Taking the HPEC Challenge with VSIPL++

Don McCoy, Stefan Seefeld, Mark Mitchell, Jules Bergmann
CodeSourcery, Inc.
jules@codesourcery.com

Introduction

CodeSourcery, Inc has ported the HPEC Challenge Kernel Benchmarks to the VSIPL++ API. In this presentation we will describe these benchmarks, how they were ported to VSIPL++, and their performance on commodity clusters and embedded systems. Using Sourcery VSIPL++, a commercially available, optimized implementation of the VSIPL++ API developed by CodeSourcery, we will demonstrate that high-performance benchmark implementations can be written that are portable across multiple platforms. We will also demonstrate the ease with which the VSIPL++ API for data-parallelism can be applied, and show that data-parallelism does not penalize serial performance.

HPEC Challenge Kernel-Level Benchmarks

The HPEC Challenge Benchmark Suite [4, 5] was created by MIT/LL to allow the quantitative evaluation of different HPEC embedded systems and tools. It consists of eight single-processor kernel benchmarks and a multi-processor scalable synthetic SAR benchmark. In this presentation, we describe the implementation of the suite's 4 signal processing kernel benchmarks and 1 communication kernel in VSIPL++.

VSIPL++

The VSIPL++ API [1] specifies a standardized C++ interface to parallel, high-performance, signal-processing libraries. VSIPL++ is the next-generation, object-oriented, parallel version of the popular C-VSIPL API [7]. VSIPL++ provides improvements in portability, productivity, and performance relative to both VSIPL and ad-hoc approaches.

Sourcery VSIPL++ is a high-quality, commercially available implementation of the VSIPL++ API developed by CodeSourcery. It has been designed from the ground up to deliver the full performance possible with the VSIPL++ API. It uses technology such as expression templates and math library dispatch to deliver serial application performance on par with vendor libraries, without sacrificing application portability. It is also a full implementation of the VSIPL++ Parallel API. For this paper, we used Sourcery VSIPL++ release 1.1, which runs on commodity Intel/AMD clusters and Mercury Embedded PowerPC multi-computers.

The HPEC Challenge Benchmarks exercise both the serial performance and data parallel aspects of VSIPL++.

Implementation Methodology

For each kernel benchmark, we created a VSIPL++ implementation.

Each benchmark was separated into initialization, computation, and finalization. Performance is measured for multiple iterations of the computation. The iteration count is adjusted so the time interval is large relative to the measurement error.

```
benchmark.init();
timer.start();
for (l=0; l<loop_count; ++l)
    benchmark.compute(...);
timer.stop();
benchmark.fini();
```

At each data point, multiple measurements are taken. Of these the median is used to for the reported value, and the min and max are used to indicate expected variation.

The HPEC Challenge Kernel benchmarks define a single set of parameters for each dataset. To determine how varying the parameters affect the benchmark performance, we perform each benchmark by sweeping one or more of the parameters.

Performance measurements will be taken on the commodity Intel/AMD clusters at the Georgia Tech Research Institute PaSTEC [3] and on embedded Mercury PowerPC multicomputers.

The following paragraphs describe in more detail the FIR filter bank, one of the kernels that we will present results for at the workshop.

FIR Filter Bank Kernel Benchmark Results

This benchmark represents a bank of M FIR filters, each with K unique coefficients, operating on input/output vectors of size N. For each input vector, the following output vector is computed:

$$y_m[i] = \sum_{k=0}^{K-1} x_m[i-k]\,w_m[k], \quad \text{for } i = 0, 1, \ldots N-1.$$

This can be implemented as either a time-domain or frequency-domain computation. Which is more efficient depends on the filter parameters.

Two datasets are defined (table 1). Set 1 is sized for efficient frequency-domain processing. Set 2 is sized for efficient time-domain processing.

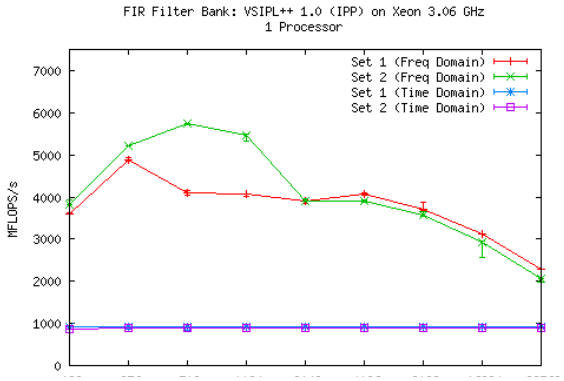
Parameter	Description	Set 1	Set 2
M	Number of filters	64	20
N	Length of input/output vectors	4096	1024
K	Number of filter coefficients	128	12
W	Workload (MFLOP)	33	1.97

Table 1: FIR Filter Bank Datasets

FIR Filter Bank Serial Implementation

Two implementations of the FIR filter bank were written. The first uses the VSIPL++ FIR object and is intended for the cases where time-domain computation is advantageous. The computation loop is:

Figure 1: Serial FIR Filter Bank Performance




```
for (index_type i=0; i<M; ++i)
   (*fir[i])(inputs.row(i),
             outputs.row(i));
```

The second implementation uses the VSIPL++ FFT object to perform an explicit fast-convolution. It is intended for cases were frequency-domain processing is advantageous. The computation loop is:

```
for (index_type i=0; i<M; ++i) {
   fwd_fft(inputs.row(i), tmp);
   tmp *= response.row(i);
   inv_fft(tmp, outputs.row(i);
   }
```

Fir Filter Bank Serial Performance

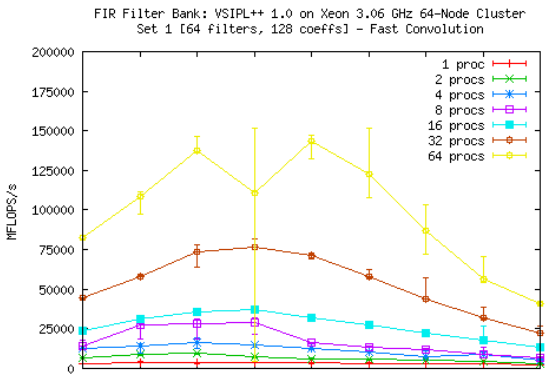
Using Sourcery VSIPL++ 1.1, results were collected on an Intel Pentium 4 Xeon system. Figure 1 shows the single processor performance.

FIR Filter Bank Parallel Implementation

VSIPL++ supports data-parallelism through global array semantics [6]. VSIPL++ blocks are given mappings that describe how they can be distributed over multiple processors. Operations on distributed blocks are then performed in parallel, with VSIPL++ managing the necessary communications and synchronizations.

Many VSIPL++ operations are implicitly data-parallel, requiring no changes after mappings have been given to data. Operations that are explicitly data-parallel, such as those used to implement the frequency domain FIR Filter bank, can be made to run in parallel by converting them use local views. For example, the following changes are necessary to the frequency domain implementation's computation loop:

Figure 2: Parallel FIR Filter Bank Performance




```
length_type l_M = inputs.local().size(0);
for (index_type i=0; i<l_M; ++i) {
   fwd_fft(inputs.local().row(i), tmp);
   tmp *= response.local().row(i);
   inv_fft(tmp, outputs.local().row(i);
   }
```

FIR Filter Bank Parallel Performance

Using Sourcery VSIPL++ 1.1, results were collected on the GTRI cluster's 32 dual-processor Intel Xeon nodes. Figure 2 shows the performance of data set 1 running from 1 to 64 processors.

Figure 3 shows the near linear speedup when the input/output size N is fixed at 4096 for data set 1.

References

[1] CodeSourcery, Inc. *VSIPL++ Specification 1.0.* Georgia Tech Res. Corp. 2005 [online] Available: http://www.hpec-si.org.

[2] CodeSourcery, Inc. Sourcery VSIPL++. [online] Available: http://www.codesourcery.com/vsiplplusplus.

[3] Georgia Tech Research Institute. *Parallel Software Testing and Evaluation Center.* [online] Available: https://pastec.gtri.gatech.edu/.

[4] R. Haney, T. Meuse, J. Kepner, and J. Lebak. T*he High Performance Embedded Computing (HPEC) Challenge Benchmark Suite.* MIT/LL 2005. [online] Available: http://www.ll.mit.edu/HPECChallenge.

[5] R. Haney, T. Meuse, J. Kepner, and J. Lebak. "The High Performance Embedded Computing (HPEC) Challenge Benchmark Suite." *HPEC Workshop*, Lexington, MA, 2005.

[6] J. Lebak. et al. “Parallel VSIPL++: an open standard software library for high-performance parallel signal processing,” *Proceedings of the IEEE*, Vol 93, Issue 2, Feb. 2005.

[7] D. A. Schwartz, R. R. Judd, W. J. Harrod, and D. P. Manley, *Vector, Signal, and Image Processing Library (VSIPL) 1.0 application programmer's interface*: Georgia Tech Res. Corp, 2000 [online] Available: http://www.vsipl.org.

Figure 3: FIR Filter Bank Performance For Dataset 1