Simple, Efficient, Portable Decomposition of Large Data Sets

William Lundgren (Gedae, Inc., wlundgren@gedae.com), David Erb (IBM, djerb@us.ibm.com), Max Aguilar (IBM, maguilar@us.ibm.com), Kerry Barnes (Gedae, Inc., kbarnes@gedae.com), James Steed (Gedae, Inc., jsteed@gedae.com)
Gedae, Inc., 1247 N. Church St., Suite 5, Moorestown, NJ 08057
IBM, 11501 Burnet Rd., Austin, TX 78758

Introduction

The age of multicore processors has made the decomposition of large data sets and distribution of that data through hierarchical memory structures a key hurdle in both program efficiency and programmer productivity. Single cores often have limited local storage. For example, the current implementation of the Cell Broadband Engine (Cell/B.E.) processor combines one Power Processing Element (PPE) with 8 identical Synergistic Processing Elements (SPE). Each SPE contains a high-speed SIMD processor with its own 256 kB local store (LS) and DMA engine as shown in Figure 1. Larger system memory is available via a memory controller with a bandwidth of 25.6 GB/s – much less than the high speed Element Interconnect Bus (EIB). These issues are not unique to the Cell processor; software cache management on Intel Core 2 processors, shared memory management on embedded DSP boards, and management of other modern memory structures require programmer skill and time to produce efficient applications.

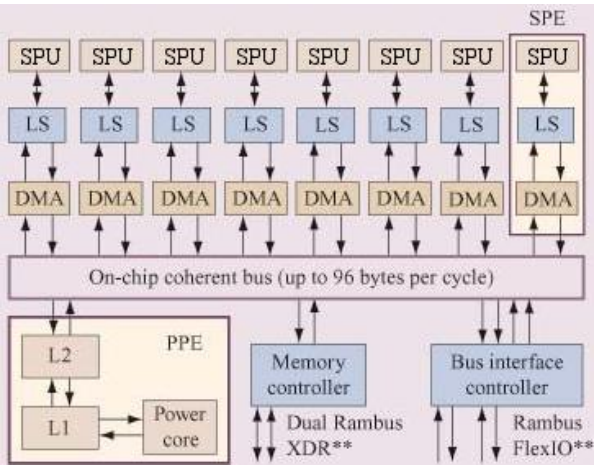

Figure 1 – The current Cell/B.E. Processor combines a PPE core with 8 SPE cores, all interconnected via a high-speed bus.

Previously, the Gedae programming language has been used to benchmark the Cell/B.E. platform for several key applications, including matrix multiplication and both the polar format algorithm (PFA) and convolution backprojection (CBP) methods of synthetic aperture radar (SAR) image processing. To optimally use the hierarchical memory of the Cell/B.E. architecture, complex flow graphs were created. These flow graphs achieved efficiencies at or above hand-coded benchmarks for the architecture, but the complexity of the programs was not substantially better than the hand-coded versions. Several ad hoc programming constructs were utilized to efficiently use the hardware, including the decomposition and distribution of the data sets and the multibuffering of the communication between system memory and SPE LS. This presentation will discuss additions to the Gedae language which automate the decomposition, communication, and multibuffering of large data sets. Through the specification of token decomposition and mapping of token parts to memory partitions, the Gedae compiler autocodes the implementation, inserts efficiency considerations such as use of DMA and multibuffering, protects against pointer misuse, and provides readability, maintainability, and portability of the algorithm design. The three benchmarks have been reimplemented using the new capability, and the focus of the discussion will be illustrating how the benchmarks utilize the features and demonstrating the automated implementation still achieves the highest levels of performance.

Matrix Multiplication Algorithm

The matrix multiply's core operation is a multiply-add which lends itself well to the SPE's VMX SIMD instruction set. Also, the operation can be easily decomposed into tiles that fit inside the SPEs' limited local storage. To perform a parallel matrix multiply, we tile the matrices and multiply a row of tiles from the first matrix with a column of tiles from the second matrix, accumulating the results, i.e.,

$$out[p,i1,i2][j1,j2] \mathrel{+}= [p]a[i2][k2](i1,k1) * b[k2][j2](j1,k1)$$

where the parentheses notation indicates temporal distribution, the square brace preindex indicates spatial distribution, and the square brace postindex indicates normal array indices.

A key issue in implementing this benchmark is overlapping the communication with the processing. A general matrix multiply accumulate kernel (equivalent to BLAS' GEMM operation) is the workhorse of the tiled algorithm, and if the developer can fetch the next tiles to process while the GEMM operation is processing the current tiles, the algorithm can achieve close to the highest FLOP count for the hardware. The theoretical maximum performance of a single Cell/B.E. chip is 204.8 GFLOPS, and the Gedae implementation of this algorithm can maintain a rate of 95% of that peak theoretical rate using a block data layout and 84% using a normal data layout.

To tile the matrices in this algorithm, we introduce tiled data streams to the language. A non-tiled R-by-C matrix stream is declared as

```
stream datatype in[R][C];
```

We extend this declaration to include an optional tile size Rt-by-Ct,

stream datatype in[Rt:R][Ct:C];

where the memory region is defined by the R and C variables but the token is defined by the Rt and Ct variables. The pointer "in" is the address of the first element of the tile, inside the R-by-C region. The i^{th} row of the tile can be accessed by adding i*C to the pointer.

Using this new syntax, new partitioning and concatenation kernels are available to partition matrices spatiallyand temporally, which can be generated and inserted into the application automatically based on the developer's decomposition of the data set. While 2-d partitioning can be done with this syntax, we have chosen to implement this benchmark using a nested partitioning, partitioning the left-hand matrix first row-wise and then column-wise and partitioning the right-hand matrix first column-wise and then row-wise. The temporal partitioning is shown in Figure 2 and the temporal concatenation in Figure 3.

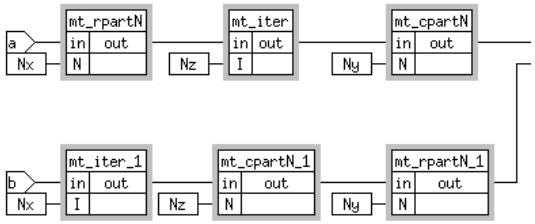

Figure 2 – Form the temporal partitioning and iteration required to implement a tiled matrix multiply.

To perform the partitioning and concatenation kernels inplace (without memory copies), pointers are manipulated. The partitioning kernel is straightforward; an input token arrives, and pointers to submatrices are sent downstream. The concatenation kernel requires extra care by the compiler's scheduling algorithm. Pointers are fanning-in to form a larger output token; thus, the concatenation kernel must provide submatrix pointers to the upstream kernels. With this pointer manipulation and tiling syntax, the algorithm can be written using 10 kernels, none of which contains more than 10 lines of code in its Apply method.

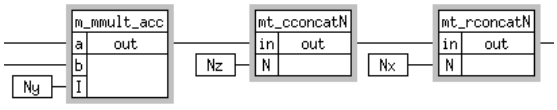

Figure 3 – Form the temporal concatenation of the matrix products.

Synthetic Aperture Radar Algorithm

The PFA SAR algorithm has two key stages: the range processing of the rows of the matrix and the azimuth processing of the columns of the matrix. To distribute the SAR algorithm we must add three stages: the partitioning of the data to distribute the rows across the processors, a corner turn of the data (distributed matrix transpose) to transition between range and azimuth processing, and the concatenation of the column-based results. In real-world applications, we must also do additional preprocessing to unpack error check the data. The Gedae implementation of this distributed algorithm achieves over 88 GFLOPs on 8 SPEs, sustaining a rate of 18.4 GB/s to multibank system memory, which acts as a bottleneck to the FLOP rate.

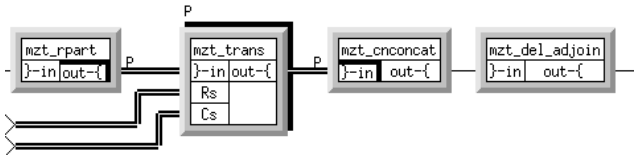

Figure 4 – Corner-turn with adjoin implemented with tiling.

The range, corner-turn with adjoin, and azimuth phases of the algorithm can each be easily written using the tiling notation. The corner-turn with adjoin phase is shown in Figure 4. Each SPE will transpose $1/N^{th}$ of the rows of the matrix. The spatial partitioning and concatenation kernels are used to distribute the work to "p" processors, partitioning row-wise and concatenating column-wise. The transpose subgraph further tiles the data for use on the SPE, partitioning the data in both dimensions. The entire PFA application uses 20 kernels, each with no more than 10 lines of Apply method code.

The CBJ SAR algorithm is also presented. The CBJ algorithm decomposes into tiles, as shown in Figure 5, although the decomposition must be arranged carefully to minimize DMA accesses over the memory controller.

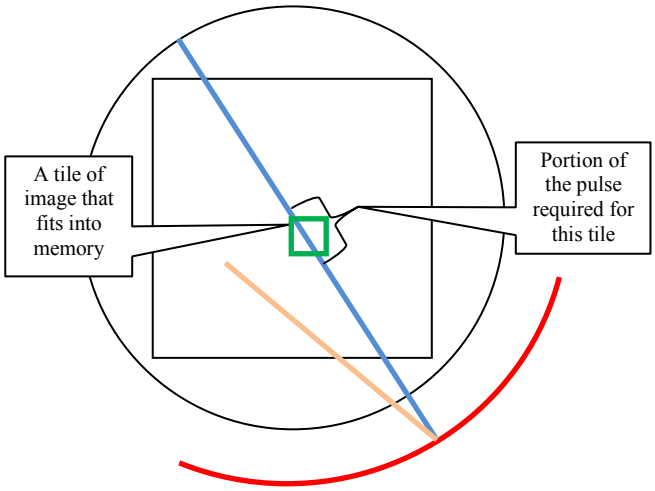

Figure 5 – Backprojection can be decomposed such that only one tile is processed at a time.

References

[1] IBM, Sony Computer Entertainment, Toshiba. *Cell Broadband Engine Programming Handbook*, Version 1.1, April 2007. <http://www.ibm.com>.

[2] D. Hackenberg, "Fast matrix multiplication on Cell (SMP) systems," July 2007 <http://www.cellperformance.com/articles/2007/07/fast_matrix_multiplication_on.html>.

[3] W. Lundgren, et al., "Programming Examples That Expose Efficiency Issues for the Cell Broadband Engine Architecture," *HPEC Conference*, September 2007.

[4] W. Lundgren, et al., "Implementing SAR Image Processing Using Backprojection on the Cell Broadband Engine," *IEEE Radar Conference*, May 2008.