



API and Product Status of the v1.0 Vector, Signal, and Image Processing Library (VSIP L)

R. S. Janka¹, R. Judd², J. Lebak³, M. A. Richards¹, D. Schwartz⁴

¹Georgia Tech Research Institute, Atlanta, GA

²U.S. Navy SPAWAR Systems Center, San Diego, CA

³MIT Lincoln Laboratory, Lexington, MA

⁴HRL, LLC, Malibu, CA



Goal



- Define an *industry standard* API for embedded signal (vector/signal/image) processing
 - Maximize portability
 - Maximize performance
 - Define useful profile(s)



VSIP L Software



- **Freeware**
 - TASP VSIP L Demonstration Library
 - Test Suite
- **Commercial VSIP L implementations**
 - Embedded multiprocessing vendors
 - CSPI
 - Mercury Computer Systems (announced)
 - SKY Computers (announced)
 - Annapolis Micro Systems (tentative)
 - MPI Software Technology Inc. (announced)
- **Profiles**
 - Core Lite initially
 - Varying plans for Core and/or other supersets



VSIP L Library Properties



- **Based On ANSI C**
- **Two modes of operation**
 - Development mode with extensive error checking
 - Performance mode with minimal error checking
- **Object-based API**
- **Strong data typing**
- **Opaque abstract data types**
- **Vectors, matrices, and 3-tensors based on contiguous memory data blocks (not C array of arrays)**
- **Matrices can be row-major or column-major to simplify mixed library support**
- **Profiles**
 - Subset of VSIP L targeted at a specific user need.



VSIP L Data Spaces



VSIP L has two logical memory spaces

User Data Space

- User manipulates data using
 - Direct access
 - I/O functions
 - Other math libraries
 - Communication libraries (e.g. MPI, MPI/RT)
- VSIP L will not operate on data in user space

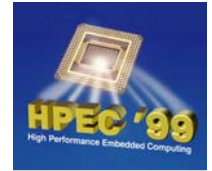
VSIP L Data Space

- User manipulates data using VSIP L functions (only)
- Memory hierarchy details hidden
- Implementation may optimize memory use
 - Chaining
 - Deferred execution
 - Strip-mining

These logical spaces may be the same physical address space, depending on the implementation



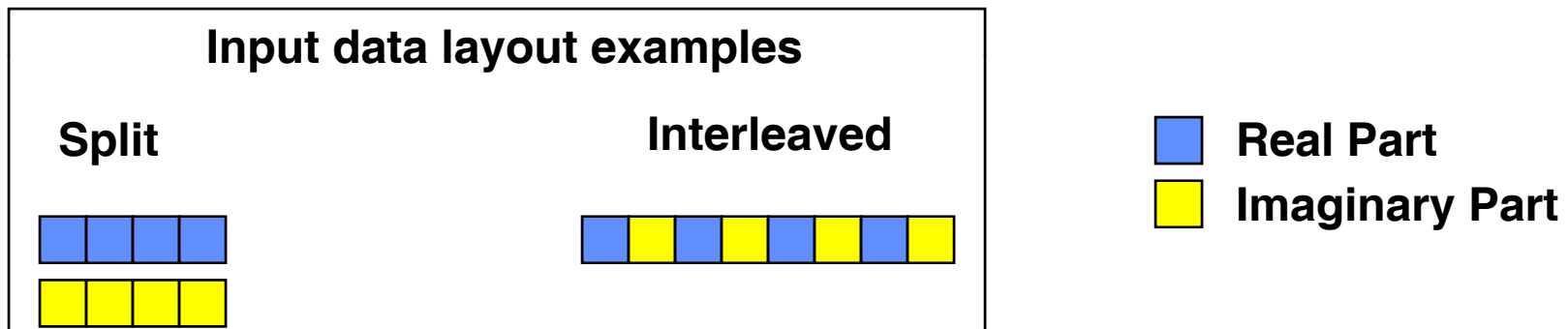
Supported Data Types



Boolean, integer, floating-point, and index types

Complex integer and floating-point data

- Input data may be stored split or interleaved
- Implementation allowed to choose preferred order for VSIP L space



Portable precision specifiers for user-defined types

Float

At least n decimal digits of accuracy

Integer

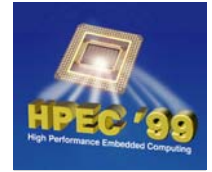
Exactly n bits

At least n bits

Fastest type of at least n bits



Blocks and Views



- A *block* is a contiguous storage area
- Data in a block may be *viewed* as a vector, matrix, or 3-tensor
- View characteristics:
 - Offset from start of block
 - Length (number of elements in view)
 - Stride (spacing between elements in view)

Example data in a block

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

View of entire block as a vector
(offset 0, length 9, stride 1)

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

View of even-numbered elements
(offset 1, length 4, stride 2)

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

View of block as 3 by 3 matrix
(offset 0, row length 3, row stride 1,
column length 3, column stride 3)

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

1	2	3
4	5	6
7	8	9



Vector Processing



- Elementary math operations
- Binary and ternary vector arithmetic
 - $R=A+B$ (Add, binary)
 - $R=A*B+C$ (Multiply-add, ternary)
- Comparison operations (<,>=)
- Selection operations
- Bitwise logical operations (AND,OR,NOT,XOR)
- Data conversion
 - Vector gather and scatter
 - Complex conversion to and from polar form



Signal Processing



- FFT, filter, convolution, and correlation functions use opaque objects for storage
- Separate calls for setup and inner-loop operations

FFT

- 1D, Multiple 1D, 2D, 3D
- Real-to-complex, complex-to-real
- In-place and out-of-place

Filters

- FIR
- IIR

Histogram

Convolution and Correlation

- 1D
- 2D

Window Functions

- Blackman
- Chebyshev
- Hanning
- Kaiser



Linear Algebra



- VSIP L provides
 - All-in-one solvers, for convenience and implementation optimization
 - Individual decompositions, for special algorithms
- Implementation allowed to choose the algorithm

System Solvers

- General Linear System
- Symmetric Positive Definite Linear System
- Toeplitz System
- Covariance System
- Linear Least Squares Problem

Decompositions

- LU
- Cholesky
- QR
- SVD

Matrix and Vector Operations

- Vector inner & outer products
- Matrix-vector multiply
- Matrix-matrix multiply
- Kronecker product



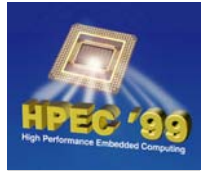
Test Suite Goals & Philosophy



- **Test suite code to be publicly released**
 - Distribution via VSIP L web page
 - Version controlled
- **Honor system (independent testing organization possible)**
 - Compliance with a specific version or profile
 - No degrees of compliance (all or nothing)
- **Test suite relies on test-generated data**
 - Both special case values and randomly generated samples
 - Fixed, trusted data sets may be used in special cases
 - Traditional Cody & Waite approach for elementary functions
- **Test suite *does* test**
 - Completeness, API behavior, functionality
- **Test suite does *not* test**
 - Accuracy—it is reported, but not “tested”
 - Performance (speed and/or memory)



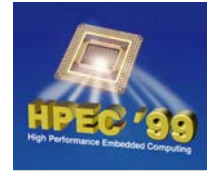
Methodology



- **Test completeness**
 - Are all prototypes in group or profile?
- **Test API behavior**
 - Did function build as advertised?
- **Test functionality**
 - Did function work as advertised?
 - Per “Functionality” & “Return Value”
 - Did function break as advertised?
 - Per “Errors”
 - Did function behave as advertised?
 - Per “Notes/References”
- **Try to be “pure” by testing “downstream”**
 - Do not use untested functions in code of functions under test...
 - ...Unless of course we have to (we’re engineers....)



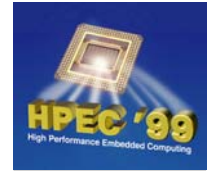
Status



- **Test Suite Lite**
 - Tests compliance for Core Lite profile
- **Release plan is for staged roll-out of incremental modules**
 - Released as ready to support vendors
 - Final packaged module integration will be TSL v1.0 (late Nov.)
- **Modules**
 - **Block: 10 May (alpha) & ~13 June (beta)**
 - **View: 6 August (alpha)**
 - **Elementwise: 17 September**
 - **Signal Processing: 15 October**
- **“Test Suite Working Group” (TSWG) resolves testing issues**
 - **GTRI (lead) & SPAWAR**
 - **Vendors (CSPI, Mercury, SKY)**
 - **Industry (Lockheed Martin)**



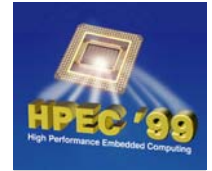
TASP VSIP L Background



- The Tactical Advanced Signal Processing Common Operating Environment working group needed a VSIP library suitable for demonstrating VSIP L functionality to potential users
- The Reference Library being produced by Hughes Research Laboratory did not appear to have sufficient performance for use as a demonstration product
- Using an early pre-release alpha version of the Reference Library as a starting point TASP endeavored to rewrite it to be suitable for demonstration of VSIP L
- The result was termed the TASP VSIP L Demonstration Library



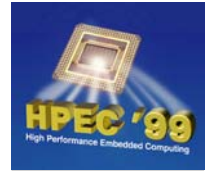
TASP VSIP L Demonstration Library



- Purpose of distribution is to provide code that is
- **Public Domain**
- **Suitable for demonstrating VSIP L programming techniques and methods**
- Goals of distribution are
- VSIP L Compliance for **performance mode** including functions
- **Core and Core Lite Profiles**
 - In The Core Lite Profile
 - Library routines that produce the correct answer
- **Source code which compiles on any platform with ANSI C**
- **Stand alone tutorial documentation keyed to the distribution**
- **Reasonable performance, but not necessarily fast**



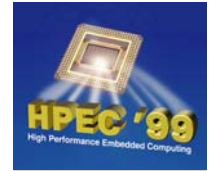
TASP VSIP L Status



- Available from www.vsipl.org
- Beta versions of Core profile and the Core Lite profile are available for Distribution
- Draft versions of documentation are available with the distributions in PDF (Acrobat Reader) format
- Debugging and testing continues as the library is used
- Additional work may be done to
 - Improve performance of
 - Fourier Transform
 - Filters
 - Linear algebra
 - Add additional elementwise Matrix functionality
 - Improve development mode checking
 - Development mode compliance is not a goal for TASP VSIP L



Potential TASP VSIP Uses



- **Directly**
 - As a demonstration library on Workstations
 - As a demonstration library on Embedded Platforms
- **Indirectly**
 - As a starting point for writing performance code for a selected profile needed for a particular application
 - To encapsulate platform specific library calls in a VSIP wrapper
- **For Development and Testing**
 - Develop code for embedded products on a workstation
 - Compare workstation results with embedded results



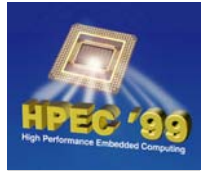
Code Encapsulation



- If a platform specific library uses memory pointers and strides to access data, then the vendor library call may frequently be encapsulated in TASP VSIP L by replacing the inner loop in the TASP VSIP L call with the appropriate vendor call.
- **Complex requires caution**
 - For complex arrays TASP VSIP L uses either interleaved or split data arrays of type float
 - For functions involving complex, if the vendor call uses an array of type complex,
`typedef struct {real, imaginary} complex;`
then the layout of the complex real and imaginary values in memory is not defined by ANSI C.
Frequently an array of complex will look like an array of interleaved complex, but this is platform specific.



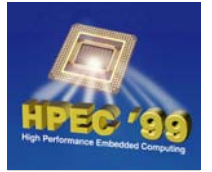
Code Encapsulation Partial VSIP L Compliance



- A platform specific library encapsulated in TASP VSIP L may not be suitable to support total VSIP L compliance but will support partial compliance
 - Partial compliance means the user can write VSIP L code which will run on any VSIP L compliant library
 - Partial compliance will not allow a user to run all VSIP L code on the platform, but code written that runs on the platform is portable to any VSIP L compliant platform



Using the TASP VSIP L Distribution



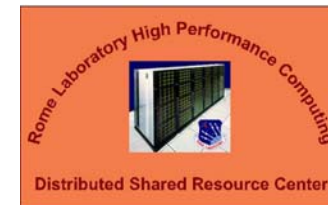
- Get the distribution from www.vsipl.org
- The distribution is developed in a Unix environment
 - Unzip and untar the distribution.
 - Make the distribution using the *make* Unix command
 - You may also *make* the distribution on NT or Windows 95/98 using the free Cygwin environment available from Cygnus. This environment has a make utility which works with the TASP VSIP L make files.
 - Make files will need to be modified for the chosen compiler, linker, and C options.
- No support is available for platforms other than Unix. The code is standard ANSI C so making the library should not require much work on any platform.



Early Adopters and Users



- U.S. Navy Tactical Advanced Signal Processor Common Operating Environment (TASP COE) Program
- San Diego Supercomputer Center PET Tiger Team
- Lockheed-Martin Government Electronics Systems
- Rome Laboratory High Performance Computing Facility





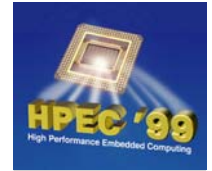
Future Work



- **VSIP L Journal of Development**
 - Repository for proposed expansions to the spec
- **Current JoD items**
 - **Image processing specification**
 - Incomplete; looking for volunteers in the area
 - **Error checking**
 - Incomplete proposal for better error handling
 - **Better view manipulation**
 - Iteration over rows/columns of a matrix
 - Views that can be bound to different blocks
- **Future Items**
 - **VSIP L C++ Binding**
 - **Parallel Processing Extensions (pVSIP L)**



Sponsors & Participants



U.S. Navy TASP COE

- Alacron
- CSPI
- Compaq/Digital
- DARPA
- Georgia Tech
- Hughes Electronic
- HRL, LLC
- Khoral Research
- LNK Corp.
- Lockheed Martin GES
- Mercury Computer Systems
- Mississippi State University
- MIT Lincoln Laboratory
- MITRE
- MPI Software Technology
- Northrop Grumman
- Ohio State University
- ORINCON
- Raytheon Systems Company
- Science & Technology Associates
- Silicon Graphics Inc.
- SKY Computers
- SPAWAR
- Texas Instruments
- Texas Memory Systems
- U.S. Air Force
- U.S. Navy and NAWC AD
- University of San Diego Supercomputing Center