PVTOL: A High-Level Signal Processing Library for Multicore Processors

Hahn Kim, Nadya Bliss, Ryan Haney, Jeremy Kepner,
Sanjeev Mohindra, Sharon Sacco, Glenn Schrader, Edward Rutledge
{hgk, nt, haney, kepner, smohindra, ssacco, gschrad, rutledge}@ll.mit.edu
MIT Lincoln Laboratory, 244 Wood St., Lexington, MA 02420-9108

1 Introduction

For decades, Moore's Law has enabled ever faster processors that have supported the traditional von Neumann programming model, i.e. load data from memory, process, then save the results to memory. As clock speeds near 4 GHz, physical limitations in transistor size are leading designers to build more processor cores (or "tiles") on each chip rather than faster processors. Multicore processors improve raw performance but expose the underlying processor and memory topologies. This results in increased programming complexity, i.e. the loss of the von Neumann programming model.

Consider IBM's Cell Broadband Engine. The Cell architecture consists of 9 cores: 1 PowerPC Processing Element (PPE) and 8 Synergistic Processing Elements (SPE), the Cell's processing engines. The Cell architecture acheives very high levels of performance. IBM's Cell API is very low level, however, and has a steep learning curve [1]. Applications launch on the PPE, which spawns threads onto the SPE's. The PPE loads data into main memory, and then each SPE transfers data from main memory into its local memory via DMA's.

The Cell is just one of many emerging multicore architectures, each with its own programming model. The MIT Lincoln Laboratory is developing the Parallel Vector Tile-Optimized Library (PVTOL) which provides a uniform means of writing high-performance signal processing code that is portable across a range of traditional and multicore architectures [2]. PVTOL also increases productivity by providing a set of high-level programming constructs.

2 Parallel Vector Tile-Optimized Library

PVTOL's architecture consists of several layers, shown in Figure 1. At the user interface level, PVTOL provides an object-oriented C++ API. Internally, PVTOL is built on existing technologies optimized for various platforms. This architecture results in a middleware library that maintains high productivity, high performance and portability.

2.1 Productivity

PVTOL's API provides high-level constructs for data (vectors, matrices and tensors), data distribution (maps), communication (conduits) and computation (tasks and kernels). These constructs increase productivity by supporting a partitioned global address space (PGAS) programming model, which overlays a shared, global address space on a physically partitioned address space.

Maps were introduced in Lincoln's Parallel Vector Library (PVL) and later adopted by the VSIPL++ standard. Maps concisely describe how to allocate parallel arrays across multiple processors. PVTOL has *hierarchical maps* that describe how to allocate *hierarchical arrays* across the processor hierarchy [3]. For example, a hierarchical map for the Cell may consist of two maps. One map distributes an array between multiple Cell processors, similar to PVL and VSIPL++. The other map distributes each processor's data across its SPE's. Figure 2 shows sample PVTOL code that allocates hierarchical matrices to be processed by a frequency-domain FIR filter kernel. Details have been omitted for the sake of brevity.

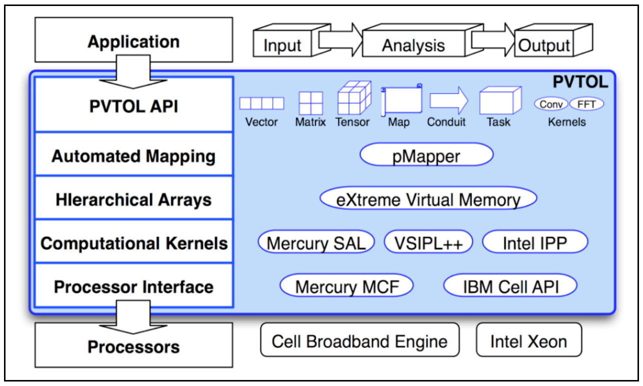

Figure 1: PVTOL Architecture

```
// Distribute across cores
Grid grid2(...);
DataDist dist2(...);
Vector<int> procs2(...);
ProcList procList2(procs2);
RuntimeMap map2(grid2, dist2, procList2);

// Distribute across processors
Grid grid(...);
DataDist dist(...);
Vector <int> procs(...);
ProcList procList(procs);
RuntimeMap map(grid, dist, procList, map2);

// Create input, weights, and output matrix
typedef Dense<2, float, tuple<0, 1> > blk_t;
typedef Matrix<float, blk_t, RuntimeMap> mat_t;
mat_t input(num_vects, len_vect, map),
      filts(num_vects, len_vect, map),
      output(num_vects, len_vect, map);

// Initialize matrices
...

// FDFIR filter
output = fdfir(input, filts);
```

This work is sponsored by the Department of the Air Force under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government

Figure 2: PVTOL code for a frequency-domain FIR filter

Nominally, programmers explicitly construct maps to parallelize arrays. Alternatively, programmers can use PVTOL's *automated mapping* capability, a first among signal processing middleware libraries [4]. PVTOL allows programmers to tag arrays for automatic mapping; based on the computation, data flow and processing architecture, PVTOL determines the optimal mapping.

2.2 Performance

Optimized libraries for communication and computation are available for most processor architectures. For example, Intel's Integrated Performance Primitives (IPP) is an optimized computation library for Intel processors; Mercury Computer Systems' MultiCore Framework (MCF) and a version of their Scientific Algorithm Library (SAL) are optimized communication and computation libraries, respectively, for their Cell-based systems.

PVTOL is able to use vendor optimized libraries. Where possible, PVTOL also uses standard vendor-neutral libraries that support a range of architectures. Using industry standards such as VSIPL++ expands PVTOL's portability.

2.3 Portability

By building on a range of programming technologies, PVTOL provides two degrees of portability. First, PVTOL's support for both traditional and multicore architectures allows programmers to incrementally development applications. Programmers first develop on desktop workstations. Once serial correctness is verified, the application is parallelized on a cluster by adding maps. Finally, parallelized applications can be deployed onto multicore processors by adding hierarchical maps.

Second, PVTOL applications can be easily ported to new multicore architectures, allowing developers to easily take advantage of technology refresh cycles.

3 Results

We have built a prototype of PVTOL and implemented several computational kernels: a time-domain FIR filter bank (TDFIR), a frequency-domain FIR filter bank (FDFIR) and an image projective transform (PT). TDFIR and FDFIR are part of the HPEC Challenge Benchmark Suite [5]; PT was developed for a project at Lincoln.

Figure 3 compares performance and productivity of four implementations of the FDFIR kernel using IPP, Sourcery VSIPL++ (using IPP) [6], MCF/SAL, and PVTOL. The IPP and VSIPL++ implementations were run on a 3.2 GHz Intel Xeon processor and the MCF/SAL and PVTOL implementations were run on an 3.2 GHz IBM Cell processor using all 8 SPE's.

The X-axis shows productivity and the Y-axis shows performance in GFLOPS. The IPP code acheives 4.48 GFLOPS but suffers from low productivity due to larger code size. The version of VSIPL++ we used is built on IPP and, hence, acheives 4 GFLOPS or about 90% of IPP performance. VSIPL++'s productivity is much higher due to a significant reduction in code size.

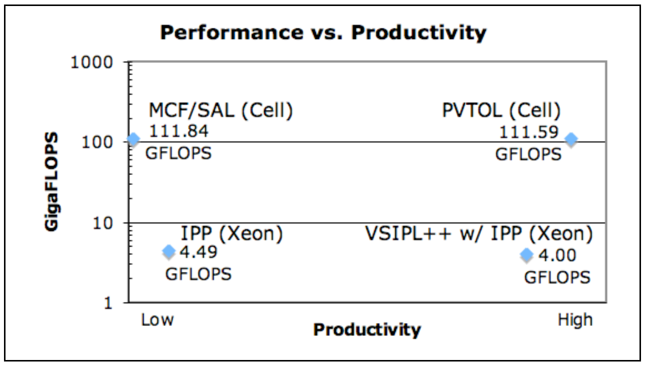

Figure 3: HPEC Challenge FDFIR benchmark

The Mercury code shows a dramatic increase in performance of the Cell over the Xeon, acheiving 111.84 GFLOPS. Acheiving this performance, however, requires significantly more code than the IPP code. The PVTOL version is built on top of the MCF/SAL kernel, acheiving 111.59 GFLOPS with a higher level of productivity than VSIPL++.

4 Summary

By hiding the complexity of multicore architectures, PVTOL's PGAS model preserves the simple von Neumann model familiar to most programmers, improving programmer productivity. Building on technologies optimized for a variety of architectures allows PVTOL to achieve high performance and portability.

We are continuing to implement PVTOL programming constructs for the Intel and Cell architectures and include support for more computational kernels. As the Cell implementation of PVTOL matures, we will investigate supporting other promising multicore architectures.

References

[1] S. Sacco, G. Schrader, J. Kepner. "Exploring the Cell with HPEC Challenge Benchmarks." *High Performance Embedded Computing Workshop 2006*. Lexington, MA. September 2006.

[2] G. Schrader, J. Kepner. "High Performance Signal Processing using Next Generation Multi-Core Processors and Standard APIs." *High Performance Computing Modernization Program Users Group Conference 2006*. Denver, CO. June 2006.

[3] H. Kim, J. Kepner. "Parallel Out-of-Core Programming in MATLAB Using the PGAS Model." *PGAS Programming Models Conference 2006*. Washington, DC. October 2006.

[4] N. Bliss, S. Mojindra. "pMapper: Automatically Partitioning the Global Address Space." *PGAS Programming Models Conference 2006*. Washington, DC. October 2006.

[5] HPEC Challenge website. http://www.ll.mit.edu/hpecchallenge

[6] J. Bergmann, D. McCoy. "Sourcery VSIPL++ HPEC Benchmark Performance." *High Performance Computing Modernization Program Users Group Conference 2006*. Denver, CO. June 2006.