in inner loops

• Binary program size for embedded systems
VSI PL++ Specification

• Issues with specification
  – I/O for data Fixed in final spec
  – Row/Column major Fixed in final spec
  • matrix layout in memory
  – Real and Imaginary subviews Fixed in final spec
  – Sticky subview variables with remapping Proposed fix for final spec

• There were still limitations in the VSI PL++ reference implementation we used
  – Tensors
  – Transpose views and operations
Ongoing VSI PL++ Questions

• Knowing when data is copied and when it isn’t and what we can do about it: there are subtle C++ distinctions
• Continuing general concern about efficiency
• Use of bleeding-edge C++ features and compiler compatibility
Our Contributions

• Demonstrated that VSI PL++ can be used for real DoD application code
• Close look at details improved specification
  – Fixing inconsistencies and small errors
  – Improving understandability of the spec
• Redesign of the FFT and multiple-FFT API
• Bug fixes in reference implementation
• Improvements to underlying VSI PL reference library
Conclusions

- VSI PL++ serial specification has the functionality to implement a typical DoD signal processing application

- Resulting code is more understandable and maintainable

- VSI PL++ can deliver comparable performance
Evaluation of the VSI PL++ Serial Specification Using the DADS Beamformer

HPEC 2004
September 30, 2004

Dennis Cottel (dennis.cottel@navy.mil)
Randy Judd (randall.judd@navy.mil)
SPAWAR Systems Center San Diego