

# 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

---

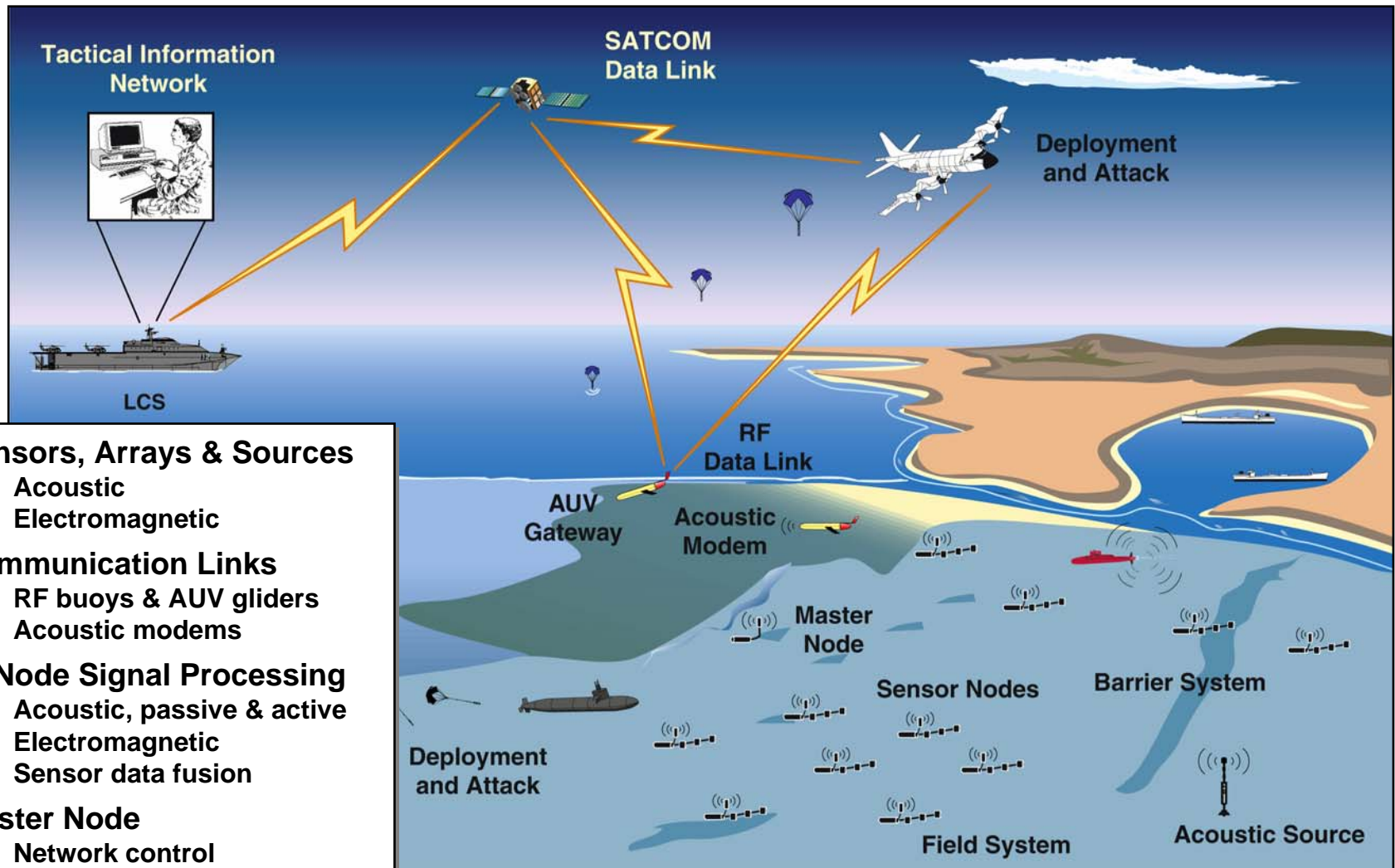
- HPEC-SI is moving VSIPL functionality to object oriented programming and C++: VSIPL++
- Goal of this demonstration:
  - Evaluate the draft VSIPL++ Serial Specification
  - Identify both advantages and problems with the VSIPL++ methodology
  - Suggest improvements
- Method
  - Port a DoD acoustic beamformer 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](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.)

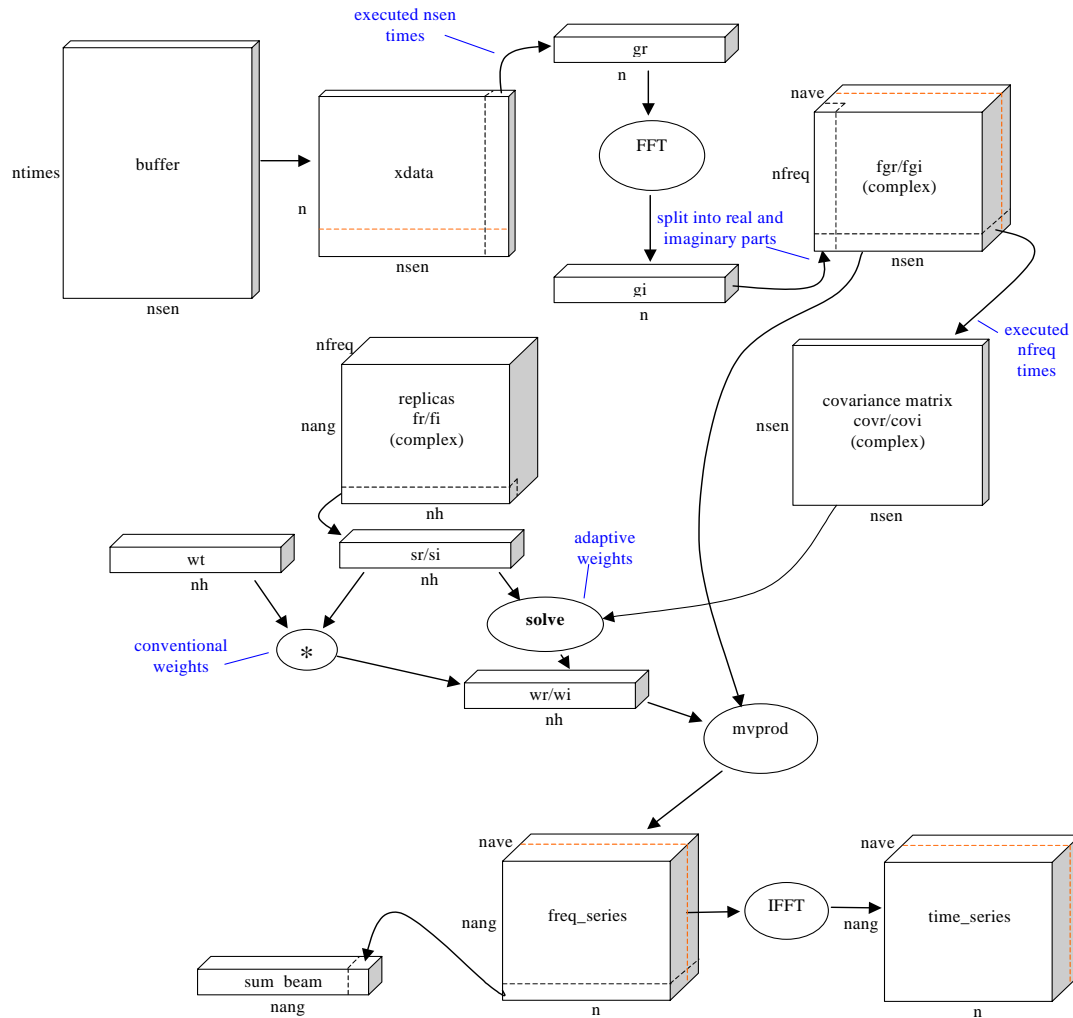
# Initial Steps

---

- 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)
fi ptr = fi; // pointer to replica buffer (imag)

for (i freq = i bin1; i freq <= i bin2; i freq++)

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

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

```
for (int i freq = i bin1; i freq <= i bin2; i freq++)
    w = vsip::vmmul<0>(wt, replica.get_xy(i freq-i bin1));
```

# 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):

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

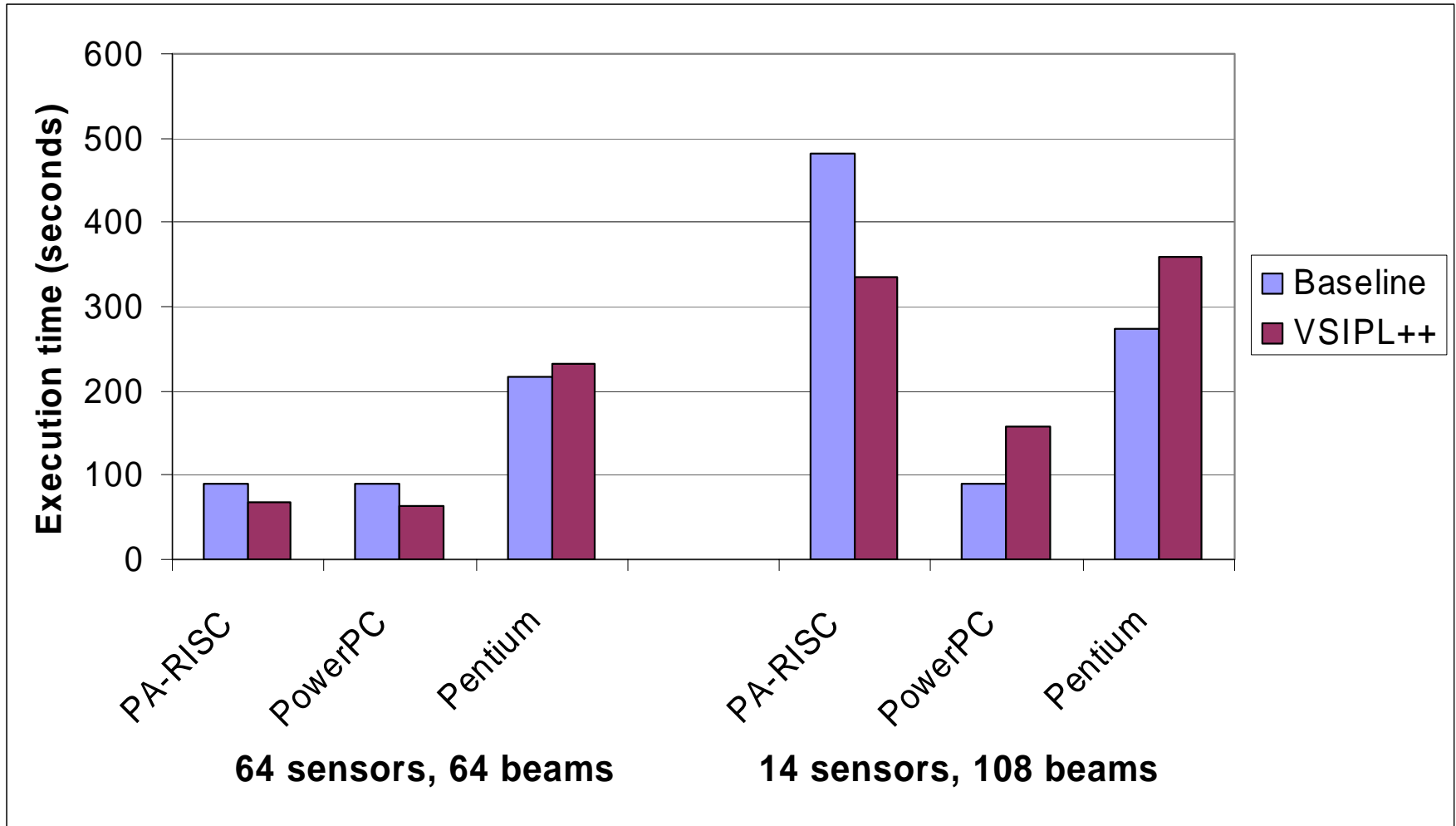
- Memory footprint and usage:
  - Weren't able to measure this
  - VSIPL++ programs might be expected to use larger structures
    - For example, N vectors become a matrix
  - For this program's statically allocated structures and arrays, it should be a wash

# Test Cases

---

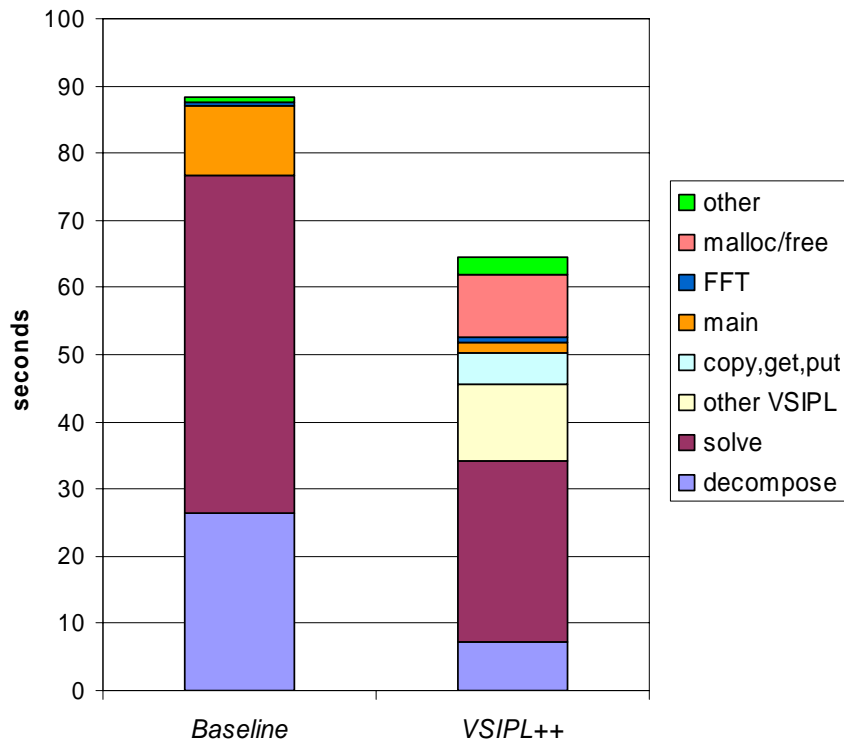
- **64 input sensors, 64 output beams**
  - 64x64 covariance matrix
  - Forward FFTs 64 x 1024
  - Inverse FFTs 64 x 1024
  - Smaller data set
  - Fewer larger objects created, more computing per object
- **14 input sensors, 108 output beams**
  - 14x14 covariance matrix
  - Forward FFTs 14 x 2048
  - Inverse FFTs 108 x 2048
  - Larger data set
  - More smaller objects created, object creation amortized over less computing

# Execution Time Examples



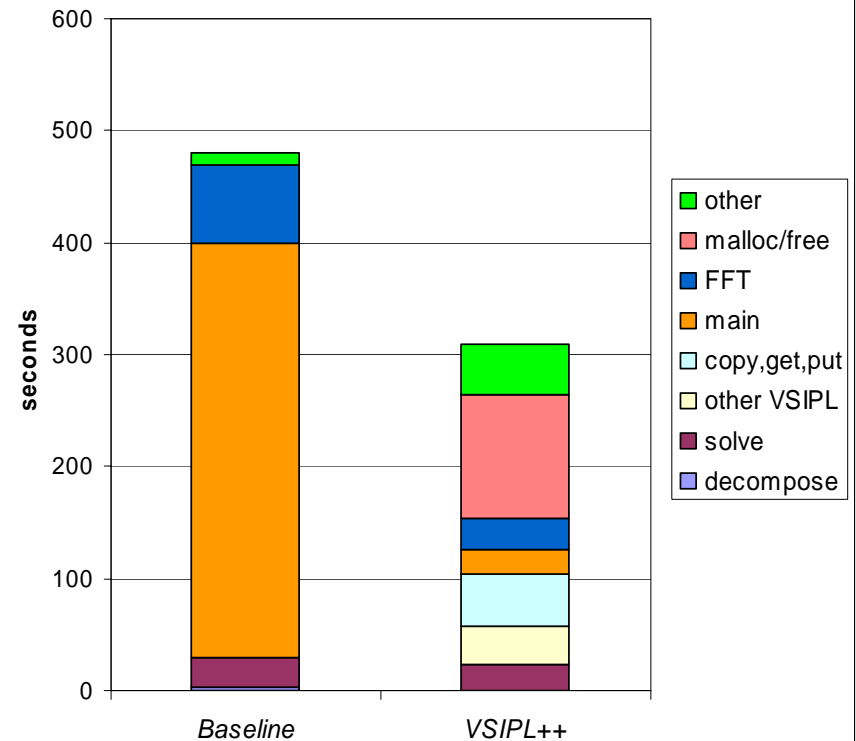
# Profiling Results for PA-RISC

64 sensors, 64 beams, 1024 point FFTs



PA-RISC 8600, 550 MHz,  
HP-UX 11.11, g++ 3.3.2

14 sensors, 108 beams, 2048 point FFTs

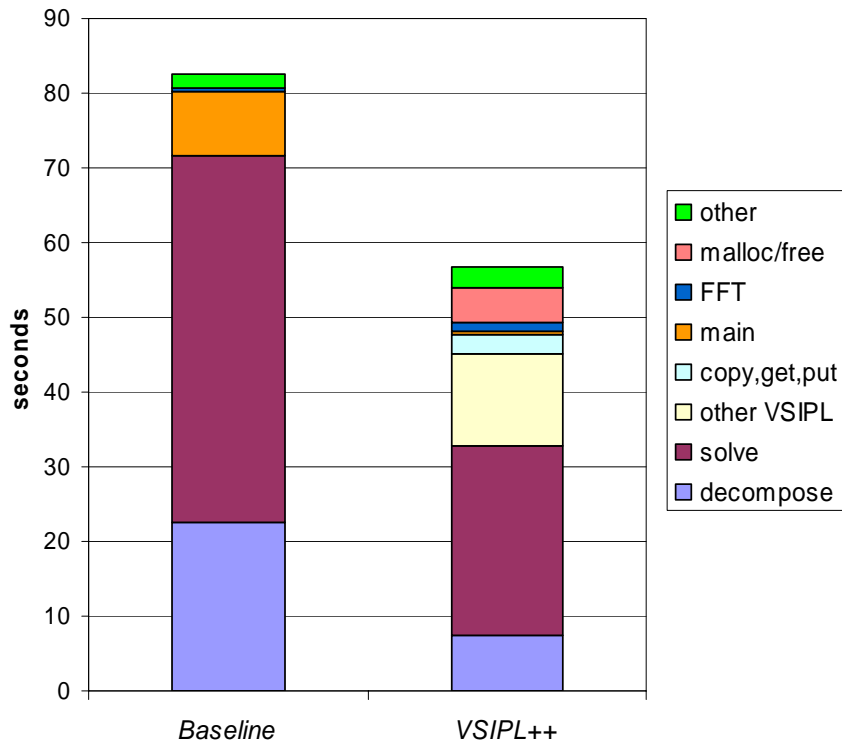


PA-RISC 8600, 550 MHz,  
HP-UX 11.11, g++ 3.3.2



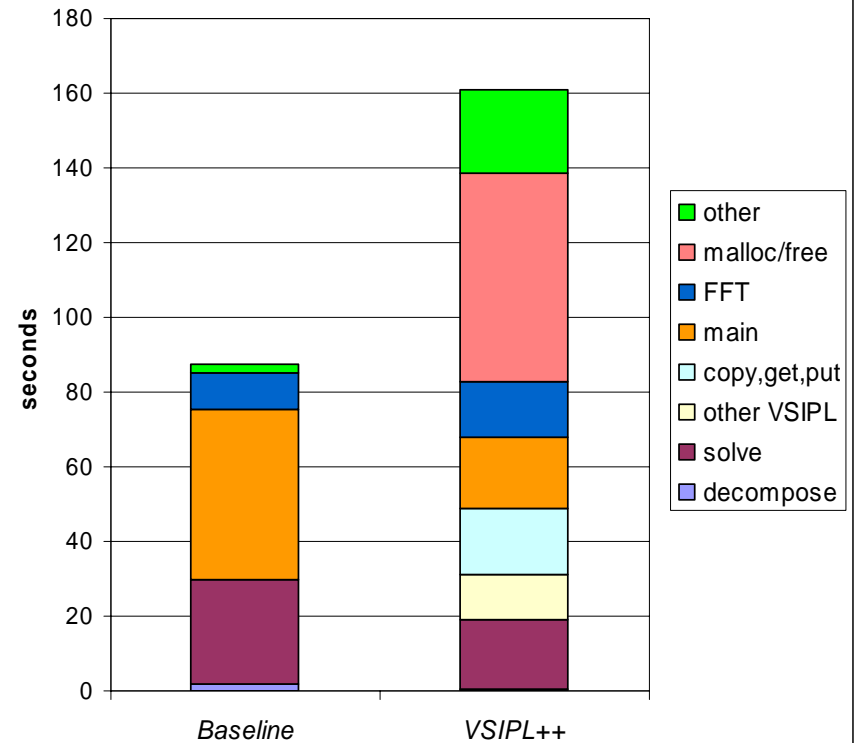
# Profiling Results for PowerPC

64 sensors, 64 beams, 1024 point FFTs



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

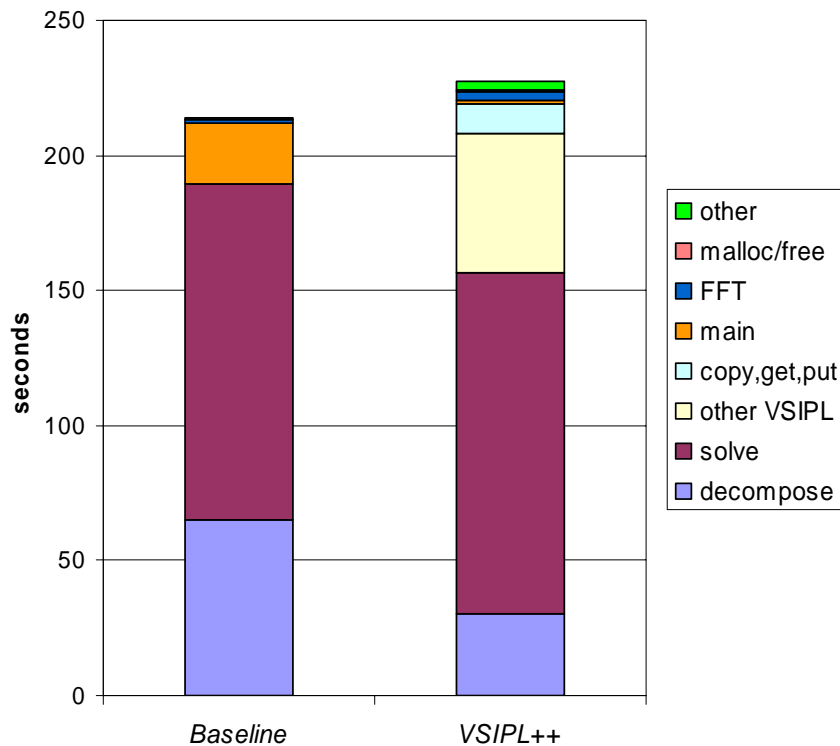
14 sensors, 108 beams, 2048 point FFTs



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

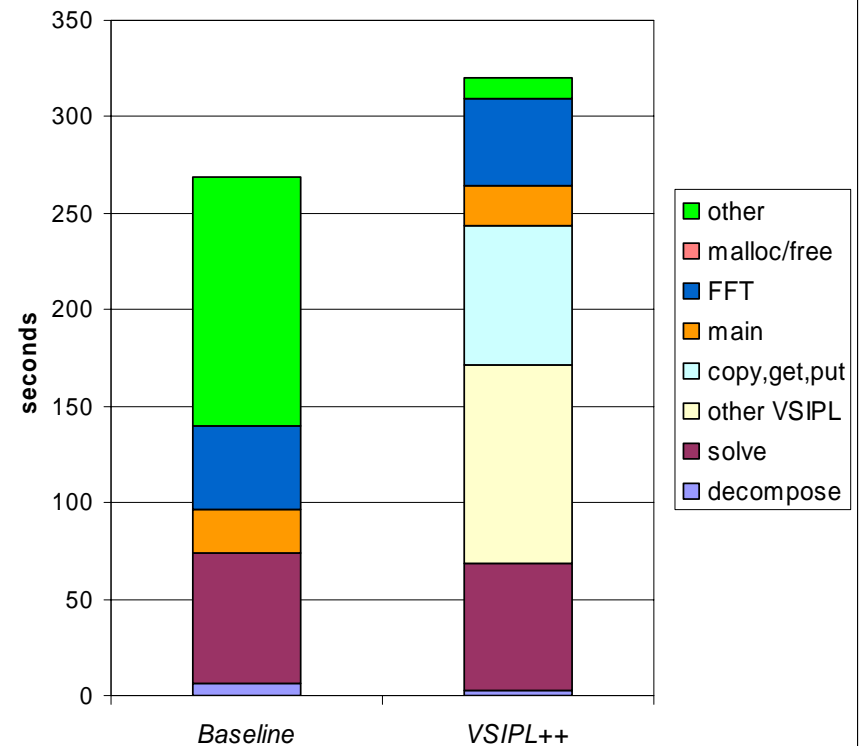
# Profiling Results for Pentium

64 sensors, 64 beams, 1024 point FFTs



Pentium, 450 MHz,  
Red Hat 8.0, g++ 3.2

14 sensors, 108 beams, 2048 point FFTs



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

# 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