Sourcery VSIPL++ for the Cell/B.E.

High Level Libraries for Multi-core Architectures

J. Bergmann[1], M. Mitchell[1], D. McCoy[1], S. Seefeld[1], A. Salama[1], F. Christensen[2], R. Pancoast[3], T. Steck[3]
[1]CodeSourcery, Inc. [2]IBM [3]Lockheed-Martin
jules@codesourcery.com

Introduction

Sourcery VSIPL++ for the Cell/B.E. implements the open standard VSIPL++ signal and image-processing API [1] on the IBM Cell/B.E. multi-core processor architecture [3]. It is suitable for implementing high-performance signal-processing applications that take full advantage of the Cell/B.E. processor throughput, without sacrificing programmer productivity or application portability.

For example, fast convolution in VSIPL++ sustains over 80 GFLOP/s on a single Cell/B.E (40% of peak) with no architecture specific code. The algorithm scales to multiple processors, sustaining over 320 GFLOP/s on four Cell/B.E.s. It remains portable to other architectures, achieving 6 GFLOP/s on Intel Xeon and 6.6 GFLOP/s on Power.

Cell/B.E.

The Cell/B.E. is an asymmetric, multi-core processor architecture developed by IBM, Sony, and Toshiba. It is described as "supercomputer on a chip" capable of over 200 peak single-precision GFLOP/s on a single chip with 9 cores. For more detailed descriptions of its architecture, refer to [3].

The key challenge for Cell is programming it effectively. At the low-level, the cores' simple micro-architecture provides an attractive programming model: in-order issue, uniform large register file, fixed memory latency, and off-loaded communication. However, at the high-level, the asymmetry and tiered memory require applications to expose and manage greater coarse-grain parallelism.

Coding to the architecture directly has the potential for high-performance, but it also limits portability to/from other architectures, and lowers developer productivity. Experience in DOD software development, where system lifecycles are much longer than technology refresh rates, shows that low portability and low productivity lead to much greater software development costs and program risk. Successful adoption of the Cell/B.E. architecture into DOD programs requires software development approaches that can achieve high-performance without sacrificing portability and productivity.

VSIPL++

VSIPL++ [1] is an open standard, high-level API for parallel high-performance signal and image-processing. It is defined by the High Performance Embedded Computing Software Initiative (HPEC-SI) [4], a consortium of industrial, academic, and governmental partners, with sponsorship from the DOD. VSIPL++ defines a pure C++ interface for operations including FFTs, filters, linear system solvers, and other operations useful in developing radar, sonar, communication, and medical imaging applications.

The API's goal is to simultaneously deliver the "three P's" – productivity, portability, and performance. Improved productivity derives from the high-level functionality which requires fewer lines of code to express complex algorithms. Greater portability follows from standardization and the broader optimization scope afforded by high-level descriptions. Higher performance results from sophisticated implementation techniques allowed by the API design.

Sourcery VSIPL++

Sourcery VSIPL++ is a high-performance implementation of the parallel API. Sourcery VSIPL++ uses a number sophisticated implementation techniques to achieve high performance on GNU/Linux, Mercury Power, and Windows single and multiple processor systems [2].

Expressions templates allow the library to manipulate parse trees for application code at compile time. Code is evaluated by a powerful, extensible *dispatch engine*. Compile-time attributes (such as dimension ordering, and parallel distribution) and run-time attributes (such as stride) are considered to choose the highest performance implementation.

Sourcery VSIPL++ can take advantage of existing optimized low-level math libraries, such as the Intel Performance Primitives (IPP) or the Mercury Scientific Algorithm Library (SAL). Simple operations, such as vector addition or matrix product, can be dispatched through a math library interface to vendor libraries with near zero overhead.

Sourcery VSIPL++ recognizes *fused operations* from simple operations, like fused multiply-add: A*B + C, to complex operations like fast convolution, shown below. Dispatch considers the entire fused operation allowing global optimizations to be performed, such as changing order of computation to improve cache locality and reduce memory bandwidth.

Sourcery VSIPL++ for Cell/B.E.

Sourcery VSIPL++ for the Cell/B.E. balances a simple programming model with optimal utilization of the Cell's capability. The PPE is used to run the application. The SPEs are used as high-performance computation engines. IBM's Acceleration Library Framework (ALF) [5] manages the SPEs, handling initialization and double-buffered data transfer to hide communication latency behind computation.

Sourcery VSIPL++'s dispatch engine recognizes computation which can be mapped to the SPEs. Compile-time and run-time attributes control which and how many SPEs are allocated for a computation. A variety of factors are considered, including data layout, operation being performed, and the ratio of computation to communication. Application attributes can also be used to tune the allocation.

This approach allows existing VSIPL++ codes to take advantage of the Cell/B.E.s by recompiling. Additional performance may be gained by tuning data structure attributes to influence resource allocation, and using fused operations to create locality and optimization potential.

Example: Fast Convolution

Fast convolution – convolution in the frequency-domain – is widely used in signal processing applications to implement filters and other convolutions. In radar pulse compression it implements a matched filter on received radar data against the transmitted pulse. A datacube contains many pulses which can be filtered independently. In VSIPL++ this is expressed using multiple FFT (`Fftm`) objects and vector-matrix multiply (`vmmul`).

First, views are declared to hold the data cube and the FFT signal processing objects:

```
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);
Domain<2> dom(n_pulses, n_cells);
Fftm<T, T, row, fft_fwd> fwd(dom, 1.0);
Fftm<T, T, row, fft_inv> inv(dom, 1.0/n_cells);
```

Next, the weights are transformed into the frequency domain via an in-place FFT:

```
fft_ip<fwd_fft>(weights);
```

Finally, the convolution itself is expressed as:

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

This statement performs three steps: First, the `fwd Fftm` object transforms the rows of matrix `data` from the time-domain to the frequency-domain. Next, the vector-matrix multiply operation multiplies each frequency-domain row by the `weights` vector. Finally, the `inv Fftm` object transforms the result back into the time-domain.

On the Cell/B.E., Sourcery VSIPL++'s fuses the three steps into a single fast convolution operation. The dispatch engine maps this to a fused convolution kernel that runs on the SPEs. This kernel streams through `data`, performing forward FFT, vector-multiply, and inverse FFT one row at a time. Fusion allows intermediate results to be kept within the SPE, eliminating unnecessary communication.

Performance

Figure 1 shows the fast convolution performance on the Cell/B.E.

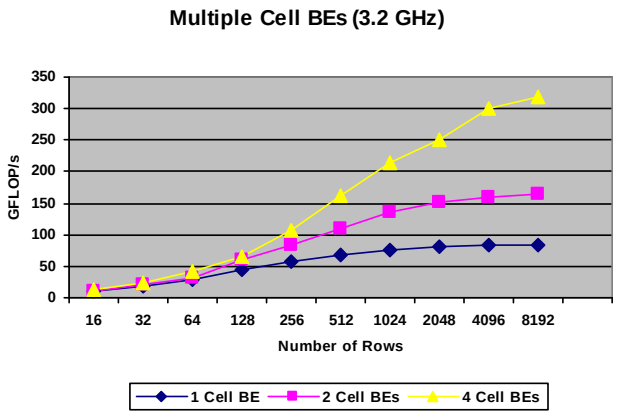

Figure 1: Cell/B.E. Fast Convolution Performance

The dark blue line shows performance for a single Cell/B.E. At 4096 rows, 82.6 GFLOP/s is sustained (40.3% of the SPEs' theoretical peak performance of 204.8 GFLOP/s). This corresponds to 11.6 GB/s of memory bandwidth (45.3% of the peak bandwidth of 25.6 GB/s).

Scalability. The pink and yellow lines show performance using 2 and 4 Cell/B.E.s respectively. Going from 1 to multiple PPEs is controlled via run-time parameters to the application. For 4 processors (yellow), 300 GFLOP/s is sustained at 4096 rows, a 3.6-fold speedup. The speedup is sub-linear because each Cell is now processing a smaller, less efficient problem size. When the problem size is scaled with the number of processors, the speedup is over 3.9 times.

Portability. Because the VSIPL++ API is portable, and because Sourcery VSIPL++ runs on multiple architectures, the fast convolution application runs on other systems as well. A recompilation is all that is required. Figure 2 shows fast convolution performance on the Pentium 4 Xeon, Power 970FX, and Cell/B.E. architectures.

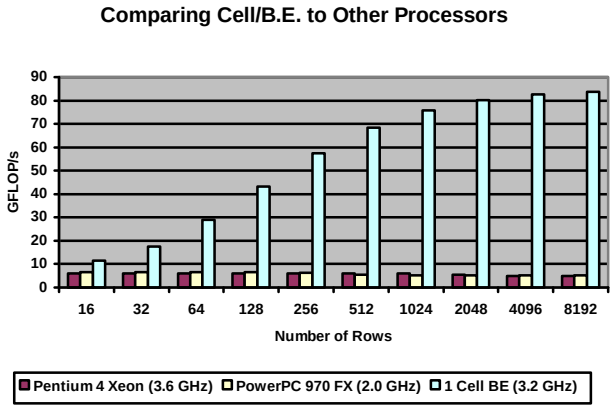

Figure 2: Comparing Cell/B.E. to Other Processors

For the Pentium 4 Xeon processor, Sourcery VSIPL++ uses IPP for low-level operations. The best sustained performance is 6.0 GFLOP/s, 41.8% of theoretical peak. On the PowerPC 970 FX, Sourcery VISPL++ uses FFTW for FFTs. The best sustained performance is 6.6 GFLOP/s, 41.2% of theoretical peak.

The Cell/B.E. substantially outperforms these other processors.

Conclusion

The Cell/B.E. architecture's performance capability has strong potential for signal processing applications. Sourcery VSIPL++ provides a demonstrated development approach for Cell/B.E. that delivers performance without sacrificing application portability or developer productivity.

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] M. Gschwind, et al. "Synergistic Processing in Cell's Multicore Architecture." *IEEE Micro*, March 2006.

[4] High Performance Embedded Computing Software Initiative. [online] Available http://www.hpec-si.org/.

[5] IBM Cell Broadband Engine Software Development Kit. [online] Available: http://www.alphaworks.ibm.com/tech/cellsw?open&S_TACT=105AGX16&S_CMP=DWPA