Evaluation of the VSIPL++ Serial Specification Using the DADS Beamformer

Dennis Cottel **and** ***Randy Judd***
SPAWAR Systems Center San Diego
Phone: 619-553-1645
Email Address: {dennis.cottel,randall.judd}@navy.mil

The High Performance Embedded Computing Software Initiative (HPEC-SI) Development Working Group has been creating VSIPL++, a new software standard to promote portability, productivity, and performance in embedded parallel systems. This standard expands the Vector Signal Image Processing Library (VSIPL) to encompass parallel systems in C++. HPEC-SI has contracted with CodeSourcery, LLC, to produce a VSIPL++ Reference Library. The Reference Library is intended to allow early users to experiment with the functionality of VSIPL++.

This presentation discusses a project to evaluate the VSIPL++ specification by using the CodeSourcery VSIPL++ Reference Library to implement a part of the current operational signal processing code for the Deployable Autonomous Distributed System (DADS). The authors of this presentation have extensive experience in developing and using the original VSIPL library, working with parallel signal processing algorithms, and developing other HPC middleware and standards. They have been involved with the development of the VSIPL++ standard, but are not C++ programming experts. One aspect of this work will be to compare features and ease-of-use of VSIPL and VSIPL++.

The Deployable Autonomous Distributed System (DADS) is an advanced development program, sponsored by ONR-321, to demonstrate deployable autonomous undersea technology for operations in coastal waters. The system consists of small acoustic arrays on the ocean floor with embedded in-node signal processing. Detections are transmitted to the surface using acoustic modems.

The current DADS acoustic beamforming software is written in ANSI C, is available in both development and embedded configurations, and is unclassified. The code is sequential, but future hardware and algorithm upgrades could require parallelization. Of several candidate software modules, the beamformer was chosen as most appropriate for conversion to VSIPL++. The DADS beamformer does either adaptive processing (using a minimum variance distortionless response algorithm) or conventional beamforming. The beamformer source code consists of about 1000 lines of non-blank, non-comment code.

The project established a test data set and an environment in which the current DADS beamforming code could be executed. The code was then rewritten in C++ using the CodeSourcery VSIPL++ Reference Library and the results of running the new code were verified. Metrics were recorded on the time to develop the code and the resulting changes in lines of code from the original version.

We will report on these metrics and other lessons learned. Of particular interest will be whether addressing real-world algorithms exposes any functional problems or deficiencies of the VSIPL++ specification that should be addressed by the HPEC-SI Working Group.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 FEB 2005	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Evaluation of the VSIPL++ Serial Specification Using the DADS Beamformer	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **SPAWAR Systems Center San Diego**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM00001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004 Volume 1., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**26**	

Evaluation of the VSIPL++ 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

VSIPL++ Demonstration

- HPEC-SI is moving VSIPL functionality to object oriented programming and C++: VSIPL++
- Goal of this demonstration:
  - Evaluate the draft VSIPL++ Serial Specification
  - Identify both advantages and problems with the VSIPL++ methodology
  - Suggest improvements
- Method
  - Port a DoD acoustic beamformer algorithm written in standard C to use VSIPL++ and C++
  - Measure and Evaluate (when compared to baseline code)

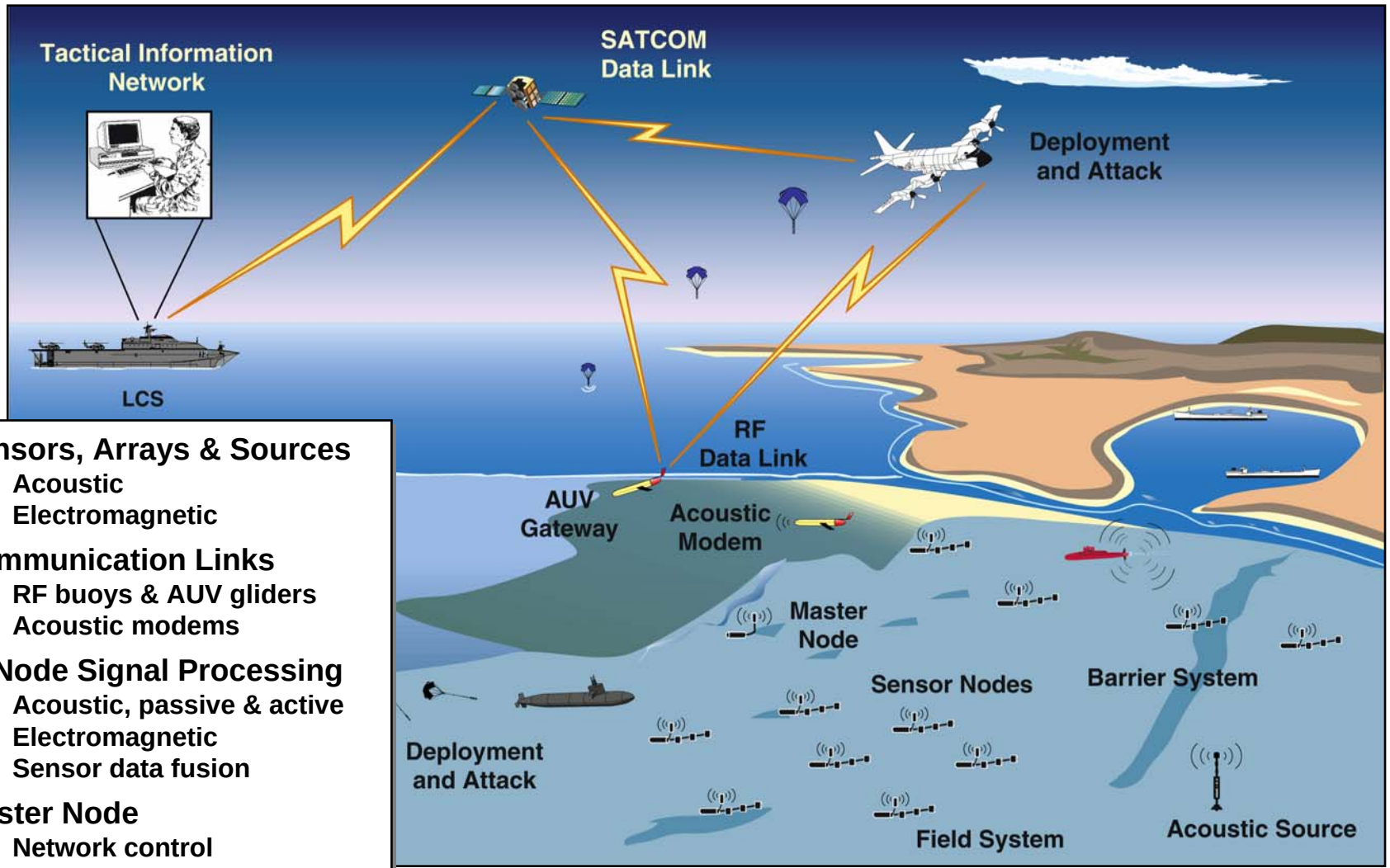
Deployable Autonomous Distributed System (DADS)

- 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

  http://www.onr.navy.mil/sci_tech/ocean/321_sensing/info_deploy.htm

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 VSIPL++
- Validate VSIPL++ version
- Report specification issues and code metrics

Used pre-release of CodeSourcery sequential VSIPL++ reference implementation which in turn uses the VSIPL reference implementation

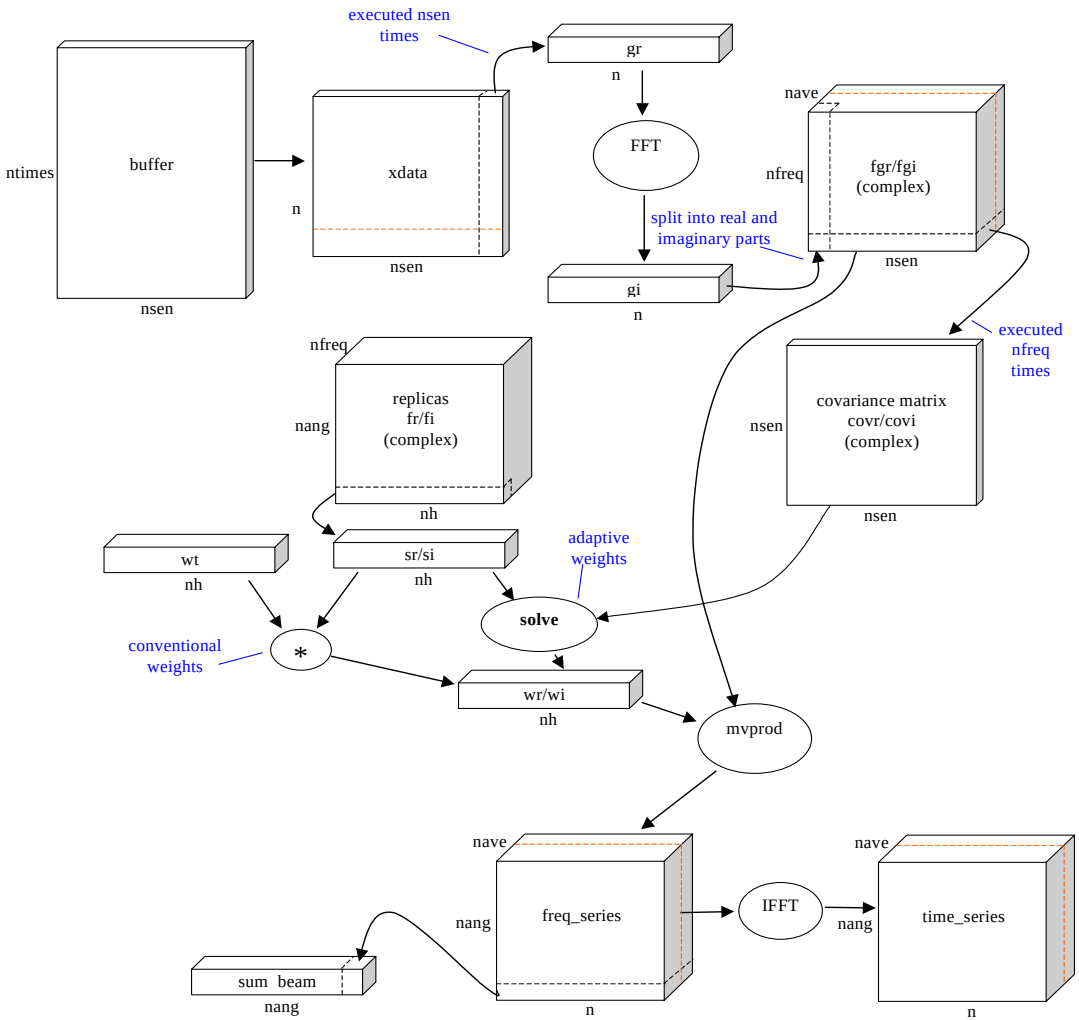
Deliverables

- 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

Data Flow Map



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

```
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
    - VSIPL++ 630 **-29%**
  - Counting non-blank, non-comment lines:
    - Baseline 1389
    - VSIPL++ 1018 **-27%**
- Heart of the beamformer calculation (all lines):
    - Baseline 410
    - VSIPL++ 180 **-56%**
- Lines of code changed: Most!

Memory Size Metrics

- Binary program sizes (statically linked):

	HP-UX/PA-RISC	Red Hat/Pentium
– Baseline	560 KB	700 KB
– VSIPL++	1,800 KB	3,900 KB

- Memory footprint and usage:
  - Weren't able to measure this
  - VSIPL++ 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

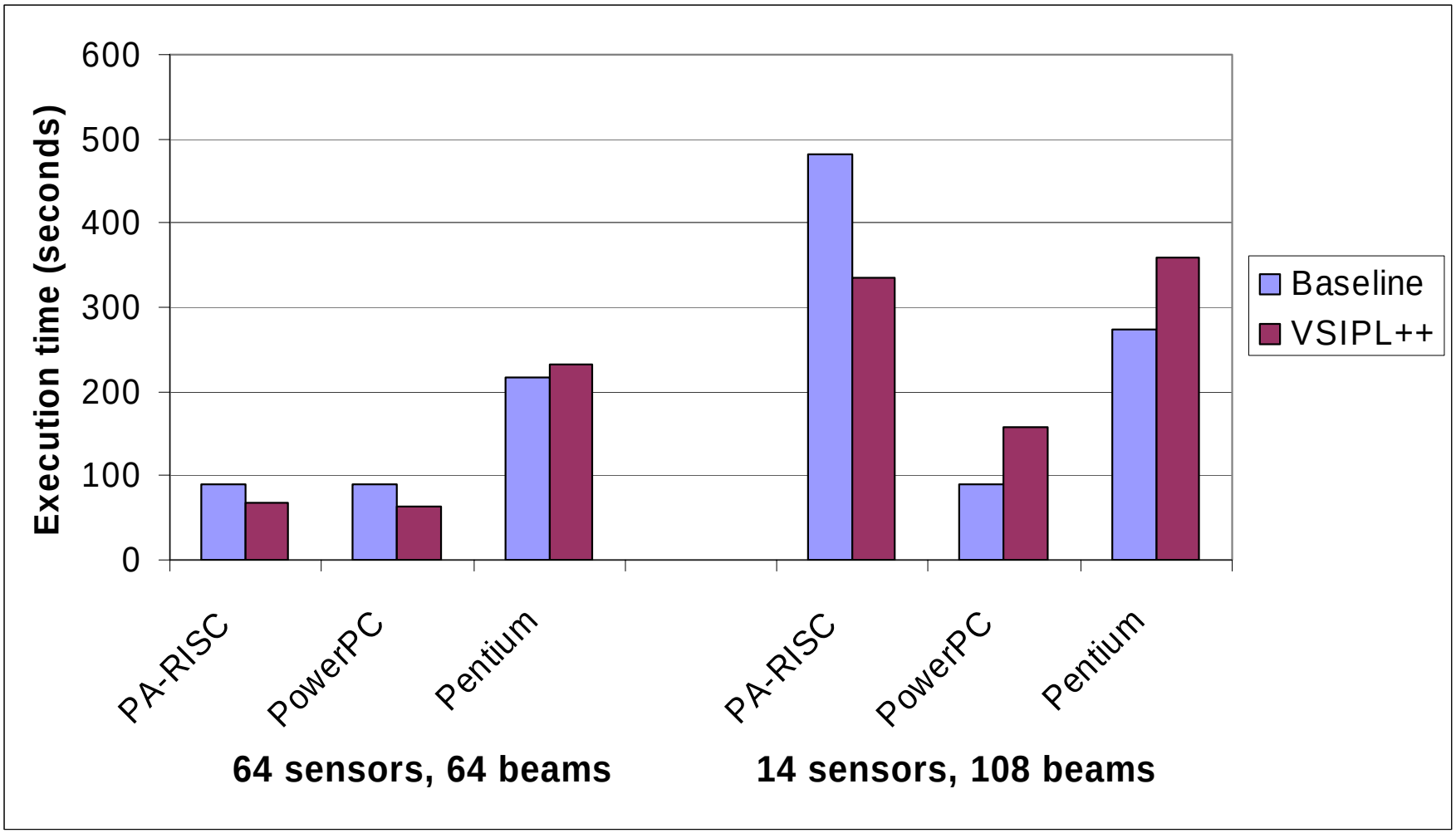
- **64 input sensors, 64 output beams**
- 64x64 covariance matrix
- Forward FFTs 64 x 1024
- Inverse FFTs 64 x 1024
- Smaller data set

- Fewer larger objects created, more computing per object

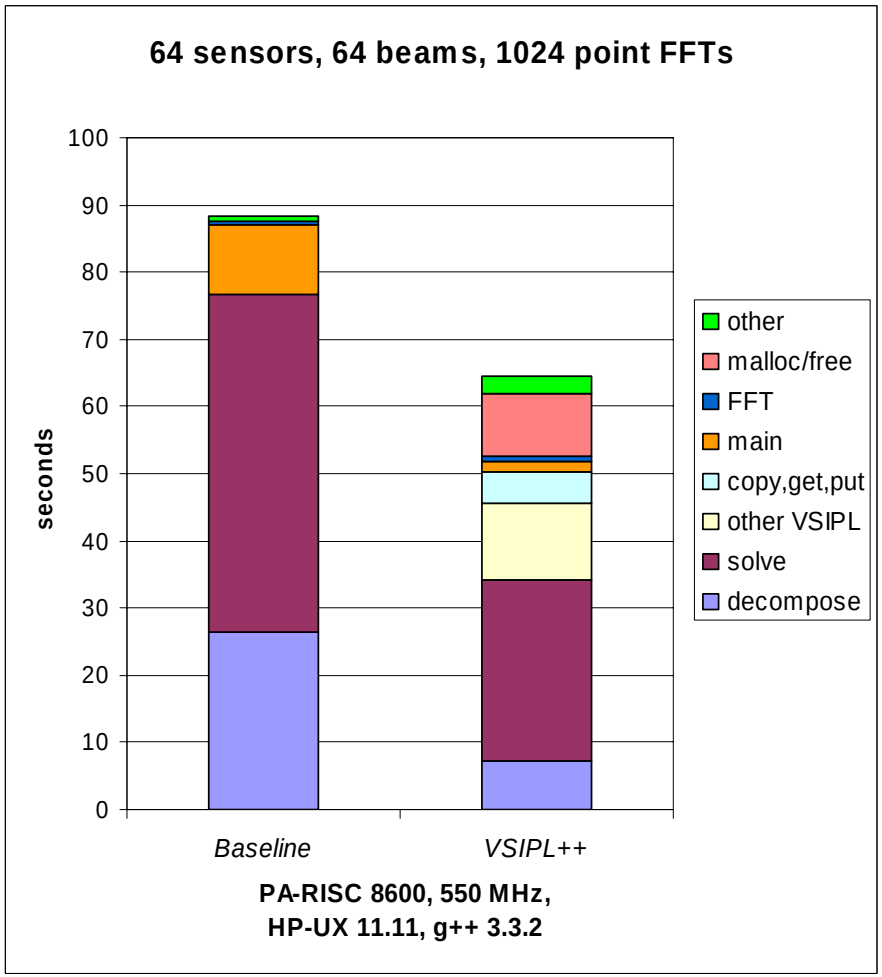
- **14 input sensors, 108 output beams**
- 14x14 covariance matrix
- Forward FFTs 14 x 2048
- Inverse FFTs 108 x 2048
- Larger data set

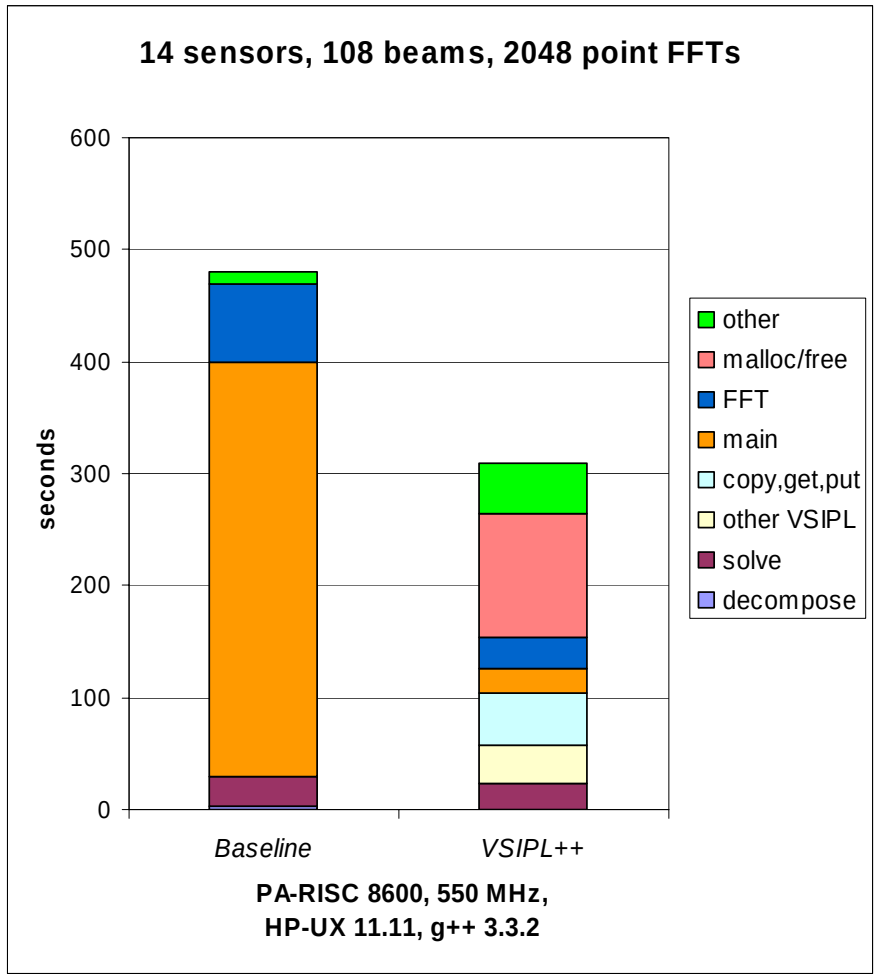
- More smaller objects created, object creation amortized over less computing

Execution Time Examples

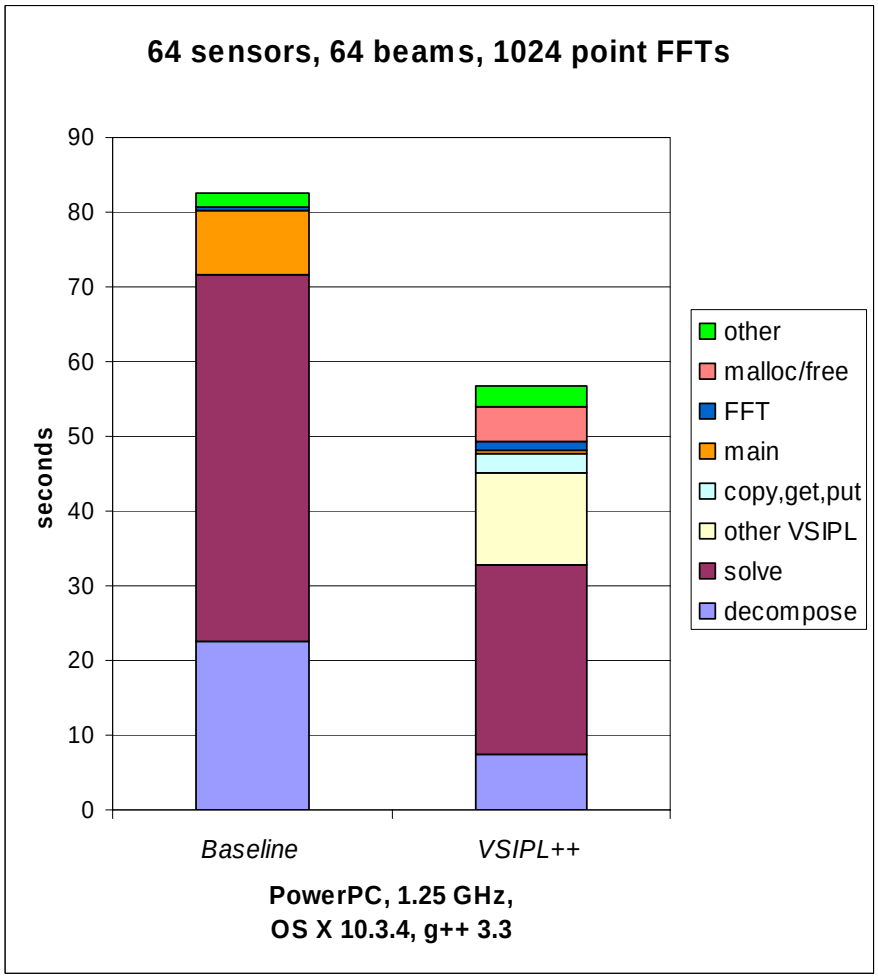


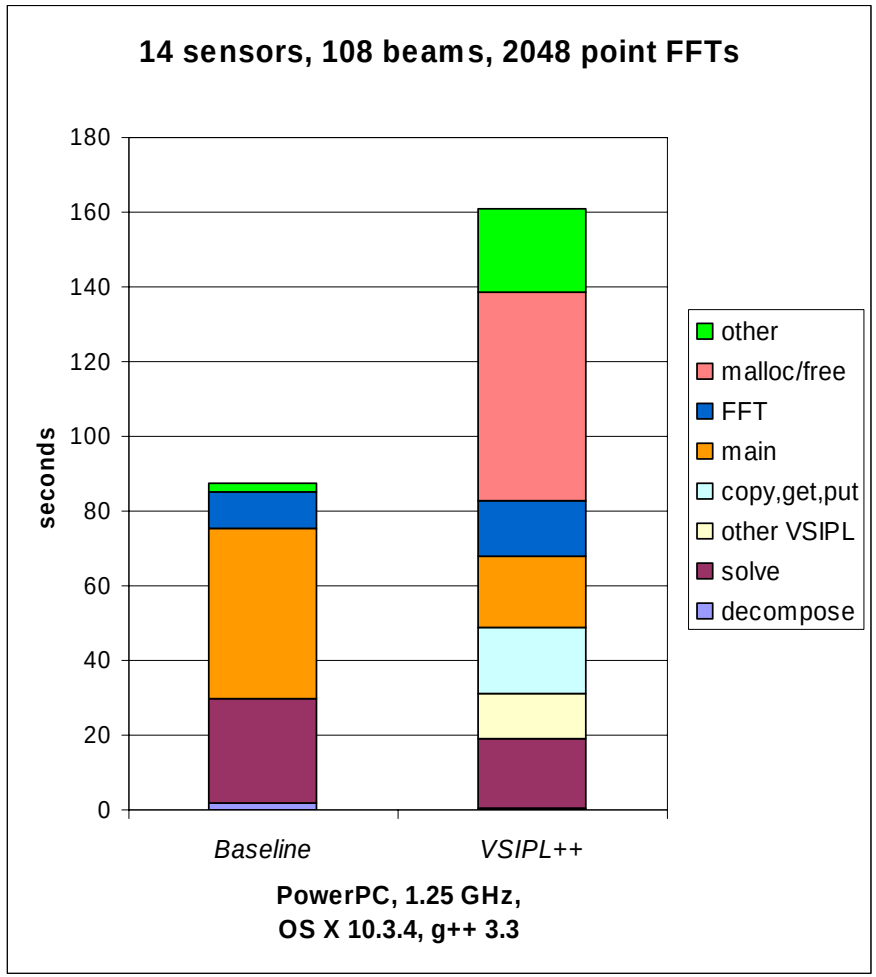
Profiling Results for PA-RISC



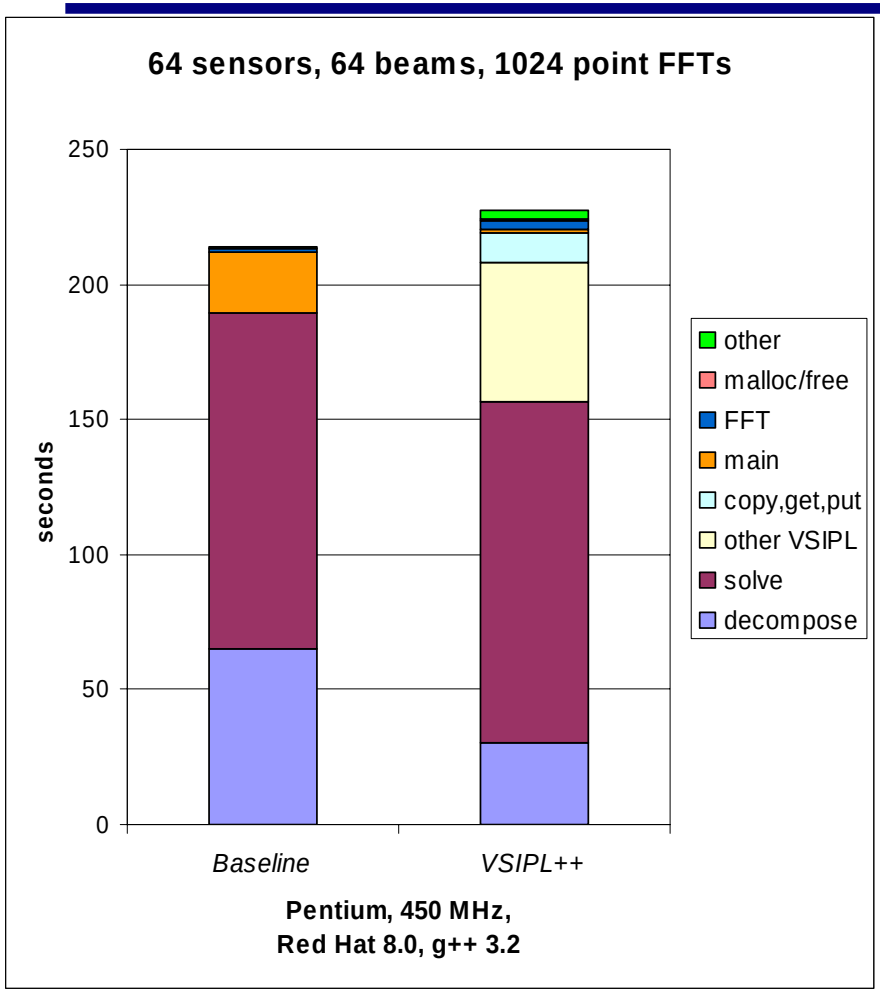


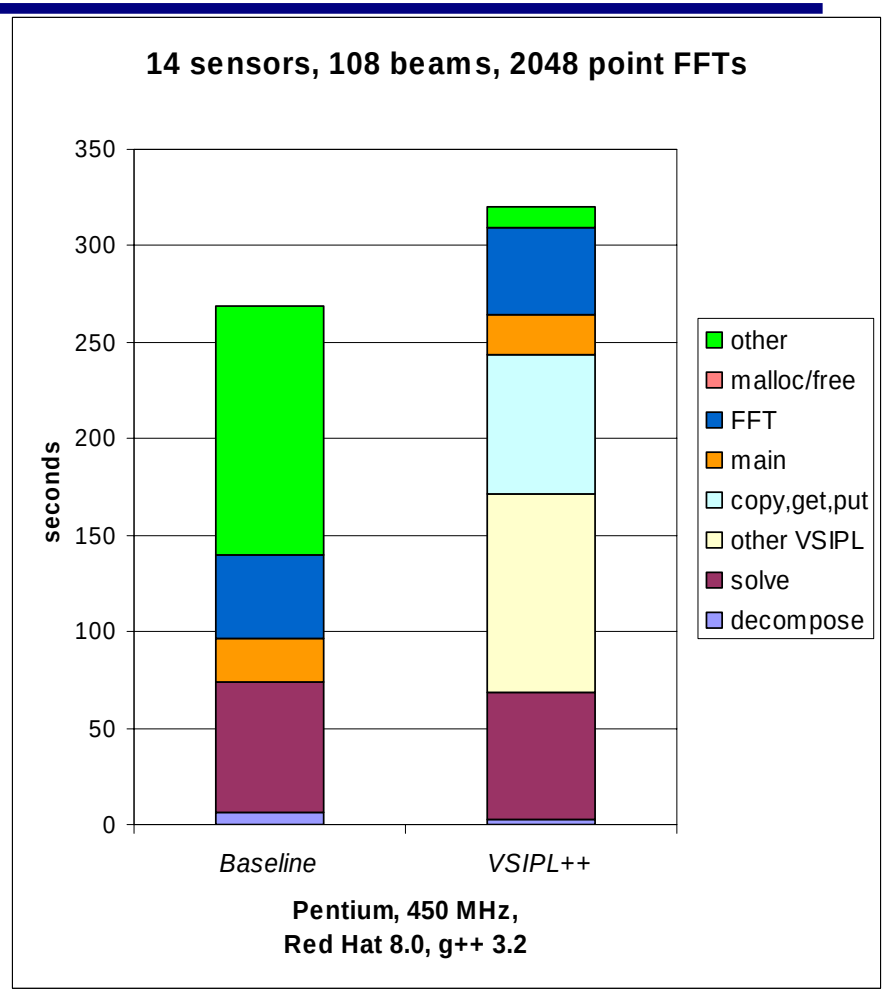
Profiling Results for PowerPC





Profiling Results for Pentium





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

VSIPL++ 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 VSIPL++ reference implementation we used
  - Tensors
  - Transpose views and operations

Ongoing VSIPL++ 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 VSIPL++ 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 VSIPL reference library

Conclusions

- **VSIPL++ serial specification has the functionality to implement a typical DoD signal processing application**
- Resulting code is more understandable and maintainable
- VSIPL++ can deliver comparable performance

Evaluation of the VSIPL++ 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