Implementation of Polar Format SAR Image Formation On the IBM Cell Broadband Engine

Jeffrey A Rudin
Mercury Computer Systems, Inc.
jrudin@mc.com

Introduction

The advent of multicore processors to address the ever increasing requirements for processing speed and density has led to increased programming difficulty when implementing defense applications. In particular, efficient adaptation of the IBM Cell Broadband Engine (CBE), which was developed for the video gaming industry, to defense applications can be uniquely challenging.

The IBM CBE is essentially a distributed memory, multiprocessing system on a single chip. It consists of a ring bus that connects a single PowerPC Processing Element (PPE), eight Synergistic Processing Elements (SPE), a high-bandwidth memory interface to the external XDR main memory, and a coherent interface bus to connect multiple Cell processors together.

In order to examine the challenges of using the Cell processor in detail, and to develop tools and methodologies to increase the programmability and computational efficiency of the Cell processor, Mercury Computer Systems performs analyses and develops benchmarks focused on defense applications. One particular application, which has benefited from high-performance computing technology is Synthetic Aperture Radar (SAR) image formation processing.

One well known method of SAR image formation is the Polar Format Algorithm (PFA). It is a simple algorithm whose processing steps are representative of those contained in other Fourier-based image formation techniques.

PFA consists of interpolation, corner-turns, and range and azimuth compression. While one may hypothesize that the compression algorithms may be the most computationally intense portion of the algorithm, it is the interpolation algorithm that can dominate the processing cycle. For this reason, the implementation of the interpolation algorithm was the particular focus of this effort. (Benchmarking of corner-turns and compression were accomplished in a separate study.)

Challenges to implementing PFA on the CBE include efficiently partitioning and moving the data and the tasks across the SPEs while maintaining a high computational efficiency through the use of efficient instruction pipelining, register use, and exploitation of the SIMD processing engines.

Implementation

The test platform was a Mercury 3.2 GHz Dual Cell-Based Blade; however only one of the Cell processors was used during the benchmarking of the algorithm. The 1 GB of XDR memory on the test platform was allocated to provide two complete image buffers. Given that an 8k x 8k image requires 512 MB for single precision, complex floating-point storage, the upper limit of the benchmarking was set to 50 Mpixel to allow for instruction code. Half of the SPE's local store memory of 256 KB was allocated to instructions leaving sufficient memory for two, 8 kpt. complex floating-point buffer.

The algorithm was implemented using a "function-offload" programming model. In this model, the PPE acts as a manager directing the work of the SPEs. Sections of the algorithm are loaded into the SPEs as individual "tasks." Data is then moved as "tiles" to the SPE where it is processed. Utilization of the SPEs is maximized by minimizing the number of trips that the data must take to the XDR memory.

The interpolation algorithm requires a non-uniform 2D raster of interpolation grid values. It was assumed that these values are one-time uses because the flight geometries for any two images are not exactly alike and must be recomputed for each image. These may be computed either on the fly during the tiling process, or precomputed and stored in memory. This latter approach was

not used because the problem with precomputing the interpolation grid is that it requires additional storage in XDR memory and additional data movement between the XDR memory and the SPE's local store.

Interpolation was accomplished by FFT/zero-pad/IFFT interpolation and nearest-neighbor selection. The FFT upsample process requires that there is at least one overlapping point between every two tiles. Additional overlapping is required minimize the effects of the Gibbs phenomena that occurs near the edge of the transformed data. The size of the SPE's local store memory directly affects the efficiency of the interpolation algorithm due to this minimum tile overlap requirement – the smaller the tile the larger the relative overhead and the lower the efficiency. Additionally, the overlap requirement is quantized to meet the 128-byte memory alignment restriction for efficient DMA. Given a buffer size of 8k, and an upsample factor of 8, this would limit the input tile size to 1 kpt. However, auxiliary buffer requirements reduce this to 512 pts., including a before and after overlap of 32 points each.

After a tile has been received by an SPE, the k-space boundaries and interpolation grid points are computed for each tile once it is sent to the SPE. The computation of the interpolation grid values was vectorized to take advantage of the SIMD computation engines within the SPEs in order to boost efficiency.

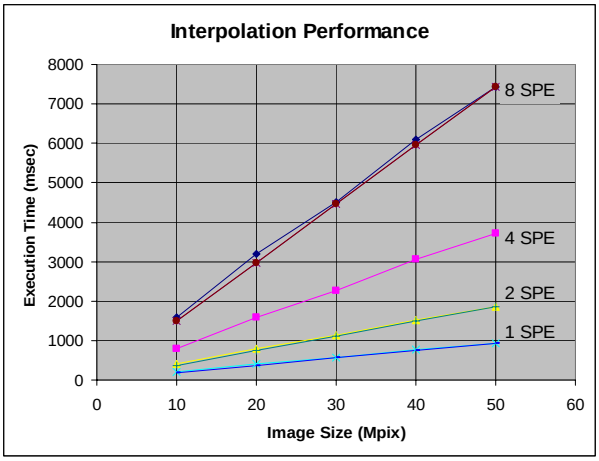
Because the interpolation grid is non-uniform, and because the number of output points is permitted to be independent from the number of input points, the scaling and offset factor between the raw and interpolated data may be different. This implies that a particular input tile may produce zero, one, or more than one output tiles. This alters the typical get-tile/put-tile loop that the SPEs would normally execute in a simple image processing operation and requires the use of a nested input-output loop structure.

In addition, the memory alignment that is required for efficient DMA requires that the data in the output buffer be properly aligned, which requires additional data movement within the SPE that is an overhead factor.

Data-flow management was accomplished through the use of Mercury's MultiCore Framework (MCF). This middleware permitted the dataflow to be managed through the use of the "tile channel" construct, which automatically partitions and synchronizes the scatter-gather dataflow between the main XDR memory and the SPEs' local store memory. Processing on the SPEs was accelerated through the use of Mercury's Scientific Algorithm Library (SAL) in order to exploit the SIMD engines on the SPEs.

Results

The benchmark was run for various image sizes and numbers of SPEs to examine scalability. The image sizes were varied independently in both range and azimuth. The results show a high degree of scalability of the algorithm. Processing of a 10k x 10k image was estimated to take approximately 1/17 of the aperture collection time. Future work will include further optimization of the code and investigation of larger image sizes.



Processing time estimate for 10K x 10K image

Synthetic Aperture Time (sec): 73.86 sec.

Range Interpolation: 1.86 sec	*(53.8 Mpixel/sec)*
Cornerturn: 0.11 sec	*(909 Mpixel/sec)*
Azimuth Interpolation: 1.86 sec	*(53.8 Mpixel/sec)*
Cornerturn: 0.11 sec	*(909 Mpixel/sec)*
Range Compress: 0.08 sec	*(1250 Mpixel/sec)*
Cornerturn: 0.11 sec	*(909 Mpixel/sec)*
Azimuth Compress: 0.08 sec	*(1250 Mpixel/sec)*

Total processing time: 4.21 sec. (23.8 Mpixel/sec)