

VSIPL++: A Signal Processing Library Scaling with Moore's Law

Jules Bergmann
CodeSourcery, LLC and
jules@codesourcery.com oldham@cs.stanford.edu

2005 May 20

VSIPL++ [1, 2] is the object-oriented successor to the Vector Signal and Image Processing Library (VSIPL) [23]. Like VSIPL, VSIPL++ specifies an Application Programming Interface (API) for use in the development of high-performance numerical applications, with a particular focus on embedded real-time systems performing signal processing and image processing. Its API is designed for performance, portability, and productivity so software written using this API will benefit from and be easily ported to the new and faster hardware architectures that Moore's Law continues to bring.

Moore's Law, which predicts the number of transistors available on an integrated circuit will double every eighteen months, has held for the past forty years, and it appears it will continue for at least the next decade [17]. During these years, microarchitectures have added many features to exploit single-thread instruction-level parallelism, such as pipelining, multiple out-of-order issue, speculative execution, and branch prediction. For a compiler to be able to exploit these features, a program needs to expose as much information as possible. VSIPL++ does this.

More recently, there has been a shift towards more parallel architectures with multi-threading [7, 12], multiple cores [8, 13], and many small processors connected with flexible interconnections [3, 16, 11, 20]. To exploit these technologies, signal and image processing software should use functional and data parallelism, moving data to where it is needed without increasing software complexity. VSIPL++ supports this.

The "PPP" Goals of VSIPL++

VSIPL++ is being developed by the High Performance Embedded Computing Software Initiative (HPEC-SI) [9] as an open-source, C++-based, middleware API with a publicly available specification. Working together, researchers, military contractors, and computer scientists designed the library with three objectives, permitting it to take full advantage of current and future hardware architectures. In particular, the VSIPL++ specification [1, 2] provides

performance: reductions in program execution times through more efficient use of current and future hardware architectures,

portability: porting to new hardware requires very few, if any, changes to existing programs, and

productivity: significant decreases in program length when using VSIPL++.

These "PPP" goals support write-once, run-everywhere VSIPL++ programs executing efficiently both on today's architectures as well as on future hardware architectures.

The VSIPL++ specification, version 1.0, [2] was released 2005 May so programmers can now benefit from this API. The corresponding open-source reference implementation [25] is also available both for use by programmers and as the basis of academic and commercial VSIPL++ implementations.

Performance

VSIPL++'s performance improvements derive from a combination of data parallelism and compiler and implementation optimizations such as loop-fusion technologies and algorithmic specialization. Using object-oriented syntax to simplify notation, VSIPL++ supports data-parallel computation. Element-wise operations such as vector, matrix, and tensor addition and subtraction and reduction operations such as vector and matrix maximum, sum, and mean can be computed using one or more processors or threads, depending on the underlying hardware. The specification supports implementing the same functionality using specialized hardware with no changes to programs' expressions. For example, the same functionality can be implemented using SIMD instructions to have a single processor perform multiple simultaneous operations, reducing these operations' execution times by up to a factor of four [21].

The specification requires VSIPL++ to make type information available during compilation so programs can be optimized. For example, it supports using compile-time expression-template technology to fuse loops so multiple data-parallel operations are implemented using a single loop, avoiding the need for storage of intermediate values [6, 19, 22, 24]. For example, the element-wise

*VSIPL++ development work performed while this author was an employee of CodeSourcery, LLC, but his opinions expressed here are his own.

assignment $A = C + C * D / E$ can be computed using one loop containing assignments to elements i : $A[i] = C[i] + C[i] * D[i] / E[i]$. In many hardware architectures, memory locality is important so the same technology permits a VSIPPL++ implementation to automatically reorder operations on matrixes and tensors into blocks when compiling without any revision of user programs.

The specification supports specialization of operations using C++ template specializations. For example, a convolution implementation designed to support all numeric types may be supplemented by a special faster implementation for `doubles`. At compile time, the most specialized (and most efficient) implementation will be used. The VSIPPL++ library implementer, a VSIPPL++ user, or even specialized hardware can provide the specialization. These are just a few examples of how the type information made available by VSIPPL++ during compilation permits it to incorporate current and future implementation and compilation techniques without the need to revise existing programs.

Portability

The VSIPPL++ specification facilitates porting programs to new hardware architectures with zero or very few changes in user programs. The features supporting portability include encapsulation of user data layout, built-in support for distributed data and computation, and type-independent expression syntax.

VSIPPL++ separates storage and use of data. Blocks contain data, whether stored or computed, and views such as vectors, matrixes, and tensors operate on data. Because of this separation, the VSIPPL++ specification supports uniprocessor, multi-processor, and multi-threaded computation with no changes to programs except for revising the declarations of the data blocks used by views. For example, to port a serial program for use with multiple processors or threads, the only necessary changes are declarations how the data should be distributed among the available processors or threads. Block, cyclic, and block-cyclic distributions are supported [1, 10]. The VSIPPL++ library is responsible for using MPI [14, 15], DRI [4], or threading commands to ensure data is moved to where it is needed for computation and storage. Making the library, not the programmer, responsible for data movement permits porting to new hardware architectures with different communication mechanisms.

Encapsulation of user data in blocks permits supporting special-purpose hardware and architectures, both existing and future. For example, a user-defined VSIPPL++ block can ensure data is stored according to SIMD memory alignment requirements. To use these blocks in

a program, the declarations of these data blocks must be modified, but no expressions need to be modified. The usual VSIPPL++ functions can still access this data, but faster, function template specializations using SIMD commands can be added to the library by the user or a VSIPPL++ implementor. When compiling, the types of blocks participating in expressions are determined and the faster, specialized implementations are used. Because this detection occurs during compilation, there is no run-time overhead for using specialized code. Similar extensions to support other hardware architectures such as PCAs [3], FPGAs, and data-parallel graphics processors [5, 18] are also possible. For each architecture, a new block would be introduced and some functionality would be specialized, either by the user or by a VSIPPL++ library implementer. VSIPPL++ programs need not be changed except for the declarations of a few data blocks. This easy migration path helps code take advantage of the new hardware that Moore's Law provides.

Productivity

VSIPPL++ improves programmer productivity by providing high-level, algorithmic syntax; by permitting program development and testing on desktop computers before porting to more complex hardware architectures; and by isolating hardware-dependent code in the library. VSIPPL++ provides all the normal linear algebra and signal processing functionality, e.g., vectors, matrixes, tensors, SVD solvers, FFTs, FIR filters, convolutions, and windowing functions, as well as expression syntax supporting normal mathematical and algorithmic notation, e.g., $x = 3 * \sin(y) - \text{fft}(x)$. Having program syntax similar to mathematical, algorithmic syntax simplifies translating from algorithms to programs and eases checking for correctness. The library guarantees its correctness so programmers can concentrate on correctly using the tools it provides.

Unlike other middleware, porting from a uniprocessor environment to a high-performance environment is trivial, as we described above; only modifying block declarations is required, and program correctness is maintained. No additional knowledge or statements, such as multi-processor communication, SIMD instructions, or threading primitives, need to be known by the VSIPPL++ user because the VSIPPL++ implementation is responsible for issuing commands to move data where it is needed for computation. As new, faster hardware architectures are created, the commands to use the architecture's features can be added to an existing VSIPPL++ implementation without affecting user code. This helps the software take advantage of Moore's Law.

References

- [1] CodeSourcery, LLC. VSIPL++ specification — parallel specification. <http://www.hpec-si.org/private/specification-parallel.pdf>, September 2004.
- [2] CodeSourcery, LLC. VSIPL++ specification 1.0 final. <http://www.hpec-si.org/spec-1.0-final.pdf>, May 2005.
- [3] DARPA Information Processing Technology Office. Polymorphous computing architectures (PCA). <http://www.darpa.mil/ipto/programs/pca/index.htm>.
- [4] Data Reorganization (DRI) Forum. Document for the data reorganization interface (DRI-1.0) standard. <http://www.data-re.org/documents/dri-report-09252002.pdf>, September 2002.
- [5] General-purpose computation using graphics hardware. <http://www.gpgpu.org/>.
- [6] Scott Haney, James Crotinger, Steve Karmesin, and Stephen Smith. PETE: The portable expression template engine. *Dr. Dobbs's Journal*, pages 88–95, October 1999. <http://www.ddj.com/documents/s=898/ddj9910h/>.
- [7] H. Peter Hofstee. Industry chip hardware technology. In Jeremy Kepner, editor, *Eighth Annual High Performance Embedded Computing (HPEC) Workshop*, number HPEC-7, Lincoln, MA, September 2004. Massachusetts Institute of Technology Lincoln Laboratory.
- [8] H. Peter Hofstee. Power efficient processor architecture and the cell processor. In *International Symposium on High-Performance Computer Architecture*, volume 11, Los Alamitos, CA, February 2005. IEEE Computer Society. http://www.hpcaconf.org/hpca11/papers/25_hofstee-cellprocessor_final.pdf.
- [9] High Performance Embedded Computing Software Initiative. <http://www.hpec-si.org>.
- [10] J. Lebak, J. Kepner, H. Hoffmann, and E. Rutledge. Parallel VSIPL++: An open standard software library for high-performance parallel signal processing. *Proceedings of the IEEE*, 93(2):313–330, February 2005. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1386654.
- [11] Ken Mai, Tim Paaske, Nuwan Jayasena, Ron Ho, William J. Dally, and Mark Horowitz. Smart memories: A modular reconfigurable architecture. In *ISCA '00: Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 161–171, New York, NY, USA, 2000. ACM Press. http://www.stanford.edu/~jayasena/docs/sm_isca_00.pdf.
- [12] Deborah T. Marr, Frank Binns, David L. Hill, Glenn Hinton, David A. Koufaty, J. Alan Miller, and Michael Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1), February 2002. http://www.intel.com/technology/itj/2002/volume06issue01/art01_hyper/p01_abstract.htm.
- [13] Jim McGregor. A day at the races. *Microprocessor Report*, 09 May 2005.
- [14] Message Passing Interface Forum. MPI: A message-passing interface standard. <http://www.mpi-forum.org/docs/mpi-11.ps>, June 1995.
- [15] Message Passing Interface Forum. MPI-2: Extensions to the message-passing interface. <http://www.mpi-forum.org/docs/mpi-20.ps>, July 1997.
- [16] MONARCH: A MOrphable Networked microARCHitecture. <http://www.isi.edu/asd/monarch/index.html>.
- [17] Gordon E. Moore. No exponential is forever ... but we can delay “forever”. ftp://download.intel.com/research/silicon/Gordon_Moore_ISSCC_021003.pdf, 10 February 2003. Presentation at the 2003 IEEE International Solid-State Circuits Conference.
- [18] John Owens. GPUs: Engines for future high-performance computing. In Jeremy Kepner, editor, *Eighth Annual High Performance Embedded Computing (HPEC) Workshop*, number HPEC-7, Lincoln, MA, September 2004. Massachusetts Institute of Technology Lincoln Laboratory.
- [19] PETE. <http://acts.nersc.gov/pete/>.
- [20] Rodric M. Rabbah, Ian Bratt, Krste Asanovic, and Anant Agarwal. Versatile tiled-processor architectures: The raw approach. In Jeremy Kepner, editor, *Eighth Annual High Performance Embedded Computing (HPEC) Workshop*, number HPEC-7, Lincoln, MA, September 2004. Massachusetts Institute of Technology Lincoln Laboratory.
- [21] Edward Rutledge. AltiVec extensions to the portable expression template engine (PETE). In

Sixth Annual High Performance Embedded Computing (HPEC) Workshop, number HPEC-5, Lincoln, MA, September 2002. Massachusetts Institute of Technology Lincoln Laboratory.

- [22] Edward M. Rutledge. C++ expression templates in an embedded, parallel, real-time, signal processing library. In *Fourth Annual High Performance Embedded Computing (HPEC) Workshop*, number HPEC-3, Lincoln, MA, September 2000. Massachusetts Institute of Technology Lincoln Laboratory.
- [23] David A. Schwartz, Randall R. Judd, William J. Harrod, and Dwight P. Manley. VSIPL 1.1 API. http://www.vsipl.org/VSIPL_1p1.pdf, June 2002.
- [24] T. Veldhuizen. Expression templates. *C++ Report*, 7(5):26–31, June 1995. <http://osl.iu.edu/~tveldhui/papers/Expression-Templates/exprtmpl.html>.
- [25] Serial VSIPL++ reference implementation. <http://www.hpec-si.org/private/software1.html>, March 2005.