VSIPL++Pro – A High-Performance VSIPL++ Implementation

Jules Bergmann[1], Mark Mitchell[1], Stefan Seefeld[1], Zack Weinberg[1], Nathan Myers[1], Rick Pancoast,
[1]CodeSourcery, LLC; [2]Lockheed Marting NE&SS
jules@codesourcery.com, Rick.Pancoast@lmco.com

Introduction

CodeSourcery, LLC is developing VSIPL++Pro, a high-quality implementation of the VSIPL++ API. Its goal is to enable the development of high-performance signal-processing applications that are portable, scalable, and written at the domain level.

CodeSourcery is working closely with Lockheed Martin NE&SS Moorestown to use VSIPL++Pro for signal processing applications from the AEGIS BMD program. CodeSourcery and Lockheed Martin plan to present performance, portability, and productivity results from VSIPL++Pro for both common SIP processing kernels and real SIP application codes.

Our goal is to demonstrate serial and parallel performance that is on par with applications written using existing math and communication libraries, but without sacrificing application portability or requiring explicit parallelization.

VSIPL++ API

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 [8]. VSIPL++ provides improvements in portability, productivity, and performance relative to both VSIPL and ad-hoc approaches.

VSIPL++ delivers high productivity by making it possible to implement signal and image processing algorithms close to the domain level. It provides the rich set of functionality specified by C-VSIPL, including vector math, linear algebra, and signal processing. C++ object-oriented syntax allows computations to be expressed directly, without requiring arcane function names and explicit types.

VSIPL++ achieves high-performance through both the efficiency of C++ as a systems language, and by using C++ features for high-performance generic programming to convert domain-level abstractions into efficient code at compile time. The VSIPL++ API allows for early binding of computation and communication, allowing setup for complex functions and communications to be done early, out of critical computation loops.

VSIPL++'s high-level mechanisms have been designed to map efficiently onto a range of processor and system architectures. This separation of application functionality concerns from the library's performance concerns improves portability. For example, VSIPL++ blocks abstract details of how and where data is stored so that an application can focus on the logical use of data while the implementation determines the best storage for the platform.

VSIPL++ is an API for both computation and communication, making it possible to write parallel programs that are not dependent on a particular paradigm (shared memory vs message passing) or particular system configuration (machine size). As a result, applications are scalable both up and down, easing both development and technology refreshes.

VSIPL++ portability and scalability simplifies software development for embedded systems. Initial development can be done on desktop machines, which are much less expensive than embedded platforms and often have better development tools. Scaling allows validation of functional performance against large volumes of data to be done on high-performance commodity clusters. Portability allows the validated software to deploy onto the embedded system with minimal effort. Moreover, as new technology becomes available, it can be accommodated without costly redesign and reimplementation of the software.

The simultaneous goals of performance, portability, and productivity are often at odds. Many library implementations are able to achieve two of the three at the expense of the third. Significant design effort has gone into VSIPL++ to achieve all three together. Many of the features and technologies that comprise VSIPL++ have individually been proven in research. Lessons learned from previous library designs have been applied [5, 4, 9]. While early experience using VSIPL++ has been positive [2, 3], its effectiveness in real usage on real applications has yet to be gauged.

In this presentation we will present VSIPL++Pro, a high-quality implementation of the VSIPL++ API designed from the ground up to take full advantage of VSIPL++'s potential. We will present implementation details of the library along with performance results for both SIP kernels and real SIP applications. The following sections describe several such areas. As the first high-quality implementation available, VSIPL++Pro offers an exciting glimpse at VSIPL++ potential.

Expression Templates

VSIPL++Pro allows developers to efficiently program close to the domain level. Consider an image non-uniformity correction (NUC) in VSIPL++:

```
Img = gain * Raw + offset;
```

This has clear notational convenience over the equivalent C-VSIPL:

```
vsip_mmul_f(tmp, gain, Raw);
vsip_madd_f(Img, tmp, offset);
```

Moreover, VSIPL++ gives better performance. Using expression templates, a technique that allows libraries to evaluate an entire expression at once [10], VSIPL++Pro transforms this expression into a single fused loop:

```
for (i=0; i<width*height; ++i)
   Img[i] = gain[i] * Raw[i] + offset[i];
```

By avoiding the temporary, VSIPL++ improves cache locality and reduces IO bandwidth.

Math Library Interface

While loop-fusion can out perform optimized library routines in some cases, it is often advantageous to use available libraries, especially for complex mathematical

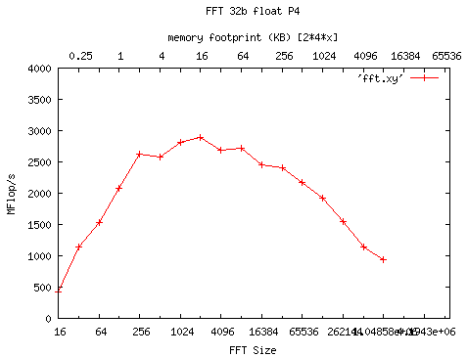

Figure 1FFT Kernel Performance

functions such as transforms and decompositions. Developing highly optimized routines for these requires significant skill, time, and effort. Moreover, system and processor vendors have invested significant effort to develop libraries that efficiently utilize their products.

Rather than reimplement these routines, VSIPL++Pro has a math library interface that allows the highest performance libraries available for a platform to be used. VSIPL++Pro's data structures are designed to allow efficient access to data in formats suitable for math libraries. VSIPL++ API functions use a dispatch mechanism to determine the most appropriate routine for a computation. Users can pass data from VSIPL++ data structures to legacy algorithms and libraries implementing functionality not provided by VSIPL++.

VSIPL++Pro's dispatch mechanism is extensible. In the future, users will be able to extend it to consider additional libraries. This will simplify the porting of VSIPL++Pro to new platforms, and allow applications to take advantage of new algorithm development in the research community.

The end result is that VSIPL++ programs will be able to take advantage of the best performance available for each platform.

Figure 1 characterizes VSIPL++Pro's current FFT performance on a 2 Ghz Pentium-M with 1024KB of L2 cache. For 1024 point FFTs, performance is 36% of peak. Figure 2 demonstrates VSIPL++Pro's FFT performance while executing an MTI (moving target indicator) application. From 0.002s to 0.008s, fast convolution is being performed using forward and inverse 512 point FFTs. From 0.010s to 0.011s doppler filtering is being performed

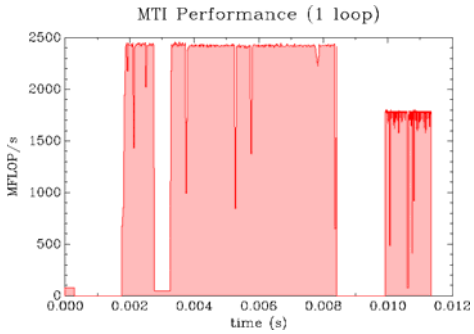

Figure 2FFT Application Performance

using 256 point FFTs. At the HPEC workshop, more complete characterizations will be presented, including comparisons against vendor optimized libraries.

Data-Parallelism

VSIPL++ is an API for both computation and communication. This eliminates the need to move data between different libraries. Moreover, VSIPL++'s approach to parallelism removes explicit assumptions on platform specifics such the number of processors or the rank of the local processor. Together these result in greater programming productivity and greater program scalability [6].

In previous work we demonstrated a prototype parallel VSIPL++ based on the reference implementation and showed that parallel VSIPL++ programs can have no overhead relative to comparable VSIPL/MPI programs [7].

VSIPL++Pro will fully implement the Parallel VSIPL++ API. It currently implements distributed vectors, matrices, and tensors with block-cyclic mappings in all dimensions, parallel assignment (“corner turns”), and data-parallel expressions. It is planned to provide full support for data-parallel operations, including signal-processing objects.

VSIPL++Pro's software architecture separates parallel VSIPL++ concerns (mappings, data layout, etc) from communication library concerns (efficient message transfer). Currently MPI is used for communications, but support for PAS is planned for 2006.

Signal Processing Application Results

CodeSourcery and Lockheed-Martin are currently working with VSIPL++Pro to implement signal-processing applications for the AEGIS BMD program. Results from this effort will be presented at HPEC.

CodeSourcery has also implemented several signal-processing demonstration programs for SAR and MTI, for which performance will also be available.

Availability

VSIPL++Pro will be available under two licenses. For commercial users, a commercially licensed will be available priced on a per-developer-seat basis with no royalty fees or hidden costs. In addition, an open source licensed version using the GPL will be available. This dual licensing provides the guaranteed support and maintenance that commercial users require, while making the library freely available for potential users to evaluate and for the research community.

VSIPL++Pro will be available for desktop and commodity clusters in Q4 2005. This release will be optimized for the Intel C++ compiler, Intel Performance Primitives (IPP) math library, and MPICH. VSIPL++Pro will be released for embedded Mercury systems in 2006. For more detailed availability information, including support for other platforms and math libraries, please contact CodeSourcery or visit www.codesourcery.com.

References

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

[2] J. Cook, et al. "Implementation of a Shipboard Ballistic Missile Defense Processing Application using the HPEC-SI API," *HPEC Workshop*, Lexington, MA 2004.

[3] D. Cottell and R. Judd. "Evaluation of the VSIPL++ Serial Specification using the DADS Beamformer," *HPEC Workshop*, Lexington, MA 2004.

[4] Parallel Object-Oriented Methods and Applications. [online] http://www.nongnu.org/freepooma.

[5] H. Hoffmann, et al, "Achieving Portable Task and Data Parallelism on Signal Processing Architectures," *HPEC Workshop*, Lexington, MA, 2000.

[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] M. Mitchell and J. Oldham. "VSIPL++: Parallel Performance," *HPEC Workshop*, Lexington, MA 2004.

[8] 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.

[9] T. Veldhuizen, *Techniques for Scientific C++*, Indiana University CS-TR-542, Aug. 2000.

[10] T. Veldhuizen, "Expression templates," *C++ Rep.*, vol. 7, no. 5, pp. 26-31, 1995.