Executive Summary

The Gedae Development Environment (Gedae DE) for the Cell Broadband Engine™ (Cell/B.E.) processor is a platform that allows programmers to leverage the exponential power of a new generation of multi-core processors, including the Cell Broadband Engine. In this technical note we study the issues associated with implementing the SAR algorithm with Gedae on the Cell Broadband Engine. We demonstrate a **116x** speed-up in frame rate over a quad 500 MHz PowerPC board using the Gedae DE.

Introduction

The Cell Broadband Engine Architecture was developed as a collaboration between Sony, Toshiba, and IBM. 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 (Single-Instruction, Multiple-Data) processor with its own 256 kB local store and DMA engine. The 9 cores and the on-chip memory controller and I/O controller are interconnected with the high speed Element Interconnect Bus (EIB). The EIB provides a measured peak bandwidth of over 200 GB/s at 3.2 GHz.

Because the PPE provides relatively modest performance, the key to using the Cell/B.E. processor efficiently is to take best advantage of the SPEs. While an SPE can process vector arithmetic very efficiently, each SPE has only 256 kB of dedicated local storage which holds both instructions and data. For instance, an entire 1k-by-1k complex matrix cannot reside in the SPE local storage, even if the program size is zero. Therefore large data sets will initially reside in system memory, and then the developer will stripmine the data, bringing it chunk-by-chunk to the SPE local storage for processing.

We studied how these programming considerations affect the implementation of the synthetic aperture radar (SAR) algorithm and the performance improvement possible by distributing the work to the Cell/B.E. processor's cores, as well as the issues that needed to be resolved to create that distribution. The Gedae programming language and development tools were used to conduct these experiments on the Cell/B.E. Architecture and analyze the performance.

Distributing the SAR Algorithm

To study the practicality and performance of real-world applications on the Cell/B.E. processor, we ported a SAR algorithm to the PPE and SPEs. The 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 added 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. With these added processing requirements due to the distribution, the distributed SAR algorithm is implemented as shown in Figure 1.

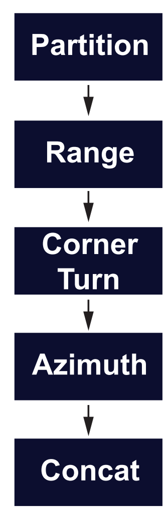

Figure 1

A key issue in implementing this distributed SAR algorithm efficiently on this architecture is the implementation of the corner turn. While the EIB provides a high bandwidth between the SPEs for corner turn operations, the data sets typical for a SAR algorithm cannot fit in the local storage of the

Gedae Inc.
1247 N Church Street, STE 5, Moorestown, NJ 08057 USA
www.gedae.com; gedae@gedae.com; 1-856-231-4458

SPEs. To do the work of a corner turn on a large SAR data set, the results of the range operation must be transferred to system memory, and then – once full columns are available – transferred back to the SPEs in transposition. This approach places demands on the bandwidth between the SPEs and system memory as opposed to the relatively fast bandwidth between SPEs.

A second issue in implementing this algorithm on this architecture is taking advantage of the 4 ALU paths on each of the SPEs. The SPEs can perform vector processing with amazing efficiency, but poorly written code does not fully utilize this resource. As the SPEs local storage must be used as both program memory and data memory, its size is a limiting factor. Program overhead must be minimized and unused code must be removed so that the implementation can be achieved.

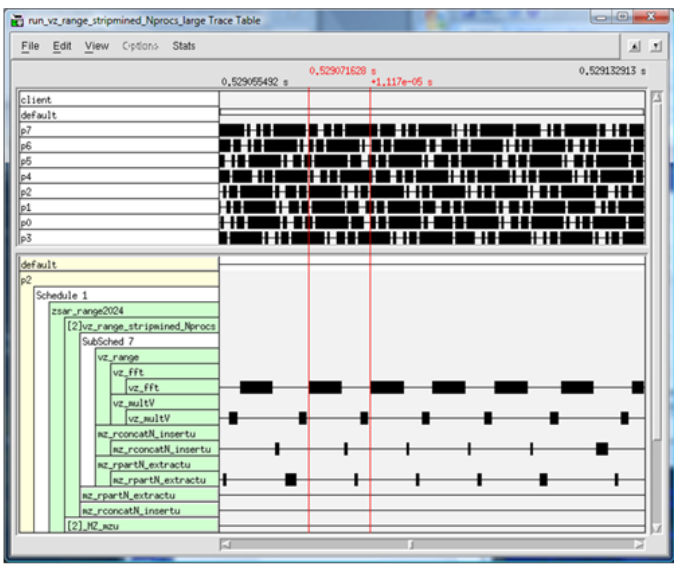

Figure 2

Gedae has been used to implement this distributed SAR algorithm and map it to all 8 SPEs on a Cell/B.E. processor. The Trace Table for this implementation is shown in Figure 2. The top of the table shows the load for each of the 8 SPEs (the black bars showing the time the SPE is taking to process data), and the bottom of the table is collapsed to show the processing of one of the 8 threads in the application, highlighting the range processing.

Conclusions/Analysis of Results

Algorithm	Rate (GFLOPS)	Rate(GB/s)
Range	71.1	19.9
Corner Turn	-	22.6
Azimuth	128.4	14.3
Total SAR	81.1	16.9

Multi-core architectures like the Cell/B.E. Architecture are powerful compute engines as illustrated in the achieved GFLOP rates in the table above. The Gedae implementation sustains a rate of 81.1 GFLOPS during the full SAR algorithm with a maximum rate of 128.4 GFLOPS during the azimuth processing. The bus and memory architectures of these multi-core processors play an important role in determining their applicability to a wide variety of problems. Making proper use of these components has a large impact on achieving performance close to the published theoretical maximums of the architectures. Coding for these architectures can easily obfuscate these issues and direct the effort away from exploiting the hardware and towards simply getting the code working. Using Gedae to address these issues in a structured and systematic way frees the developer to focus on making the best use of the hardware, while still achieving high performance with low overhead.

Get Started Today

Gedae and IBM have collaborated on a quick start package that includes a PS3 pre-loaded not only with the Gedae software, but with the IBM SDK and Fedora Core Linux OS.

1247 N Church Street, STE 5, Moorestown, NJ 08057 USA
www.gedae.com; gedae@gedae.com; 1-856-231-4458
Gedae Inc.