Parallel Implementation of Irregular Terrain Model on IBM Cell Broadband Engine

Yang Song∗, Jeffrey A Rudin† and Ali Akoglu∗
∗Department of Electrical and Computer Engineering
The University of Arizona, Tucson, Arizona USA 85721
Email: {yangsong,akoglu}@ece.arizona.edu
†Mercury Computer Systems, Inc., USA
Email: rudin@mc.com

Abstract
Prediction of radio coverage, also known as radio “hear-ability” requires the prediction of radio propagation loss. The Irregular Terrain Model (ITM) predicts the median attenuation of a radio signal as a function of distance and the variability of the signal in time and space. Algorithm can be applied to a large amount of engineering problems to make area predictions for applications such as preliminary estimates for system design, surveillance, and land mobile systems. When the radio transmitters are mobile, the radio coverage changes dynamically, taking on a real-time aspect that requires thousands of calculations per second, which can be achieved through the use of recent advances in multicore processor technology. In this study, we evaluate the performance of ITM on IBM Cell Broadband Engine (BE). We first give a brief introduction to the algorithm of ITM and present both the serial and parallel execution manner of its implementation. Then we exploit how to map out the program on the target processor in detail. We choose message queues on Cell BE which offer the simplest possible expression of the algorithm while being able to fully utilize the hardware resources. Full code segment and a complete set of terrain profiles fit into each processing element without the need for further partitioning. Communications and memory management overhead is minimal and we achieve 90.2% processor utilization with 7.9x speed up compared to serial version. Through our experimental studies, we show that the program is scalable and suits very well for implementing on the CELL BE architecture based on the granularity of computation kernels and memory footprint of the algorithm.

Table 1. Ranges of the parameters in ITM.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency ($f_0$)</td>
<td>20 MHz ~ 40 GHz</td>
</tr>
<tr>
<td>antenna heights ($h_{a1}$, $h_{a2}$)</td>
<td>0.5 m ~ 3,000 m</td>
</tr>
<tr>
<td>distance ($d$)</td>
<td>1 km ~ 2,000 km</td>
</tr>
<tr>
<td>surface refractivity ($N_s$)</td>
<td>250 ~ 400 N units</td>
</tr>
</tbody>
</table>

1. Introduction
Irregular Terrain Model (ITM), also known as the Longley-Rice model [1], was developed by the U.S. Department of Commerce NTIA/ITS, Institute for Telecommunication Sciences in order to compute radio propagation loss with frequencies 20 MHz and above over irregular terrain. The model is based on electromagnetic theory and on statistical analysis of both terrain features and radio measurements. The model includes the effects of free-space propagation, reflections from the ground, refraction by the atmosphere, diffraction by obstacles, and atmospheric scattering. It predicts the median attenuation of a radio signal as a function of distance and the variability of the signal in time and space. The model may be applied either with point-to-point mode or area mode. The area mode is used in the event that the exact terrain elevations between the transmitting and receiving sites are not known. It uses a statistical estimate of terrain roughness to predict the propagation loss. In the point-to-point mode, the actual terrain elevations between the transmitting and receiving sites (terrain profile) are taken into account. This mode uses both diffraction theory and empirical formulae to provide a refined prediction of propagation loss.

The ITM model determines a radio link by two terminals in some region on the earth, hence the input required by the model should be a detailed description of that link. General input parameters include: distance between the two terminals ($d$), a terrain profile of the propagation path ($p_f$), transmitter and receiver antenna heights ($h_{a1}$, $h_{a2}$), frequency ($f_0$), wave number ($k$), terrain irregularity parameter ($\Delta h$), surface refractivity ($N_s$), the earth’s curvature ($\gamma_e$) and transfer impedance of the ground ($Z_g$). For the area mode, the additional input is the siting criteria indicating how to take care of the terminal to assure good radio propagation conditions. For the point-to-point mode, more specific numerical values should be provided, such as antenna effective heights ($h_{e1}$, $h_{e2}$), distances from each terminal to its radio horizon ($d_{l1}$, $d_{l2}$) and the horizontal angles of each terminal ($\theta_{e1}$, $\theta_{e2}$). Ranges of some of the above parameters are illustrated in
Table 1 [2]. In addition, the point-to-point mode takes into account the terrain elevation profile between the transmitter and receiver.

There are several options for the user to choose as the output from the model. Median reference values of attenuation ($A_{\text{ref}}$), which are calculated as a function of distance, are the simplest of these forms. The second form of output, $A(q_T, q_L, q_S)$, gives the three-dimensional cumulative distribution of the attenuation, including the time, location and situations. In the actual environment, the radio transmitters usually vary in a wide range, which results in a more dynamical frequency coverage. Additionally, if we satisfy a real-time demand to obtain the reference attenuation, it would require hundreds and thousands of calculations per second. However, each profile entity is independent from the other, which enables us to potentially carry out the algorithm on a multicore processor by taking advantage of its parallel nature.

IBM Cell Broadband Engine (Cell BE) is an example of a multicore architecture [3–5]. It has eight high frequency specialized execution cores with pipelined SIMD (single instruction multiple data) capabilities, and a fast transfer architecture. Cell BE is specifically an attractive architecture for applications and computational kernels with highly vectorizable data parallelism, such as image and video processing, encryption [6, 7]. The processor is reported to achieve a theoretical peak performance of over 200 Gflops for single-precision FP (floating point) calculations and has a peak memory bandwidth of over 25 Gigabytes/s. These performance figures make Cell BE a promising architecture for developing ITM. In this paper we study the architecture features of CELL BE and identify a strategy to partition the workload of the ITM algorithm. We then develop a scalable and highly parallel version of the irregular terrain model on CELL BE. The core functions of the ITM library compile to less than 64 KB. Full code segment and a complete set of terrain profiles fit into each processing element without the need for further partitioning. Communications and memory management overhead is minimal and we achieve 90.2% processor utilization with 7.9x speed up compared to serial version. Our analysis shows that ITM is an ideal candidate for implementing on CELL BE.

The remainder of the paper is organized as follows. In Section 2, we present a brief summary of the Cell Broadband Engine processor. Section 3 describes the strategy to parallelize the source code of ITM and build on this to evaluate the performance on Cell BE. The experimental results and analysis are shown in more detail in Section 4. The main conclusions are presented in Section 5.

2. The Cell Broadband Engine Processor

The IBM Cell Broadband Engine is essentially a distributed memory, multiprocessing system on a single chip (Figure 1). 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 [8]. All these elements are connected with an on-chip Element Inter-
connect Bus (EIB). The SPEs are placed in equal distances around the bus. The first level instruction and data cache on the PPE are 32 KB and the L2 cache is 512 KB. From a software perspective, the PPE can be thought of as the “host” or “control” core, where the operating system and general control functions for an application are executed.

The eight SPEs are the primary computing engines on the Cell processor. Each SPE contains a Synergistic Processing Unit (SPU), a memory flow controller, a memory management unit, a bus interface and an atomic unit for synchronization mechanisms [9]. SPU instructions cannot access the main memory directly. Instead, they access a 256 KB local store (LS) memory, which holds both instructions and data. Programmer should keep all the codes and data within the size of LS and manage its contents by transferring data between off-chip memory and LS via mailboxes or direct memory access (DMA). This allows the programmer to overlap the computations and data transfer via double-buffering techniques [10].

2.1. Data Communication Mechanism

To make full use of all the computational power available on the Cell BE processor, data must be distributed and communicated between PPE and SPEs. The PPE interacts with the SPEs through Memory-Mapped Input/Output (MMIO) registers supported by the Memory Flow Controller (MFC) of each SPE. These registers are accessed by the associated SPE through its channel mechanism. The three primarily used communication techniques between the PPE and SPEs are mailboxes, signal notification registers, and DMA. Message queue is one example of mailboxes that exchange data in the unit of 32-bit. Two mailboxes (the SPU Write Outbound Mailbox and the SPU Write Outbound Interrupt Mailbox) are provided for sending messages from the SPE to the PPE. One mailbox (the SPU Read Inbound Mailbox) is provided for sending messages from the PPE to the SPE [11]. Mailboxes can also be used as a communications mechanism between SPEs.

In Cell BE, the PPE is often used as an application manager, assigning and directing work to the SPEs. A large part of this task is loading main storage with the data to be processed, and then notifying the SPE by writing to the SPU mailbox via a message queue. Since in the Longley-Rice propagation model, data transmission is of small amount, low-latency but also low-bandwidth, we choose message queues as the communication scheme between PPE and eight SPEs.

2.2. The Multicore Framework

We use Cell SDK Version 3.0 and all the software implementations are completed under MultiCore Framework (MCF) developed by Mercury Computer Systems Inc.. MCF is a software toolkit that provides precise control of fine-grained data distribution and assignment of processing resources on a multicore processor, while isolating developers from the hardware-specific details involved. MCF provides a simplified parallel-programming model and offers multiple levels of programming that trade-off efficiency with programming simplicity. This permits the implementer to concentrate his efforts on those parts of the algorithm with the highest value while achieving the real-time goals of:

- Minimizing latency
- Efficient resource utilization
- Overlapping computation and communication
- Controlling computational granularity
- Maximizing SPE computational efficiency

MCF uses a Function Offload Engine (FOE) model (Figure 2). 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 to the SPE where it is processed.

3. Methodology for Parallelizing ITM on Cell

The source code of ITM propagation model is publicly available from NTIA/ITS [12]. This is a series of subprograms which generally involves three consecutive steps, as shown in Figure 3: the preparation of parameters, the computation of the reference attenuation, and the selected statistics. Much of the input and output is shared through the three common structures, prop_type, propv_type, propa_type, which must be accessed directly by the user. In this section, we first show the dataflow of the original ITM algorithm in serial manner and then explore ways to map the model onto the Cell BE in order to parallelize the computing.

3.1. Algorithm Implementations

The ITM may be used in two separate modes of operation: the area mode and the point-to-point mode. Both modes
use very similar calling sequences and they are treated in parallel. In our experimental study, we mainly focus on the point-to-point mode.

The two top level functions called are the \textit{qlrps()} and \textit{qlrpfl()} functions. The \textit{qlrps()} function takes environmental parameters and computes a reference propagation loss and additional parameters that are independent of the terrain profile. The \textit{qlrpfl()} function then takes this data along with a given terrain profile and computes a refined propagation loss based on the actual terrain elevations between the transmitter and receiver. For each radio coverage contour \textit{qlrps()} is called only one once; \textit{qlrpfl()} is called for each terrain profile.

In the serial operational flow as shown in Figure 3(a), the algorithm iteratively reads one single profile each time to initialize the parameters and call the two functions \textit{qlrps()} and \textit{qlrpfl()}. Moreover, the execution involves at least three separate structures in each iteration. Apparently, this approach does not work quite efficiently on the parallel architectures due to excessive initialization overhead. The processor will spend too much time on the file reading and data access. Thus, in order to address such drawback, we suggest a highly parallel ITM execution manner. First we packetize those three structures into one main structure and include all the other related data required for the computation as well. Next, we re-organize the code flow so that the main program reads the whole input file only once at the beginning and stores the parameters into the "main structure". After that, \textit{qlrps()} function is called only once and \textit{qlrpfl()} is executed several times according to the number of the terrain profiles. Finally, the result of the refined propagation loss and other data modification will be written back to the "main structure". Figure 3(b) indicates the proposed parallel execution flow. In this way, we greatly reduce the initialization overhead and different profiles can be processed in parallel by different processors.

### 3.2. Parallelism on IBM Cell BE

Parallelization of any algorithm requires careful attention to the partitioning of tasking and data across the system resources. Each SPE has a 256 KB local store memory that is allocated between worker code and data. The worker
Figure 4. The parallel code flow on Cell. PPE transmits the data package to SPEs via one message queue and receives the results from SPEs by another.

code section must contain the top-level worker code, the MCF functions, and any additional library functions that are used. If the total amount of worker code is too large for the allocated memory, it may be loaded as several “plug-ins”. If the total amount of data exceeds the data allocation, it may be loaded down as “tiles”. (MCF contains plug-in and tile channel constructs to facilitate this as required.) The trade-off here is in increased code complexity.

The core functions of the ITM library compile to less than 64 KB. MCF adds up to 64 KB depending on the functions that are used. Rounding this up suggested that the worker code would somehow be greater than half of the available SPE memory. For a contour with one terrain profile taken ever 6 degrees of arc, 60 profiles would be required. If each profile would contain 200 points, with 4 bytes per point, this would amount to about 48 KB of data. These estimates suggested that each SPE could receive a full code segment and a complete set of terrain profiles without the need for further partitioning. (In fact, the total size of the worker code amounted to less than 176 KB, allowing for up to 80 KB of data to be loaded in each SPE as a single job.)

The terrain profiles are independent from each other, thus when calling qlrps() and qlrpfl(), the algorithm only performs computation on the current profile and does not require any information from other profile sets. The PPE often functions as a manager, handling data I/O, assigning tasks and scheduling the SPEs (workers). The PPE reads the profile input files, transmits data to different SPEs, and invokes SPEs to complete the computations with respect to independent profiles.

We choose message queues as the parallelism strategy on IBM Cell BE due to the short bandwidth and latency of data communication. We packetize and transmit “main_structure” in a message queue between PPE and SPEs. First, the manager and the workers all initialize the MCF network. Then the PPE feeds the worker code into the local store memory of each SPE and tells them to start. As part of the initialization, we dynamically set up two message queues, one is for PPE sending data to SPEs, and the other is backwards, for SPEs passing results towards PPE. After determining the parameters of “main_structure” from the input file, the manager puts it into one message queue and sets up a loop in which the PPE sends the message to SPEs separately. The manager then waits for the synchronization semaphore from the SPEs when they finish pulling the data into local store. Sequentially, those SPEs are busy processing the data in a concurrent manner. Whoever completes the computations first sends the results back to PPE by means of the other message queue. This process continues until all SPEs finish the computation. In the meantime, a semaphore is sent by PPE to tell all the SPEs to stop. The manager then deallocates memory, destroys the MCF network and terminates the program. The whole datapath is indicated in Figure 4.

4. Experimental Results and Analysis

In this work, we use a Cell Blade with one Cell BE running at 2.8 GHz, and 1GB of XDR RAM. The PPE runs Linux Fedora Core 5. Cell SDK Version is 3.0 and all the software are completed under MCF.

For the experimental case study, we test 64 terrain profiles on various numbers of SPEs, measure the execution time and examine scalability. Each terrain profile contained 157 points, using the data supplied by NTIA/ITS, in order to verify the numerical accuracy of the results. We divide the profiles in equal part according to the number of SPEs. We record the number of SPEs, the profile partitions, the total computation time and time per profile in Table 2. Results show 7.9 times speedup over serial version of the application if we employ 8 SPEs to work.

Figure 5 depicts the regression analysis and the fitting curve for timing data in Table 2. From the regression
Table 2. Total execution time of ITM running on Cell BE (2.8 GHz). The number of terrain profiles is 64.

<table>
<thead>
<tr>
<th>Number of SPEs</th>
<th>Number of Profiles Per SPE</th>
<th>Total Time (ms)</th>
<th>Time per profile (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64</td>
<td>4.079</td>
<td>63.73</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>2.042</td>
<td>63.81</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>1.023</td>
<td>63.94</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0.513</td>
<td>64.13</td>
</tr>
</tbody>
</table>

Figure 5. The regression analysis between the number of SPEs and the total running time. The data is taken from Table 2.

analysis, we see that the regression formula is Equation (1).

\[ y = 4.077 \times x^{-0.99} \]  \hspace{1cm} \text{(1)}

It should be noted that the power to \( x \) is -0.99, which is quite close to -1. Theoretically, as the number of SPEs increases, the execution time should decrease linearly. That means the relation between the number of SPEs and the total time should be an exact rectangular hyperbola. In other words, the power to \( x \) should be -1. But due to the overhead of the message transmission between PPE and SPEs, the actual time is a little larger than the expected, which means the more SPEs are employed, the more overhead time they will cost.

Further experimentation was performed to evaluate the scalability. In these experiments, each SPE was given a “job” containing one set of parameters and 64 terrain profiles, each containing 157 terrain elevation points. The benchmark was run for various job sizes and numbers of SPEs to examine scalability. We record the total jobs, the number of SPEs, the computation time for the whole job in Table 3. The results (Figure 6) show the execution times scale linearly with the number of jobs and inversely with the number of SPEs used.

An analysis of the PPE and SPE timing (Figure 7) was performed to examine framework overhead. The analysis shows that the time to perform one call to \( qlrps() \) and sixty calls to \( qlrpf() \), took 2.86 msec., and that the communications and memory management overhead took 0.31 msec. This amounts to a processor utilization of 90.2%.

Table 3. Total execution time of ITM running on Cell BE (2.8 GHz). The total number of terrain profiles is kept as a constant of 64. The job number varies from 8 to 1024. And we test the algorithm on 1, 2, 4 and 8 SPEs separately.

<table>
<thead>
<tr>
<th>Job Size</th>
<th>1 SPE</th>
<th>2 SPEs</th>
<th>4 SPEs</th>
<th>8 SPEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>32.6334</td>
<td>16.2942</td>
<td>8.1833</td>
<td>4.1463</td>
</tr>
<tr>
<td>16</td>
<td>65.2375</td>
<td>32.6274</td>
<td>16.3651</td>
<td>8.2962</td>
</tr>
<tr>
<td>32</td>
<td>130.3152</td>
<td>65.2942</td>
<td>32.7286</td>
<td>16.4243</td>
</tr>
<tr>
<td>64</td>
<td>261.0903</td>
<td>130.6327</td>
<td>65.4572</td>
<td>32.8869</td>
</tr>
<tr>
<td>128</td>
<td>522.1831</td>
<td>261.3085</td>
<td>130.9099</td>
<td>65.7171</td>
</tr>
<tr>
<td>256</td>
<td>1044.4126</td>
<td>522.6692</td>
<td>261.8178</td>
<td>131.4955</td>
</tr>
<tr>
<td>512</td>
<td>2088.8772</td>
<td>1045.3604</td>
<td>523.6359</td>
<td>262.9257</td>
</tr>
<tr>
<td>1024</td>
<td>4177.7861</td>
<td>2090.7440</td>
<td>1047.2919</td>
<td>525.7187</td>
</tr>
</tbody>
</table>

Figure 6. Parallel Longley-Rice Performance Timings. Each job incorporates 64 terrain profiles. Each profile consists of 157 terrain elevation points. The job size varies from 8 to 1024, and the profile number is kept as a constant. We run various jobs on various number of SPEs, 1, 2, 4 and 8 separately. The data comes from Table 3.

5. Conclusion

In this paper we present a parallel execution of a widely used irregular terrain model (Longley-Rice model). Based on the features of ITM model, we choose message queues as the parallelism strategy on Cell BE. We test 64 terrain profiles and measure the computation timings. Experimental results show that the program suits very well for the Cell BE architecture and achieves a 7.9 times speedup compared to the serial manner of execution. These results show that the IBM Cell Broadband Engine can be used effectively to accelerate applications whose code and data fit within the 256 KB of the SPE local store.
Figure 7. SPE timing trace analysis using TATL (Trace Analysis Tool and Library) [13]. One PPE and eight SPEs are running ITM. The red part on each worker line means the time that one SPE takes to perform one call to qlrps() and sixty calls to qlrpfl(), that is 2.86 ms. The blue part on each worker line represents the communications and memory management overhead, that is about 0.31 ms.

Acknowledgement

This work is in part funded by Sensor Visualization and Data Fusion Workstation Program of Battle Command Battle Lab-Huachuca, under Contract No.: W1SP7T-07-C-P046 as a sub contractor of Mercury Computer Systems, Inc.

The authors would like to thank Paul Cook, Jason Denno, and Jim Creutz for their input of the project.

References


