Sourcery VSIPL++ for Cell/B.E.

HPEC
Sep 20, 2007

Jules Bergmann, Mark Mitchell, Don McCoy, Stefan Seefeld, Assem Salama - CodeSourcery, Inc
Fred Christensen - IBM
Rick Pancoast, Tom Steck - Lockheed Martin MS2

jules@codesourcery.com
888-776-0262 x705
Sourcery VSIPL++: Signal & Image-Processing Library

- **Comprehensive Functionality**
  - Signal-Processing: FFTs, convolutions, correlations, etc.
  - Solvers: QR, LU, Cholesky, etc.
  - Linear Algebra: matrix multiplication, Hermitians, etc.
  - Support for multi-processor computation

- **Simple C++ API**
  - No MPI programming required
  - No SPE programming required
  - No special tools required
  - Easy to port code across systems
  - Easy to compare performance across vendors/architectures

- **Performance**
  - Automatically fuses computations to run on SPEs
  - Single digit % “abstraction penalty” for simple primitives

- **Interoperability**
  - Leverages the vendor software stacks
  - Implements the open-standard VSIPL++ API
DoD Motivation for VSIPL++: Faster, Better, Cheaper

- **Performance:**
  - Write fast code for particular CPUs once, then use it again and again
  - Let computers perform complex optimizations

- **Portability:**
  - Reuse code on multiple systems:
    - supercomputers
    - workstations
    - embedded systems

- **Productivity:**
  - Write new code faster
  - Repurpose existing code
  - Allow experimentation

---

**Issues with Current HPEC Development**
Inadequacy of Software Practices & Standards

- High Performance Embedded Computing pervasive through DoD applications
  - Airborne Radar Insertion program
    - 85% software rewrite for each hardware platform
  - Missile common processor
    - Processor board costs < $100k
    - Software development costs > $100M
  - Torpedo upgrade
    - Two software re-writes required after changes in hardware design

**System Development/Acquisition Stages**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Years</td>
<td>4 Years</td>
<td>4 Years</td>
<td>4 Years</td>
<td>4 Years</td>
<td>4 Years</td>
</tr>
</tbody>
</table>

**Today – Embedded Software Is:**
- Not portable
- Not scalable
- Difficult to develop
- Expensive to maintain

---

**COTS Benefits for Software**
**Cell / B.E. Architecture**

**Power Processing Element**
- 64-bit general purpose RISC
- 2-way hardware multithreaded
- L1 Cache: 32KB I / 32KB D
- L2 Cache: 512KB combined
- VMX SIMD ISA
- 3.2 GHz

**Synergistic Processor Elements**
- SIMD Substrate
  - 128-bit wide SIMD Units
  - 128-word register file
  - 25.6 GF/s peak @ 3.2 GHz
- 256 KB Local Store
- DMA Controller

**200+ GF/s Peak Performance**

**25.6 GB/s Memory Bandwidth**

**20 GB/s Coherent 5 Gbps IO**
Cell / B.E. Programming Challenges

Usual Challenges
- SIMD Vectorization
- Instruction-Level Parallelism
  - Pipeline latency
  - Dual issue
- Memory Hierarchy
- Compute/IO

New Multi-core challenges
- Exploit SPE level parallelism
  - Algorithm Partitioning
- Manage explicit communication
  - Comp/Comm overlap
- Manage limited SPE memory

Complex Programming Model

25.6 GB/s Memory Bandwidth
20 GB/s Coherent 5 Gbps IO
Cell/B.E. SIP Application Development Models

- **Low-Level / Direct Access**
  - Write SPE and MPI code manually
  - Explicitly manage DMAs, double-buffering, etc.
  - Pros: theoretically optimal performance
  - Cons: challenging, time-consuming, not portable
  - Programming at this level is like programming in assembly language

- **Vendor Software Stack**
  - Write SPE and MPI code manually
  - Use SDK, ALF to manage DMAs and buffering
  - Pros: simpler programming model
  - Cons: not optimized for SIP, not portable

- **Sourcery VSIPL++**
  - Use high-level API to express algorithm
  - Let Sourcery VSIPL++ manage SDK, ALF, MPI, SPEs
  - Pros: simplest programming model, portable
  - Cons: may not provide maximum performance, cover all possible use cases
Views / Blocks
• Separates concerns of data’s logical view from its physical layout
  – Split/interleaved, dimension ordering, parallel distribution
• Initial functional development independent of subsequent optimization

Expression Templates
• Library has visibility to sequence of operations
• Greater optimization potential
• Operation Fusion – Locality

Dispatch Engine
• Flexible, low-overhead dispatch of operations to computation
• Based on run-time and compile-time attributes

VSIPL++ API and Sourcery VSIPL++ Implementation
Provide Powerful Abstractions and Tools for Cell/B.E.
Fast Convolution

typedef complex<float> T;
Vector<T> weights(size);
Matrix<T> data(rows, size);

Fftm<T, T, row, fft_fwd>
  fwd(Domain<2>(rows, size), 1.);
Fftm<T, T, row, fft_inv>
  inv(Domain<2>(rows, size), 1./size);

fft_ip<fft_fwd>(weights);

data = inv(vmmul<row>(weights, fwd(data)));
Sourcery VSIPL++ manages the SPEs
- Recognizes VSIPL++ routines suitable for SPEs
- Uses IBM SDK (ALF) to control SPEs

Fast Convolution

```c
typedef complex<float> T;
Vector<T> weights(size);
Matrix<T> data(rows, size);

Fftm<T, T, row, fft_fwd>
  fwd(Domain<2>(rows, size), 1.);
Fftm<T, T, row, fft_inv>
  inv(Domain<2>(rows, size), 1./size);

fft_ip<fft_fwd>(weights);
data = inv(vmmul<row>(weights, fwd(data)));
```
VSIP++ Model for Cell/B.E.

PPE
- User Application
- Sourcery VSIP++
- IBM SDK (multi-core)

SPE N
SPE 1
- Fused Kernel
  - FFT⁻¹ → vmul → FFT⁻¹

Memory (RDRAM)

Compute kernels run on SPEs

data = inv(vmmul<row>(weights, fwd(data)));
VSIP++ Model for Cell/B.E.

**PPE**

- User Application
- Sourcery VSIP++
- IBM SDK (multi-core)

**SPEs**

- **SPE 1**
  - Fused Kernel: FFT⁻¹ → vmul → FFT⁻¹
  - Local Store: buffer #1
  - Memory (RDRAM): data

**SPEs manage streaming**
- DMA to/from memory
- Double buffering
- Computation/Communication overlap
Sourcery VSIPL++ can utilize managed processors.
Fast convolution:
For each pulse: \( \text{out} = \text{InvFFT}(\text{weights} \times \text{FwdFFT}(\text{in})) \)

In VSIPL++, this takes 7 lines (just 1 for computation):

```cpp
typedef complex<float> T;
Vector<T> weights(size);
Matrix<T> data(rows, size);

Fftm<T, T, row, fft_fwd>
    fwd(Domain<2>(rows, size), 1.);
Fftm<T, T, row, fft_inv>
    inv(Domain<2>(rows, size), 1./size);

fft_ip<fft_fwd>(weights);
data = inv(vmmul<row>(weights, fwd(data)));
```

No system/architecture specific statements required
Fast Convolution

FFT⁻¹ → vmul → FFT⁻¹

Size

Rows

Fast Convolution

Size

Rows
Cell/B.E. Fast Convolution

Data is partitioned across SPEs

- Fused kernel runs on SPEs
- Data processed row at a time
- Double buffered DMA
Performance

VSIP++ fast convolution sustains 80+ GFLOP/s (40% of SPE peak)

At 4096 rows of 2048 points
- 83 GFLOP/s (40% of peak)
- ~10 GB/s bandwidth

Performance Headroom
- FFT dominates computation.
- BW available: 20 GB/s demonstrated.

Memory to memory measurement
Portability

VSIPL++ fast convolution runs *unchanged* on Xeon and PowerPC

<table>
<thead>
<tr>
<th>Platform</th>
<th># proc</th>
<th>GFLOP/s</th>
<th>Util</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6 GHz Xeon</td>
<td>1</td>
<td>6.0</td>
<td>41.8%</td>
</tr>
<tr>
<td>(Using Intel IPP)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 GHz PowerPC 7447A</td>
<td>1</td>
<td>3.7</td>
<td>46.2%</td>
</tr>
<tr>
<td>(Using Mercury SAL)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 GHz PowerPC 970FX</td>
<td>1</td>
<td>6.6</td>
<td>41.2%</td>
</tr>
<tr>
<td>(Using FFTW 3)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Portable* High Sustained Performance
Using multiple processors requires minor changes to data structures (blue):

```cpp
typedef complex<float> T;
typedef Dense<2, T, row2_major, Map<> > data_block_type;
typedef Dense<1, T, row1_major, Global_map<1> > weights_block_type;
Map<> map(num_processors());
Vector<T, weights_block_type> weights(size);
Matrix<T, data_block_type> data(rows, size, map);
```

No changes to operations or computation:

```cpp
Fftm<T, T, row, fft_fwd> fwd(Domain<2>(rows, size), 1.);
Fftm<T, T, row, fft_inv> inv(Domain<2>(rows, size), 1./size);

fft_ip<fwd_fft>(weights);

data = inv(vmmul<row>(weights, fwd(data)));
```

Expressing Data-Parallelism Straight-Forward
Parallelism

VSIPL++ fast convolution can take advantage of multiple processors

Using 4 Cell/B.E.s
- Sustains 320 GFLOP/s

Speedup (expect linear):
- Fixed problem size: 3.6x speedup.
- Scaled problem size: 3.9x speedup.
Trade-Space Exploration

For coherently connected Cell/B.E.s, what is faster?
• 1 process - 1 PPE with 16 SPEs
• 2 processes - 2 PPEs with 8 SPEs each

Just try it!

Using 2 PPEs outperforms:
• Greater memory bandwidth
• Coherent interconnect bottleneck

Easy to Explore Implementation Trade-offs
Advantages of Sourcery VSIPL++ for Cell/B.E.

- **Improves out-of-box experience**
  - Code runs unchanged on Cell/B.E. with good performance
  - Programmer retains ability to tune for maximum performance

- **Reduces software development costs**
  - Fewer lines of code
  - Very little Cell-specific code
  - No direct SPE programming
  - Trade-space exploration

- **Portability**
  - Software can be easily migrated between Cell/B.E. and other systems

**Performance, Productivity, Portability, Parallelism!**
Availability

Sourcery VSIPL++ is available today
• 1.3 for GNU/Linux, Mercury Power and Windows systems
• Technology preview for Cell/B.E.

For more information and download:
• Visit our website: www.codesourcery.com/vsiplplusplus

Join our mailing list:
• Announcements: vsipl++-announce@codesourcery.com
Sourcery VSIPL++ for Cell/B.E.

HPEC
Sep 20, 2007

Jules Bergmann, Mark Mitchell, Don McCoy, Stefan Seefeld, Assem Salama - CodeSourcery, Inc
Fred Christensen - IBM
Rick Pancoast, Tom Steck - Lockheed Martin MS2

jules@codesourcery.com
888-776-0262 x705
### Status

- **IBM Teaming Agreement**
  - VSIPL++ Proof of Concept (Complete): Optimize fast convolution (FFT, vector-multiply)
  - Cell Math Library

- **Current Performance:**
  - 1 Cell: 83 GFLOPS (~40% utilization)
  - 4 Cells (2 blades): 318 GFLOPS (~39% utilization)

- **Completely Portable:**
  - User needs no knowledge of Cell/B.E. (SPEs, etc.)
  - Porting from another system is just recompilation

### Model

- **Users program the PPE**
  - User code does not directly run on SPEs, do DMAs, etc.

- **Sourcery VSIPL++ manages the SPEs**
  - Streaming kernel accelerator
  - Translates VSIPL++ API calls into SPE routines
  - Manages DMAs, double-buffering, etc.

- **Sourcery VSIPL++ manages multi-processors**
  - Uses MPI to communicate data between processors

- **Leverages IBM Software Stack**

---

**Sourcery VSIPL++ delivers the performance of Cell/B.E. in a simple, portable, high-level API.**
Productivity

Compute BLAS zherk:

\[ C \leftarrow \alpha A \text{conj}(A)^t + \beta C \]

**VSIPL**

\[
A = \text{vsip\_cmcreate\_d}(10, 15, \text{VSIP\_ROW}, \text{MEM\_NONE});
C = \text{vsip\_cmcreate\_d}(10, 10, \text{VSIP\_ROW}, \text{MEM\_NONE});
tmp = \text{vsip\_cmcreate\_d}(10, 10, \text{VSIP\_ROW}, \text{MEM\_NONE});
\text{vsip\_cmprodh\_d}(A, A, tmp);
\text{vsip\_rscmmul\_d}(\alpha, tmp, tmp);
\text{vsip\_rscmmul\_d}(\beta, C, C);
\text{vsip\_cmadd\_d}(tmp, C, C);
\text{vsip\_cblockdestroy}(\text{vsip\_cmdestroy\_d}(tmp));
\text{vsip\_cblockdestroy}(\text{vsip\_cmdestroy\_d}(C));
\text{vsip\_cblockdestroy}(\text{vsip\_cmdestroy\_d}(A));
\]

**Sourcery VSIPL++**

Matrix<complex<double>> > A(10,15);
Matrix<complex<double>> > C(10,10);
C = alpha * prodh(A,A) + beta * C;

**Advantages**

- 70% fewer lines of code
- No explicit memory management
- Better optimization opportunities
Productivity

Vector Threshold

\[ Z \leftarrow (A > B) \ ? \ A : 0 \]

<table>
<thead>
<tr>
<th>SAL</th>
<th>Sourcery VSIPPL++</th>
</tr>
</thead>
<tbody>
<tr>
<td>float* A[size];</td>
<td></td>
</tr>
<tr>
<td>float* B[size];</td>
<td></td>
</tr>
<tr>
<td>float* Z[size];</td>
<td></td>
</tr>
<tr>
<td>lvgtx(A, 1, B, 1, Z, 1, size, 0);</td>
<td></td>
</tr>
<tr>
<td>vmulx(Z, 1, A, 1, Z, 1, size, 0);</td>
<td></td>
</tr>
<tr>
<td>Vector&lt;float&gt; A(size);</td>
<td></td>
</tr>
<tr>
<td>Vector&lt;float&gt; B(size);</td>
<td></td>
</tr>
<tr>
<td>Vector&lt;float&gt; C(size);</td>
<td></td>
</tr>
<tr>
<td>C = ite(A &gt; B, A, 0.0);</td>
<td></td>
</tr>
</tbody>
</table>

Advantages

✓ Not limited to API
✓ Fewer lines of code
✓ Better performance
  • Better cache locality
Performance

Fused multiply-add (aka non-uniformity correction):

\[ \text{out} = \text{gain} \times \text{img} + \text{offset}; \]

Expression Templates
- Represent expression as parse tree

```
=  
/ \  
/   
/    
out +   
/     
/      
/       
*       
/     
/      
/       
gain + offset
```
- Library can examine, manipulate, evaluate parse tree at compile-time

Operation Fusion
- Fuse multiple operations into single loop:
  ```c
  for (i=0; i<rows*cols; ++i)
    out[i] = gain[i]*img[i] + offset[i];
  ```
- Possibly using Altivec:
  ```c
  for (i=0; i<rows*cols; ++i)
    out = vec_madd(gain, img, offset);
    out+=4; gain+=4; img+=4; offset+=4;
  ```

Math Library Interface
- Fuse operations into vendor library call(s):
  ```c
  vma(gain,1,in,1,offset,1,out,1,size);
  ```
- Single digit overheads ~2%

Sophisticated Implementation Techniques for High-Performance
Performance

Fused Multiply-Add (NUC)

For 1 GHz PPC 7447A at 2048 points:
- VSIPL++ (red) 0.971 GFLOP/s
- Vendor (blue) 0.986 GFLOP/s
VSIPL++: 1.5% overhead

Vector Threshold

For 1 GHz PPC 7447A at 2048 points:
- VSIPL++ (red) 0.591 GPT/s
- Vendor (blue) 0.385 GPT/s
VSIPL++: 53% improvement w/fused Ops

Vendor Library Performance or Better
Portability

C++ API
- Developers use existing compilers, debuggers, etc.
- No special tools required
- No new programming languages to learn

CPUs
- IA32, EM64T, AMD64
- Power
- Cell/B.E.
- SPARC

Compilers
- Sourcery G++
- GNU
- Green Hills
- Intel

Advantages
- ✓ Compare multiple platforms
- ✓ Develop where convenient
- ✓ Deploy in multiple environments
Parallelism

Sourcery VSIPL++

- **Simple Model**
  - User specifies data distribution
  - VSIPL++ manages data movement

- **Serial/Parallel Portability**
  - Same algorithms run in serial and in parallel
  - Specify data distributions …
  - … recompile …
  - … run!

Advantages

- No MPI, PAS, etc. code required
- Same code runs on:
  - Multiprocessor workstations
  - GNU/Linux clusters
  - Embedded multiprocessors
- Experimenting with data distributions is easy