VSIPL
Short Tutorial

Anthony Skjellum
MPI Software Technology, Inc.

SC2002
November 17, 2002

Overview

• Performance & Portability
• Background on the VSIPL Standard
• Code Organization
• Details and Examples
• Some Benchmarks
• Cache Considerations
Performance & Portability
Benefits/Facts of “Recent” APIs

• Abstracted application programmer interfaces (APIs) can lead to higher or lower performance
• Abstracted APIs definitely enhance portability
• The APIs described here can be implemented to enhance portability and performance of many codes
• Careful use of these APIs must be made to achieve both portability and high performance
• Portability always comes at the cost of a little performance
Why Portability is Valuable

• Standard API’s allow you to capture customers from other hardware platforms
• Allow upgrades to a system without rewriting the software
• Software cost is an important part of overall systems cost
• Software that uses portable interfaces can evolve over time more efficiently than low-level code designed for one architecture
• Overall, lower software costs, increase lifetime
What is “Performance Portability”?  

• Portability with a requirement for performance  
• Achieved with good programming skills in C or C++, as well as  
  – The right APIs for high performance math  
  – The right APIs for high performance services other than math (e.g., message passing)  
  – Efficient implementations of the APIs  
  – The right training/experience of programmers to use the APIs well  
• Performance portability is realistic and achievable with COTS architectures, such as G4/Altivec

Background On The VSIPL Standard
What is VSIPL?

- An API specification for the Vector, Signal, Image Processing operations
- Purpose: To provide a set of portable and high performance libraries for Vector, Signal, and Image Processing applications
- VSIPL standard 1.0 provides ANSI C bindings that can be implemented on a wide variety of hardware (from embedded systems to workstations)
- Multiple vendors accept VSIPL as a valid standard
- Also “endorsed” by US Department of Defense
Who developed VSIPL Standard?

• VSIP Forum - a voluntary organization comprising of representatives from industry, government, developers, and academia

• VSIPL standard was developed with close coordination with the Navy Tactical Advanced Signal Processor (TASP) program and is part of the TASP Common Operating Environment (TASP COE)

• Initial support provided by DARPA
VSIPPL Implementations

• Public domain
  – TASP VSIPPL reference implementation - Randy Judd, US Navy (multiplatform, not optimized)

• Commercial
  – CSPI (Core Lite only)
  – Mercury Computer Systems, Inc. (Core Lite only)
  – MPI Software Technology, Inc. (multiplatform)
  – Sky Computers (TASP + G4 optimizations)
  – Wildstar - Annapolis Micro Systems, Inc. (specialized hardware)
  – DNA Computing Solutions
VSIPL Features

• Portable API
• Object-based design
• Opaque objects such as blocks and views on the blocks (vectors, matrices, and tensors)
• Support for multiple floating-point and integer data types
• Development and production modes
• Public and private data arrays
• Explicit memory/algorithm hints
VSIPL Functionality

- Basic scalar operations
- Basic vector operations
- Random number generators
- Signal processing operations
  - FFT operations
  - Filtering
  - Correlation and convolution
- Linear algebra operations
  - Matrix operations
  - Linear system solution
  - Least-squares solution
- Support functions
  - Initialization and finalization
  - Object management
  - Memory management

VSIP Profiles

VSIP Profiles

• A VSIP implementation need not implement all the functions
• VSIP standard provides different profiles to support specific subsets/feature sets of the VSIP specification
  – Core Lite - subset targeted only for low memory embedded systems – same financial expense as Core
  – Core – now simply known as VSI/Pro - most of signal processing operations and matrix operations
  – Image – fast image processing targeted for military / medical / industrial use
Core Profile

- 6 data types
- 43 block support functions
- 104 view support functions
- 12 copy support functions
- 54 matrix view and copy support functions
- 47 functions for real and complex scalar support
- 147 total functions in vector and elementwise
- 45 signal processing functions
- 59 linear algebra functions
- The Core profile contains 511 functions

Core Lite Profile

- Only for low memory embedded systems
- 5 data types
- 22 block support functions
- 34 view support functions
- 1 copy support function
- 4 functions for real and complex scalar support
- 47 total functions in vector and elementwise (including basic math, trig, log, min/max and random number support)
- 14 signal processing functions
- The Core Lite profile contains 127 functions
Core vs. CoreLite

**Data Types**
Core provide one additional data type
vsip\_scalar\_mi – scalar matrix index.

**Block Support**
Core provides 21 additional block support functions

**View Support**
Core provides 70 additional view support functions

**Copy Support**
Core provides 8 additional copy support functions

**Matrix View Support**
Core provides 52 Matrix View support functions (for new data type)

**Matrix Copy Support**
Core provides 2 Matrix copy support functions (for new data type)
Core vs. CoreLite (cont.)

Scalar Functions
Core includes 47 new functions for real scalar support and complex scalar support
• 4 functions for real matrix indexing
• 39 functions for complex math and conversion
• 4 new scalar random functions

Vector Elementwise Functions
Core includes 100 total new functions in vector and elementwise
These 100 functions include
• all trig, log, max and min, max and min squared, complex math,
  and math for real and complex combinations (missing from CoreLite)
• 3 new functions for random numbers
• 18 new functions for integer vectors
• 8 boolean operations
Core vs. CoreLite (cont.)

**Signal Processing Functions**
Core includes 31 new signal processing functions
- 3 additional functions for 1D FFTs
- 10 functions for multiple 1D FFTs
- 4 window creation functions
- 2 FIR get attribute functions
- 12 convolution functions

**Linear Algebra Functions**
Core includes 59 new functions for Linear algebra manipulations
- 10 matrix product functions
- 3 matrix transpose functions
- 6 matrix product and sum functions
- 10 LU Decomposition functions
- 10 Cholesky support functions
- 14 functions for QRD
- 6 special solvers
CoreLite Altivec Optimized Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>vsip_csvmul_f</td>
<td>vsip_vlog10_f</td>
<td>vsip_vimag_f</td>
</tr>
<tr>
<td>vsip_cvadd_f</td>
<td>vsip_vlog_f</td>
<td>vsip_vfill_f</td>
</tr>
<tr>
<td>vsip_cvconj_f</td>
<td>vsip_vmag_f</td>
<td>vsip_vexp_f</td>
</tr>
<tr>
<td>vsip_cvdot_f</td>
<td>vsip_vmax_f</td>
<td>vsip_vdot_f</td>
</tr>
<tr>
<td>vsip_cvjdot_f</td>
<td>vsip_vmaxval_f</td>
<td>vsip_vdiv_f</td>
</tr>
<tr>
<td>vsip_cvjmul_f</td>
<td>vsip_vmin_f</td>
<td>vsip_vcos_f</td>
</tr>
<tr>
<td>vsip_cvmag_f</td>
<td>vsip_vminval_f</td>
<td>vsip_vcmplx_f</td>
</tr>
<tr>
<td>vsip_cvmagsq_f</td>
<td>vsip_vmul_f</td>
<td>vsip_vatan_f</td>
</tr>
<tr>
<td>vsip_cvmul_f</td>
<td>vsip_vneg_f</td>
<td>vsip_ccfftop_f</td>
</tr>
<tr>
<td>vsip_cvneg_f</td>
<td>vsip_vramp_f</td>
<td>vsip_rccfftop_f</td>
</tr>
<tr>
<td>vsip_cvsub_f</td>
<td>vsip_vreal_f</td>
<td>vsip_crfftop_f</td>
</tr>
<tr>
<td>vsip_rcvmul_f</td>
<td>vsip_vrecip_f</td>
<td>vsip_firflt_f</td>
</tr>
<tr>
<td>vsip_rscvmul_f</td>
<td>vsip_vsinf_f</td>
<td>vsip_cfirflt_f</td>
</tr>
<tr>
<td>vsip_svadd_f</td>
<td>vsip_vsq_f</td>
<td></td>
</tr>
<tr>
<td>vsip_svddiv_f</td>
<td>vsip_vsqrtf</td>
<td></td>
</tr>
<tr>
<td>vsip_svmul_f</td>
<td>vsip_vsub_f</td>
<td></td>
</tr>
<tr>
<td>vsip_vadd_f</td>
<td>vsip_vsumsqval_f</td>
<td></td>
</tr>
<tr>
<td>vsip_vatan2_f</td>
<td>vsip_vsumval_f</td>
<td></td>
</tr>
</tbody>
</table>

Core Altivec Optimized Functions

**Vector View Copy Support**
- vsip_vcopy_f_f
- vsip_vcopy_f_i
- vsip_vcopy_i_f
- vsip_vcopy_f_f

**Matrix View Copy Support**
- vsip_mcopy_f_f
- vsip_mcopy_f_f

**Float Vector Elementwise Functions**
- vsip_vacos_f
- vsip_vasin_f
- vsip_vatan_f
- vsip_vatan2_f
- vsip_vcos_f
- vsip_vexp_f
- vsip_vexp10_f
- vsip_vlog_f
- vsip_vlog10_f
- vsip_vsin_f
- vsip_vsqr_f
- vsip_vsqrt_f
- Vsip_vtan_f
- Vsip_cvconj_f
- vsip_veuler_f
- vsip_vmag_f
- vsip_vcmag_f
- vsip_vcmagsq_f

Core Altivec Optimized Functions (cont)

**Signal Processing Functions**

vsip_ccffftop_f  
vsip_ccffftip_f  
vsip_rcffftop_f  
vsip_crffftop_f  
vsip_ccfftmop_f  
vsip_ccfftmip_f  
vsip_rcfftmop_f  
vsip_crfftmop_f  
vsip_vcreate_hanning_f

vsip_vcreate_blackman_f  
vsip_vcreate_kaiser_f  
vsip_vcreate_cheby_f  
vsip_firflt_f  
vsip_cfirflt_f  
vsip_convolveld_f  
vsip_correlateld_f  
vsip_ccorrelateld_f
Core Altivec Optimized Functions (cont)

**Linear Algebra Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>vsip_vmpred_f</td>
<td>vsip_chold_f</td>
<td>vsip_gems_f</td>
</tr>
<tr>
<td>vsip_mprod_d_f</td>
<td>vsip_cchold_f</td>
<td>vsip_cgem_f</td>
</tr>
<tr>
<td>vsip_cmprod_f</td>
<td>vsip_cholal_f</td>
<td>vsip_vouter_f</td>
</tr>
<tr>
<td>vsip_cvmprod_f</td>
<td>vsip_ccholal_f</td>
<td>vsip_cvouter_f</td>
</tr>
<tr>
<td>vsip_cmprodt_f</td>
<td>vsip_chold_f</td>
<td>vsip_lud_f</td>
</tr>
<tr>
<td>vsip_mprod_dh_f</td>
<td>vsip_cchold_f</td>
<td>vsip_lusol_f</td>
</tr>
<tr>
<td>vsip_mprod_f</td>
<td>vsip_chold_create_f</td>
<td>vsip_cclusol_f</td>
</tr>
<tr>
<td>vsip_cmprod_f</td>
<td>vsip_cchold_create_f</td>
<td>vsip_qrd_f</td>
</tr>
<tr>
<td>vsip_cmprod_d_f</td>
<td>vsip_cholal_f</td>
<td>vsip_cqrdf</td>
</tr>
<tr>
<td>vsip_mtrans_f</td>
<td>vsip_cchold_destroy_f</td>
<td>vsip_qrsol_f</td>
</tr>
<tr>
<td>vsip_cmtrans_f</td>
<td>vsip_cchold_destroy_f</td>
<td>vsip_cqrdfq</td>
</tr>
<tr>
<td>vsip_cmherm_f</td>
<td>vsip_cchold_destroy_f</td>
<td>vsip_qrdprodfq</td>
</tr>
<tr>
<td>vsip_gemp_f</td>
<td>vsip_cchold_destroy_f</td>
<td>vsip_qrdsolr_f</td>
</tr>
<tr>
<td>vsip_cgemp_f</td>
<td>vsip_toepsol_f</td>
<td>vsip_cqrdsolr_f</td>
</tr>
</tbody>
</table>
VSIPL Objects, I

• A basic VSIPL object is an abstract data type called block
• A block encapsulates the data array which is the memory used for actual data storage
• The user is not allowed to directly access the block or data array, instead views are created on the block using the VSIPL support functions and operations are performed on these views
• All VSIPL numerical functions operate only on VSIPL objects
Objects vs. C memory/data structures

- Objects abstract memory and operations in VSIPL
- For memory, blocks/views/subviews allow for simpler management and specification
- In plain C, a user must use the stack or malloc/free to handle dynamic data, and use many pointers to handle complex data
- VSIPL simplifies programming through objects

User Data vs. VSIPL Data

• VSIPL operations cannot be performed on data created in user address space (public data)

• The user data must be first bound to a VSIPL block, this is known as binding the user data

• Such a bounded block can exist in an admitted or released state
User Data vs. VSIPL Data

• If the block is in admitted state then the data is in VSIPL space (private data) whereas if the block is released state then the data is in the user space
  – Admission: User space $\Rightarrow$ VSIPL space
  – Release: VSIPL space $\Rightarrow$ User space

• VSIPL operations are performed on private data (data in an admitted block only)
VSIPL Program Initialization

- Initialize VSIPL library
- Create block(s)
- Create view(s) & bind view(s) to block(s)
- Create object(s) for filter(s), FFT(s), solver(s), etc.
VSIPPL Program Body

Obtain Data
Bind (or rebind) blocks(s) to data
Admit (or readmit) block to VSIPPL data space
Operate on data using views(s)
Release block(s) to user data space
VSIPL Program Finalization

- Destroy object(s) for filter(s), FFT(s), solver(s), etc.
- Destroy view(s)
- Release and destroy block(s)
- Finalize VSIPL library
A First VSIPL Program

#include<stdio.h>
#include "vsip.h"

#define L 7 /* length */

int main(){ /* Example of Sumval */
  vsip_vview_f* data;

  vsip_init ((void *)0);

  data = vsip_vcreate_f(L,VSIP_MEM_NONE); /* Make up some data to find the sum of */ /* Compute a ramp from zero to L-1 */
  vsip_vramp_f(0.0, 1.0, data); /* And find and print its sum */
  printf("%f \n", vsip_vsumval_f(data)); /*destroy the vector views and any associated blocks */
  vsip_blockdestroy_f(vsip_vdestroy_f(data));

  vsip_finalize ((void *)0);

  return 0;
}

/* output - 21.000000 */

VSIPL Initialization

- VSIPL main header file vsip.h must be included in all programs using the VSIPL library

```c
#include "vsip.h"
```

- A VSIPL program must initialize the VSIPL library with a call to vsip_init before calling any other VSIPL function, and must call vsip_finalize before terminating.
Block and View Create

- Declare a pointer to a vector view of precision float
  \texttt{vsip_vview_f* data;}

- Create a block object and a vector view of the block
  \texttt{data = vsip_vcreate_f(vsip_length, vsip_memory_hint);} 

- \texttt{vsip_vcreate_f} function creates a block and returns a pointer to the vector view

- To make a block of type float, and length 16, and attach it to a vector view of type float, with unit stride, and length 16
  \texttt{vsip_vview_f* v = vsip_vcreate_f(16, VSIP_MEM_NONE);} 

- How do you make the block for type double ??
Ramp Function

- Computes a vector ramp starting from the specified start value and incrementing each element by the ramp step size.
  \[ \text{vsip\_vramp\_f (start, step-size, vectorview);} \]
- To initialize the vector “data” we use
  \[ \text{vsip\_vramp\_f(0.0, 1.0, data)} \]
- After the operation, our vector contains
  \[ \text{data} = (0.0, 1.0, 2.0, \ldots 6.0) \]
Vector Sum Value

• Returns the sum of the elements in a vector
  \texttt{vsip\_scalar\_f \ vsip\_sumval\_f (\ vsip\_vview\_f);}

• To sum the elements of the vector “data”
  \texttt{sum = vsip\_sumval\_f (data);}

• After the operation the value of \texttt{sum = 21}
View and Block Destroy

• Destroy vector view and return a pointer to the associated block object
  vsip_block_f vsip_vdestroy_f(vsip_vview_f)
• To destroy the vector view “data”
  vsip_block_f *block = vsip_vdestroy_f (data);
• Destroy the block object and any data allocated for it
  vsip_blockdestroy_f (vsip_block_f);
• To destroy the block “block”
  vsip_blockdestroy_f (block);
VSI/Pro Application – Vector add, I

#include<stdio.h>
#include<math.h>
#include"vsip.h"
#define L 7 /* Length of ramps */
int main()
{
    int i;
    vsip_vview_f *dataLeft, *dataRight, *dataSum;

    vsip_init ((void *)0); /* Required initialization function */
dataLeft = vsip_vcreate_f(L,VSIP_MEM_NONE);
dataRight = vsip_vcreate_f(L,VSIP_MEM_NONE);
dataSum = vsip_vcreate_f(L,VSIP_MEM_NONE);
/* Make up some data to find the magnitude of */
vsip_vramp_f(1.0, 1.0, dataLeft);
vsip_vramp_f(1.0, -2.0/(float)(L-1), dataRight);
vsip_vadd_f(dataLeft, dataRight, dataSum); /* Add the vectors */
VSI/Pro Application – Vector add, II

/* now print out the data and its sum */
for(i=0; i<L; i++)
printf("%.7f = (%.7f) + (%.7f) \n", vsip_vget_f(dataSum,i), 
    vsip_vget_f(dataLeft,i),vsip_vget_f(dataRight,i));

/* destroy the vector views and any associated blocks (important to 'clean up') */
vsip_blockdestroy_f(vsip_vdestroy_f(dataLeft));
vsip_blockdestroy_f(vsip_vdestroy_f(dataRight));
vsip_blockdestroy_f(vsip_vdestroy_f(dataSum));

vsip_finalize ((void *)0); /* Required deinitialization function */
return 0;
}
A program must initialize the VSIPL library with a call to `vsip_init` before calling any other VSIPL function. Any program that uses VSIPL and that terminates must call `vsip_finalize` before terminating.
Code Organization

- Using an object-based style, as much of the block/view creation should be performed in the initialization(create) method as possible
- Example: (NOTE: pseudocode)

  Task_init()
  
  vsip_init()
  if (data_buffer == NULL) calloc(data_buffer)       // allocate memory

  vsip_blockbind_f(data_buffer)
  vsip_cvbind_f (data_buffer) * view1
  vsip_cvbind_f(data_buffer)  *view2

  vsip_block_admit_f(data_buffer)

  vsip_ccfftop_create_f()       // fft setup calls

Code Organization

Task_execute()

read_queue(…, read_buffer)
    // don’t have apriori info for this memory block
vsip_blockbind_f(read_buffer)
vsip_vbind_f(x_view)                     // view into read_buffer
vsip_blockadmit_f(read_buffer)
vsip_blockadmit_f(scratch_buffer)  // need one for each type (float, int, etc)

Do processing (adjust views as necessary)

vsip_blockrelease_f(scratch_buffer)
vsip_vdestroy_f(x_view)
vsip_blockdestroy_f(read_buffer)
write_queue()
consume_queue()
Code Organization

Task_finalize()

    free(data_buffer)      // free memory allocated for data_buffer
    vsip_cvdestroy_f(data_buffer) *view1           // view 1 into data_buffer
    vsip_cvdestroy_f(data_buffer) *view2           // view 2 into data_buffer

    vsip_blockdestroy_f(data_buffer)

    // FFT setup calls
    vsip_fft_destroy_f()

    vsip_finalize
Data Array Layout

• User data array bound to a block has a different layout depending on the block type
• Basic types are simply a contiguous memory of the corresponding type (float, integer, boolean, and vector index)
• Matrix and Tensor index are contiguous memory of type vsip_scalar_vi. Matrix – 2 consecutive elements (row column). Tensor – 3 consecutive elements (z y x)
• Note the data for _ti and _mi in user array are not the same as vsip_scalar_mi or vsip_scalar_ti
Data Array Layout (cont.)

- Complex data can be either interleaved or split – note format is not vsip_cscalar_f
- Interleaved is contiguous memory of type vsip_scalar_f, two consecutive elements (real, imaginary)
- Split is two contiguous memory regions of equal length of type vsip_scalar_f determined when the memory is bound to the block
Data Array Layout (cont.)

- vsip_cblockbind_p  Complex Memory Block Bind
- Prototype
  vsip_cblock_f *vsip_cblockbind_f(
    const vsip_scalar_f *data1
    const vsip_scalar_f *data2
    vsip_length N
    vsip_memory_hint hint hint);
- If data2 is null – interleaved
- If data2 is not null – split (imaginary)
Performance: Split / Interleaved

• Most G4 math libraries only optimize split data complex often not offering G4 performance for interleaved data.
• VSI/Pro does extremely well with interleaved data complex.
• VSI/Pro also optimizes split data complex.
• We recommend you use interleaved data if possible for your application.
Split vs. Interleaved

Split

Re

r(0)

r(1)

r(N-1)

Im

c(0)

c(1)

c(N-1)

Interleaved

r(0)

r(1)

r(N-1)

c(0)

c(1)

c(N-1)

“transpose”
Keep Data Admitted

- Try to write inner loops so that data is admitted and processed by several VSIPL functions before being released
- Try to use interleaved data for complex vectors in current VSI/Pro versions if you admit/release often
- In 1.07+, you can also use split data with lower overheads for admit/release
- In general interleaved data works well with VSI/Pro FFTS anyway, so split data not really a benefit in most circumstances

Bit reversal “strategy”

• VSIPL didn’t offer no-bitreversal option directly, VSI/Pro does as a hint
• If possible, you can use “faster option” for FFTs where bit reversal is not done; this is a special feature of VSI/Pro
• A simple flag is used to select this feature
• For example \( y = \text{invffft}(w*\text{fft}(x)) \) can be 20% faster if \( w \) is permuted in an outer loop, and the \( \text{fft} \), \( \text{invffft} \) functions do not have to do bitreversal
• Version 1.07+ of library required to access this capability
• Not portable to non-VSI/Pro versions of VSIPL
Cache Considerations, I.

- Example: Input data is a large 2D array

- “Instinct” is to create processing that performs all the complex vector multiplication, then the FFT on the block, then all the real-complex vector multiplication

- Higher performance by putting an “outer loop” around processing such that data is processed as chunks of the input array. This results in fewer cache loads/unloads and no thrashing

- Need to balance fewer calls versus cache loading/unloading/thrashing

Cache Considerations, II

Registers

L1 cache

L2 cache

Main Memory

Block

VSIPL user abstraction (in memory hierarchy)

smaller, faster

Bigger, slower

Other Performance Tips

• VSI/Pro internally handles and manages L1, L2 cache for efficient performance… memory used for objects should be cachable
• Verify Altivec is enabled (must be handled by vendors BSP)
• Turn off JAVA (IEEE mode) – Vector Status and Control Register (VSCR) (see Altivec PEM)
• Note: These issues all addressed correctly for most VxWorks BSPs
FFT Benchmarks
FFT Benchmarks

These benchmarks measure the speed of VSI/Pro’s Fast Fourier Transforms (FFTs)

<table>
<thead>
<tr>
<th></th>
<th>Out-of-place</th>
<th>rcfftop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real to Complex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complex to Real</td>
<td></td>
<td>crfftop</td>
</tr>
<tr>
<td>Complex to Complex</td>
<td>Out-of-place</td>
<td>ccfftop</td>
</tr>
<tr>
<td>Complex to Complex</td>
<td>In-place</td>
<td>ccfftip</td>
</tr>
</tbody>
</table>

Measured performance is very close to theoretical processor performance for certain sizes. The processor speed and L2 CACHE size is shown on each slide for the different platforms.

More Benchmarks Details, I

/*-----------------CCFFTOP-----------------------------------------------*/

cc = vsip_cvcreate_f(N, VSIP_MEM_NONE);  
dd = vsip_cvcreate_f(N, VSIP_MEM_NONE);

for(j = 0; j < N; j++) {
    z = vsip_cmplx_f(j, -j);  
    vsip_cvput_f(cc, j, z);  
    vsip_cvput_f(dd, j, z);
}

fftccop = vsip_ccfftop_create_f(N, 1.0, VSIP_FFT_FWD, 1, VSIP_ALG_TIME);
cycles = 1;
do {
    btime = vsipro_get_time();

    for(i = 0; i < cycles; i++)
        vsip_ccfftop_f(fftccop, cc, dd);

    etime = vsipro_get_time();
    c = vsipro_time_to_msec(vsipro_time_diff(etime, btime));
    if ((c < VSIPRO_UTIL_TIME_MIN) cycles <<= 1;
}   while (c < VSIPRO_UTIL_TIME_MIN);
More Benchmarks Details, II

c /= cycles;
    printf("\nBenchmark on N=%ld with %ld cycles", N, cycles);
    printf("\n ccfftop - %10.6f ms", (double)c);
    vsip_fft_destroy_f(fftccop);
    vsip_cvdestroy_f(cc);
    vsip_cvdestroy_f(dd);
Benchmarks

• Benchmark spreadsheets available for FFT and FIR
• Benchmarking programs available (signal processing)
• Benchmarking of VSIPL overhead (admit/release etc)
• Note processor size, L2 cache size, Mflops 1d: \( \frac{5*n*\log_2(n)}{\text{time}[\mu\text{sec}]} \), Mflops 2d: \( \frac{5*m*n*\log_2(m*n)}{\text{time}[\mu\text{sec}]} \)
Building on LynxOS

向 LynxOS 建立

########################################################################
#
#                    A sample makefile for VSIPRO demo
#                                                               
# ########################################################################

# Makefile for LynxOS

GCC = gcc
LD  = gcc
CFLAGS = -O2 -Wall -pedantic -DVSIPRO_COMPILER_GCC_LYNXOS

FILE_NAME = benchfft_f

USER_LIB = ../../user/lib
UTIL_LIB = ../../util/lib

# util library
UTIL_LIB_NAME = vsipro_util_LynxOS

# LynxOS
LIB_NAME = vsipro_clp_LynxOS

USER_INCLUDE = ../../user/include
UTIL_INCLUDE = ../../util/include

LINK_DIRS = -L$(USER_LIB) -L$(UTIL_LIB)

# link libraries
LINK_OPTIONS = -l$(LIB_NAME) -lstdc++ -l$(UTIL_LIB_NAME)

COMPILE_OPTIONS = -c $(CFLAGS) -I$(USER_INCLUDE) -I$(UTIL_INCLUDE) -I../../cppapi/include
LINK = $(LD) $(LINK_DIRS)

Running on LynxOS

Welcome to LynxOS
( 3.0.1 FCS version: 110298 - G )

Benchmark on N=1024 with 65536 cycles
---------------------------------------------
rcffttop    -  0.027731 ms
crffttop    -  0.063097 ms

Benchmark on N=1024 with 32768 cycles
---------------------------------------------
ccffttop    -  0.042031 ms
ccefttip    -  0.041043 ms
ccfft tip 1.0 -  0.041197 ms
cceft tip 1/N -  0.042012 ms

## LynxOS Benchmarks

### LynxOS FFT performance, G4 (7400) 400MHz 2M L2 cache

<table>
<thead>
<tr>
<th>sizelog2</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>256</td>
<td>1024</td>
<td>4096</td>
<td>16384</td>
</tr>
<tr>
<td>time, musec</td>
<td>7.67</td>
<td>32.04</td>
<td>205.08</td>
<td>1,259.77</td>
</tr>
<tr>
<td>vsip_ccfftip_f</td>
<td>6.68</td>
<td>29.14</td>
<td>240.48</td>
<td>1,035.16</td>
</tr>
<tr>
<td>vsip_ccfftop_f</td>
<td>4.46</td>
<td>17.70</td>
<td>99.49</td>
<td>649.41</td>
</tr>
<tr>
<td>vsip_crfftop_f</td>
<td>5.34</td>
<td>21.06</td>
<td>114.75</td>
<td>693.36</td>
</tr>
</tbody>
</table>

### benchFFT, Mflops

| vsip_ccfftip_f | 1,335.42 | 1,597.85 | 1,198.37 | 910.39 |
| vsip_ccfftop_f | 1,533.85 | 1,756.79 | 1,021.96 | 1,107.93 |
| vsip_crfftop_f + | 1,044.47 | 1,321.05 | 1,147.16 | 854.11 |

Building on MacOS
Running on MacOS

# Mac OS Benchmarks

MacOS FFT performance, G4 350MHz 1M L2 CACHE

<table>
<thead>
<tr>
<th>size log2</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>256</td>
<td>1024</td>
<td>4096</td>
<td>16384</td>
</tr>
<tr>
<td>time, msec</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vs ip_ccfftip_f</td>
<td>8.96</td>
<td>37.90</td>
<td>220.18</td>
<td>1,434.78</td>
</tr>
<tr>
<td>vs ip_ccfftop_f</td>
<td>7.68</td>
<td>34.10</td>
<td>283.44</td>
<td>1,309.21</td>
</tr>
<tr>
<td>vs ip_rcfftop_f</td>
<td>5.30</td>
<td>20.93</td>
<td>119.62</td>
<td>798.59</td>
</tr>
<tr>
<td>vs ip_crfftop_f</td>
<td>6.26</td>
<td>25.20</td>
<td>134.59</td>
<td>821.64</td>
</tr>
</tbody>
</table>

benchFFT, Mflops

| vs ip_ccfftip_f | 1,143.50 | 1,351.03 | 1,116.16 | 799.34 |
| vs ip_ccfftop_f | 1,333.33 | 1,501.42 | 867.07 | 876.01 |
| vs ip_rcfftop_f+ | 886.04 | 1,109.83 | 966.74 | 707.85 |

# FFT Numbers for MacOS X

MacOS X FFT performance, G4 (7410) 500MHz (Titanium Powerbook) 1M L2 cache

<table>
<thead>
<tr>
<th>sizelog2</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1024</td>
<td>2048</td>
<td>4096</td>
<td>8192</td>
<td>16384</td>
<td>32768</td>
<td>65536</td>
<td>131072</td>
</tr>
<tr>
<td>time, msec</td>
<td>vsip_ccfftip_f</td>
<td>3.30</td>
<td>6.79</td>
<td>14.64</td>
<td>28.59</td>
<td>64.33</td>
<td>187.12</td>
<td>467.48</td>
<td>1,105.21</td>
<td>2,976.00</td>
<td>6,349.06</td>
</tr>
<tr>
<td></td>
<td>vsip_ccfftop_f</td>
<td>3.46</td>
<td>6.41</td>
<td>13.27</td>
<td>31.03</td>
<td>73.91</td>
<td>228.21</td>
<td>513.65</td>
<td>1,118.74</td>
<td>2,991.22</td>
<td>9,615.28</td>
</tr>
<tr>
<td></td>
<td>vsip_rcfftop_f</td>
<td>2.46</td>
<td>4.84</td>
<td>8.74</td>
<td>17.98</td>
<td>42.03</td>
<td>98.25</td>
<td>283.25</td>
<td>638.87</td>
<td>1,397.45</td>
<td>4,195.53</td>
</tr>
<tr>
<td></td>
<td>vsip_crfftop_f</td>
<td>3.02</td>
<td>5.24</td>
<td>10.33</td>
<td>21.46</td>
<td>44.80</td>
<td>123.08</td>
<td>309.04</td>
<td>695.69</td>
<td>1,627.62</td>
<td>4,860.02</td>
</tr>
<tr>
<td>benchFFT, Mflops</td>
<td>vsip_ccfftip_f</td>
<td>1,357.58</td>
<td>1,508.10</td>
<td>1,573.77</td>
<td>1,790.84</td>
<td>1,750.97</td>
<td>1,313.38</td>
<td>1,139.04</td>
<td>1,037.70</td>
<td>825.81</td>
<td>825.77</td>
</tr>
<tr>
<td></td>
<td>vsip_ccfftop_f</td>
<td>1,294.80</td>
<td>1,597.50</td>
<td>1,736.25</td>
<td>1,850.02</td>
<td>1,524.02</td>
<td>1,076.90</td>
<td>1,025.15</td>
<td>724.69</td>
<td>545.27</td>
<td>412.38</td>
</tr>
<tr>
<td></td>
<td>vsip_rcfftop_f</td>
<td>1.06</td>
<td>1.91</td>
<td>3.53</td>
<td>7.39</td>
<td>15.84</td>
<td>35.24</td>
<td>94.05</td>
<td>231.41</td>
<td>616.62</td>
<td>1806.87</td>
</tr>
<tr>
<td></td>
<td>vsip_crfftop_f</td>
<td>1.34</td>
<td>2.51</td>
<td>4.46</td>
<td>9.41</td>
<td>21.09</td>
<td>45.92</td>
<td>107.87</td>
<td>254.33</td>
<td>692.25</td>
<td>1,917.00</td>
</tr>
</tbody>
</table>

MacOS X FFT performance, G4 (7455) 933MHz 2M L2 cache

<table>
<thead>
<tr>
<th>sizelog2</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1024</td>
<td>2048</td>
<td>4096</td>
<td>8192</td>
<td>16384</td>
<td>32768</td>
<td>65536</td>
<td>131072</td>
</tr>
<tr>
<td>time, msec</td>
<td>vsip_ccfftip_f</td>
<td>1.93</td>
<td>3.44</td>
<td>7.33</td>
<td>16.49</td>
<td>32.34</td>
<td>74.25</td>
<td>177.94</td>
<td>413.13</td>
<td>1,192.48</td>
<td>3,108.42</td>
</tr>
<tr>
<td></td>
<td>vsip_ccfftop_f</td>
<td>1.48</td>
<td>2.96</td>
<td>6.33</td>
<td>13.04</td>
<td>30.67</td>
<td>79.07</td>
<td>181.90</td>
<td>473.88</td>
<td>1,481.82</td>
<td>3,472.03</td>
</tr>
<tr>
<td></td>
<td>vsip_rcfftop_f</td>
<td>1.06</td>
<td>1.91</td>
<td>3.53</td>
<td>7.39</td>
<td>15.84</td>
<td>35.24</td>
<td>94.05</td>
<td>231.41</td>
<td>616.62</td>
<td>1806.87</td>
</tr>
<tr>
<td></td>
<td>vsip_crfftop_f</td>
<td>1.34</td>
<td>2.51</td>
<td>4.46</td>
<td>9.41</td>
<td>21.09</td>
<td>45.92</td>
<td>107.87</td>
<td>254.33</td>
<td>692.25</td>
<td>1,917.00</td>
</tr>
<tr>
<td>benchFFT, Mflops</td>
<td>vsip_ccfftip_f</td>
<td>2,326.06</td>
<td>2,977.81</td>
<td>3,143.25</td>
<td>3,104.54</td>
<td>3,483.32</td>
<td>3,309.99</td>
<td>2,992.45</td>
<td>2,776.06</td>
<td>2,060.92</td>
<td>1,688.87</td>
</tr>
<tr>
<td></td>
<td>vsip_ccfftop_f</td>
<td>3,035.23</td>
<td>3,457.12</td>
<td>3,642.69</td>
<td>3,926.38</td>
<td>3,672.52</td>
<td>3,108.17</td>
<td>2,927.40</td>
<td>2,420.17</td>
<td>1,658.50</td>
<td>1,510.03</td>
</tr>
<tr>
<td></td>
<td>vsip_rcfftop_f</td>
<td>1,868.22</td>
<td>2,316.74</td>
<td>2,877.48</td>
<td>3,047.98</td>
<td>3,050.59</td>
<td>3,027.94</td>
<td>2,637.11</td>
<td>2,361.12</td>
<td>1,877.65</td>
<td>1,407.91</td>
</tr>
</tbody>
</table>

Building on VxWorks
Running on VxWorks

# VxWorks Benchmarks

VxWorks FFT performance, G4 350MHz 1M L2 CACHE

<table>
<thead>
<tr>
<th>size</th>
<th>log2 8</th>
<th>10</th>
<th>12</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>time, usec</td>
<td>256</td>
<td>1024</td>
<td>4096</td>
<td>16384</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>benchmarkFFT, Mflops</th>
</tr>
</thead>
<tbody>
<tr>
<td>vs ip_ccfftip_f</td>
</tr>
<tr>
<td>vs ip_ccfftop_f</td>
</tr>
<tr>
<td>vs ip_rcfftop_f+</td>
</tr>
<tr>
<td>vs ip_crfftop_f</td>
</tr>
<tr>
<td>vs ip_crfftop_f+</td>
</tr>
</tbody>
</table>
### Linux PPC Benchmarks

**Linux PPC FFT performance, G4 350MHz 1M L2 CACHE**

<table>
<thead>
<tr>
<th>size log2</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>256</td>
<td>1024</td>
<td>4096</td>
<td>16384</td>
</tr>
<tr>
<td>time, ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vs ip_cfftip_f</td>
<td>8.83</td>
<td>37.06</td>
<td>250.78</td>
<td>1,506.52</td>
</tr>
<tr>
<td>vs ip_cfftop_f</td>
<td>7.63</td>
<td>33.34</td>
<td>278.11</td>
<td>1,307.53</td>
</tr>
<tr>
<td>vs ip_rcfftop_f</td>
<td>5.15</td>
<td>20.46</td>
<td>116.30</td>
<td>753.75</td>
</tr>
<tr>
<td>vs ip_crfhtop_f</td>
<td>6.17</td>
<td>24.19</td>
<td>132.72</td>
<td>820.93</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>benchFFT, Mflops</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>vs ip_cfftip_f</td>
<td>1,160.10</td>
<td>1,381.62</td>
<td>979.98</td>
<td>761.28</td>
</tr>
<tr>
<td>vs ip_cfftop_f</td>
<td>1,341.80</td>
<td>1,535.58</td>
<td>883.67</td>
<td>877.13</td>
</tr>
<tr>
<td>vs ip_rcfftop_f+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vs ip_crfhtop_f</td>
<td>904.94</td>
<td>1,146.64</td>
<td>986.89</td>
<td>728.33</td>
</tr>
</tbody>
</table>
Linux PPC 2d_FFT Benchmarks
where M = 64

<table>
<thead>
<tr>
<th></th>
<th>Mlog2</th>
<th>M</th>
<th>Nlog2</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>64</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>64</td>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>64</td>
<td>8</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>64</td>
<td>9</td>
<td>512</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>time, msec</th>
<th>vsip_ccfft2dip_f</th>
<th>39.06</th>
<th>234.38</th>
<th>1,484.38</th>
<th>3,125.00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>vsip_ccfft2dop_f</td>
<td>43.95</td>
<td>253.91</td>
<td>1,328.13</td>
<td>2,968.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vsip_rcfft2dop_f</td>
<td>34.18</td>
<td>146.48</td>
<td>625.00</td>
<td>1,406.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vsip_crfft2dop_f</td>
<td>34.18</td>
<td>175.78</td>
<td>664.06</td>
<td>1,562.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>benchFFT, Mflops</th>
<th>vsip_ccfft2dip_f</th>
<th>1,310.74</th>
<th>1,048.58</th>
<th>772.63</th>
<th>786.43</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>vsip_ccfft2dop_f</td>
<td>1,165.09</td>
<td>967.92</td>
<td>863.53</td>
<td>827.82</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vsip_rcfft2dop_f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>vsip_crfft2dop_f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>vsip_crfft2dop_f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Linux PPC 2d_FFT Benchmarks
where M = 256

LinuxPPC 2D FFT performance, G4 (7400) 350MHz 1M L2 cache where M = 256

<table>
<thead>
<tr>
<th>Mlog2</th>
<th>8</th>
<th>8</th>
<th>8</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>Nlog2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>N</td>
<td>16</td>
<td>64</td>
<td>256</td>
<td>512</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>time, mussec</th>
<th>vsip_ccfft2dip_f</th>
<th>vsip_ccfft2dop_f</th>
<th>vsip_rcfft2dop_f</th>
<th>vsip_crfft2dop_f</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>234.38</td>
<td>1,328.13</td>
<td>6,875.00</td>
<td>21,250.00</td>
</tr>
<tr>
<td></td>
<td>273.44</td>
<td>1,250.00</td>
<td>6,875.00</td>
<td>22,500.00</td>
</tr>
<tr>
<td></td>
<td>195.31</td>
<td>664.06</td>
<td>3,125.00</td>
<td>6,875.00</td>
</tr>
<tr>
<td></td>
<td>195.31</td>
<td>781.25</td>
<td>3,437.50</td>
<td>10,625.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>benchFFT, Mflops</th>
<th>vsip_ccfft2dip_f</th>
<th>vsip_ccfft2dop_f</th>
<th>vsip_rcfft2dop_f + vsip_crfft2dop_f</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1,048.58</td>
<td>863.53</td>
<td>762.60</td>
</tr>
<tr>
<td></td>
<td>898.78</td>
<td>917.50</td>
<td>762.60</td>
</tr>
<tr>
<td></td>
<td>629.15</td>
<td>793.52</td>
<td>798.92</td>
</tr>
</tbody>
</table>

VSIPL Image Processing Preview

• Another “Profile” can be constructed
• Image image processing functions (150+ functions)
• CoreLite “Profile”
• A small part of VSIPL Core API functionality needed to manipulate 2D matrices and images
• Not fully standardized by the Forum yet
• One commercial implementation thus far

VSIPL Image Functionality

- Histogram Operations
- Convolution
- Diff / Edge Detection
- Image Pad
- Arithmetic Operations
- Logical Functions

- Morphological Operations
- Image Resize
- Object Functionality (e.g., bind/rebind)
- Conversion
Resources

• http://www.vsipl.org (VSIPL)
• http://www.netlib.org (LAPACK, BLAS)
• http://www.altivec.org (Altivec Language)
Summary of VSIPL

• VSIPL is an effective standard for signal and image processing
• Uses object-based concepts for performance/portability
• Easy to use/learn
• High performance implementations exist
• Signal Processing Fully Standardized
• Image Processing not yet Fully Standardized
• Both reference implementations and commercial implementations are available
• Growing acceptance, in use worldwide, including in defense programs, in industrial setting, and in academia