

Parallel Catastrophe Modelling on a Cell Processor

Frank Dehne¹, Glenn Hickey², Andrew Rau-Chaplin³ and Mark Byrne³

¹Carleton University, Ottawa, Canada, frank@dehne.net, <http://www.dehne.net>

²McGill University, Montreal, Canada, glenn.hickey@gmail.com

³Flagstone Reinsurance, Halifax, Canada, arau-chaplin@flagstonere.bm, <http://www.flagstonere.bm>

Abstract

In this paper we study the potential performance improvements for catastrophe modelling systems that can be achieved through parallelization on a Cell Processor. We studied and parallelized a critical section of catastrophe modelling, the so called “inner loop”, and implemented it on a Cell Processor running on a regular Playstation 3 platform. The Cell Processor is known to be a challenging environment for software development. In particular, the small internal storage available at each SPE of the Cell Processor is a considerable challenge for catastrophe modelling because the catastrophe modelling algorithm requires frequent accesses to large lookup tables. Our parallel solution is a combination of multiple techniques: streaming data to the SPEs and parallelizing inner loop computations, building caches on the SPEs to store parts of the large catastrophe modelling lookup tables, vectorizing the computation on the SPEs, and double-buffering the file I/O. On a (Playstation 3) Cell Processor with six active SPEs and 4-way vectorization on each SPE (implying a maximum theoretical 24x speedup), we were able to measure a sustained 16x speedup for our parallel catastrophe modelling code over a wide range of data sizes for real life Japanese earthquake data.

Copyright © 2009 Frank Dehne¹, Glenn Hickey², Andrew Rau-Chaplin³ and Mark Byrne³. Permission to copy is hereby granted provided the original copyright notice is reproduced in copies made.

1 Introduction

Over the past three decades, catastrophe modelling technology has become a vital tool for quantifying, managing, and transferring risk in the insurance industry. The first catastrophe models for the insurance markets were introduced in the late 1980s, focusing on event-specific probabilistic modelling to quantify risk for individual locations and for portfolios of aggregated risks. Today, catastrophe risk models are the standard for quantifying catastrophe risk in many regions and perils all over the world. They are key elements of risk management, as they enable insurers to examine accumulations of risk, measure and identify worst-case losses, assess relative risk across different geographic areas, and measure the probability of loss for property and lives. [4, 8, 5]

Natural catastrophe models are used to estimate monetary risk based on vulnerabilities of specific properties and their residents to perils including hurricanes, earthquakes, severe thunderstorms and winter storms. Catastrophe models compute consequences for single events and also compute a probabilistic loss distribution based on frequency estimates derived from historical data. The initial use of these models was for the insurance industry and financial markets to quantify risk to portfolios and to manage such risk. Companies in diverse industries, as well as government organizations, now use the models to estimate total national

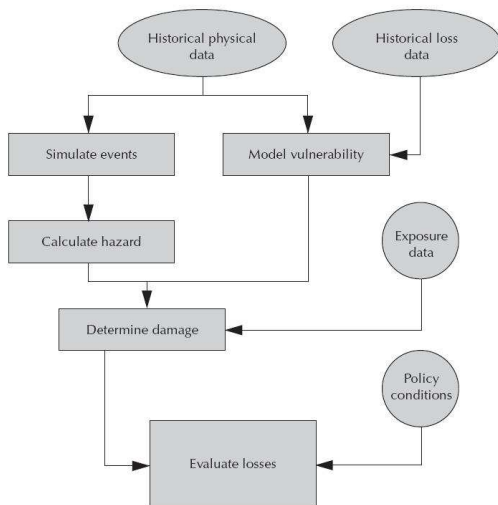


Figure 1: Catastrophe Risk Modelling Procedure.

risks. An outline of a catastrophe risk model for natural hazards is shown in Figure 1. The procedure starts with thorough accumulation, study and modelling of historical natural hazards in a region. The process then randomly draws characteristics from a statistical study on event characteristics and generates simulated artificial events within the region with the same probabilistic characteristics. For each of the stochastically simulated events, local hazard intensity is calculated at the site of a given asset. Through statistical and engineering examination of building responses at times of catastrophes, a vulnerability model is developed. Using the local hazard intensity and vulnerability model, the extent of physical and monetary damage is calculated for each and every asset in a risk portfolio. The monetary damage is pushed through a financial model, leading to the calculation of the financial losses.

Speed matters in catastrophe modelling systems because it defines how much work can be produced in a given time budget. Speed improves quality in that a fast catastrophe modelling engine allows the designers of region-peril models more cycles in a given window of time to generate higher resolution results. More precisely, catastrophe modelling systems benefit from increased modelling speed in the following ways [2]. Improved speed allows running

the catastrophe simulation at a more detailed level to better take advantage of available exposure data. Improved speed allows incorporating better physical modelling and the latest science to improve model accuracy. It allows to increase model confidence by running multiple scenarios in the same amount of time. Faster turnaround of analysis due to improved running times can also have substantial business advantages when multiple reinsurance companies are competing for a contract and time is essential which is often the case. All of these are potential benefits for the users of catastrophe models, but speed also has a role to play in model development because it can significantly shorten the calibration phase during model development, which typically lasts anywhere from 6 to 12 months.

In this paper, we report on the results of a research project that studied the question of how much a catastrophe modelling system’s performance can benefit from being parallelized and executed on a Cell Processor. To our knowledge, this is the first such study. We studied and parallelized a critical section of catastrophe modelling, the so called “inner loop”, and implemented it on a Cell Processor running on a regular Playstation 3 platform. The large aggregate computational power of the Cell Processor is our main motivation for studying its use to improve the performance of catastrophe modelling. However, the Cell Processor is known to be a challenging environment for software development. In particular, the small internal storage available at each SPE of the Cell Processor is a considerable challenge for catastrophe modelling because the catastrophe modelling algorithm requires frequent accesses to large lookup tables. Our parallel implementation of the inner loop is a combination of multiple techniques: streaming data to the SPEs, building caches on the SPEs to store parts of the large catastrophe modelling lookup tables for faster table lookup, vectorizing the computation on the SPEs, and double-buffering the file I/O. On a *Playstation 3* Cell Processor with six active SPEs and 4-way vectorization on each SPE (implying a maximum theoretical 24x speedup), we were able to measure a sustained 16x speedup for our parallel code over a wide range of data sizes for real life Japanese

earthquake data.

The remainder of this paper is organized as follows. In the following Section 2 we discuss some features of the Cell Processor that are important for this project. Section 3 outlines our parallel catastrophe modelling method for the Cell Processor as well as some implementation details. Section 4 presents experimental results showing the performance of our parallel software and Section 5 concludes the paper.

2 The Cell Processor

The Cell Processor (also called “Cell BE” for “Cell Broadband Engine Architecture”) is a microprocessor architecture jointly developed by Sony Computer Entertainment, Toshiba, and IBM. The Cell Processor emphasizes efficiency/watt, prioritizes bandwidth over latency, and favors peak computational throughput over simplicity of program code. The vastly superior computation speed of the Cell Processor (220 GFlops as compared to e.g. 48 GFlops for a 3.0 GHz Intel Core2 Duo) is the main motivation for studying its use to improve the performance of catastrophe modelling. The Cell Processor is however widely regarded as a challenging environment for software development. IBM provides a Linux-based Cell development platform but software adoption remains a key issue on whether Cell ultimately delivers on its performance potential. The Cell BE consists of four components (see Figure 2): the external I/O interface, the main processor called the Power Processing Element (PPE) (a two-way simultaneous multithreaded Power compliant core), eight fully-functional co-processors referred to as Synergistic Processing Elements, or SPEs, and a high-bandwidth circular data bus connecting the PPE, input/output and the SPEs, referred to as the Element Interconnect Bus or EIB. The SPEs do not support multithreading but can perform load, store, shuffle, channel or branch operation in parallel with a computation. They have a reduced SIMD-RISC instruction set, a 128-entry 128 bit unified register file for all data types, and 4 way SIMD vector capability. SPEs can complete up to two instructions per cycle but have no branch prediction logic in hardware. Instead,

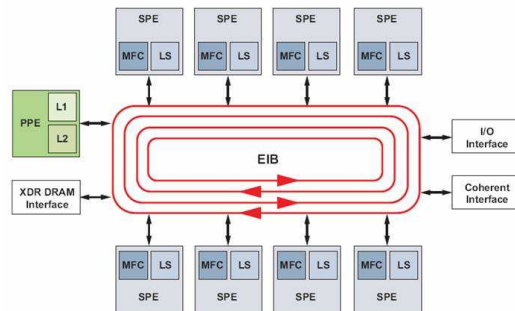


Figure 2: Cell Processor Diagram. (Note: Only six SPEs were available on our Playstation 3 platform).

they require software controlled branch prediction through branch hint instructions. Each SPE’s memory consists of a 256KB local store with 6 cycle load latency. The application software must manage data *in* and *out* for the local store. Memory access is controlled by the memory flow controller (MFC). An SPE can use the DMA controller to move data to its own or other SPEs local store and between its local store and main memory as well as I/O interfaces. The Memory Flow Controller (MFC) on each SPE is local to the SPE and connects it to the EIB. The MFC runs at the same frequency as the EIB and can begin to transfer the data set of the next task at the same time as the present one is still running (double buffering). For our project, we used a Playstation 3 platform with Yellow Dog Linux 6.1 installed plus the RapidMind software development toolkit [1]. Note that in the Playstation 3, one SPE is disabled to increase manufacturing yield and another is reserved by the native OS, leaving six SPEs available for Linux applications.

3 Parallel CAT Modelling on a Cell Processor

The input to the catastrophe (CAT) modelling software is a catalogue of events (such as earthquakes) and a set of locations (such as commercial and residential structures). The information provided for each event includes the geographic coordinates of its bounding box along with a grid of PGV (peak ground velocity) val-

		Events							
Locations									

Figure 3: Event-Location Matrix.

ues, essentially representing the magnitude of the event in the area covered by each grid cell. The locations are represented by point coordinates and have associated values, insurance policy information and structural properties. The algorithm processes this data on an event-by-event basis, by first performing a GIS query to determine the affected locations then computing the expected loss at each location by the current event. The losses are then aggregated for each event and reported as an event-location matrix (Figure 3).

Estimating earthquake damage is an extremely complex task that requires a large amount of scientific computing, most of which is performed offline and stored in lookup tables. These tables are used to build probability distributions from the input data, accounting for some of the uncertainty due to rounding and sampling. The distributions transform a single value into a weighted series of values which must be processed independently and then summed, resulting in a high volume of data that must be processed. Algorithm 1 provides a high level overview of this procedure for a single event-location (EL) computation.

Algorithm 1 “Inner Loop” for a single *event location* computation.

- (1) Look up the distribution of peak ground velocity (PGV) values for the event at this location.
- (2) FOR each point i in the PGV distribution DO
 - (3) Look up corresponding distribution of location vulnerability values.
 - (4) FOR each point j in the vulnerability distribution DO
 - (5) Compute expected loss for (i, j) and weight according to j .
 - (6) Add to total loss for i .
 - (7) END FOR

- (8) Weight the total loss for i according to i 's value in the in the PGV distribution and add it to the total loss.

(9) END FOR

- (10) **return** total loss.

— End of Algorithm —

3.1 Streaming Data to the Cell SPEs

We used the RapidMind API [1] to first port the existing sequential code to the Cell processor’s PPE and then to make a series of optimizations in order to take full advantage of the SPEs. The first optimization was to write an SPE version of the code described in Algorithm 1 that computes the expected loss from a single Event-Location (EL). Since the program is executed in SPMD mode, the chief challenge was to limit control flow that would reduce performance. This was accomplished by aggressive loop unrolling and refactoring the code to obviate nested if/else statements.

We then re-designed the PPE and SPE code to stream the ELs from the PPE to the SPEs 16 ELs (approx 2KB) at a time, using double-buffering. This number of ELs was chosen because it gave the best performance on our data, in that the time it took to process 16 ELs closely matched the time to pass them from the PPE to SPE using a DMA transfer. The expected loss values were streamed back to the PPE in a similar manner. By carefully choosing the double buffer size to overlap communication with processing, we were able to hide most of the communication overhead.

3.2 Caching

A principal concern at the outset of this project was the size of the SPE local store on the PS3, which is 256k. Since the loss computations make heavy use of precomputed (offline) lookup tables through random access, and these tables are too large to fit in the local store, performance may be lost when SPEs must wait to obtain the data from main memory. The lookup tables used for ground-up loss computation and their sizes are listed in Table 1. Note that, the local store of each SPE must store both, code and data, and only a small portion of the

Table Name	Dimensions	Table Size
<i>Ground Motion</i>	PGV x Dist. Index x Event Type	25.9 KB
<i>Ground Motion Dist.</i>	PGV x Index Point x Event Type	25.9 KB
<i>Vulnerability</i>	Coverage Index x PGV	154.7 KB
<i>Vulnerability Dist.</i>	MDR Index x Dist. Index x Coverage Type (4)	170.0 KB
Total Size: 376.5 KB		

Table 1: Precomputed look-up tables required for computation of ground-up loss. Full financial model requires several additional tables. PGV = Peak Ground Velocity. MDR = Mean Damage Ratio.

256k is available for the look-up tables. Since the locations are spatially sorted, it is reasonable to expect that the *expected* access patterns into the tables will not be completely random. Indeed, we observed that the ground velocities, coverage types and event types tend to be similar between points that are geographically close. We therefore use a caching mechanism that keeps a small local cache of the recently accessed table entries on each SPE. As shown in Section 4, this strategy proved to be extremely effective in reducing the overhead for accessing the look-up tables.

3.3 Vectorization on the Cell SPEs

The above version of the Cell Processor code only stores one value per register on the SPE. However, the Cell Processor’s SPE has 128 bit vector registers. The full power of the Cell can only be harnessed if the code is adapted to use each register as a 4-value vector. Therefore, we changed each operation on the SPE

to operate on 4-tuples instead of values. The control flow involved in computing ground-up loss is loop-based and the majority of the code could be efficiently vectorized. Vectorizing the computation of offsets in the various look-up tables was slightly more challenging since table lookups are usually not accessing adjacent values. Another challenge, from a software engineering standpoint, was to transform the data that was passed from the PPE to the SPEs as *objects* into four-value vectors. We chose to re-implement the necessary interfaces as classes whose data members are all defined as offsets into an array of 4-tuples (either integers or floats). In this manner, we were able to keep the SPE code object oriented by using the same class interfaces as the PPE code, but could lay the data out on the PPE such that the data from four objects could be properly striped across vectors. While this took some development time, the overhead in terms of computation was minimal and we expect that this approach is useful in general when porting object oriented code to the Cell Processor.

3.4 Double-Buffering for File I/O

For the Cell Processor, high volumes of data must be constantly supplied to take advantage of the processing power of the SPEs. Therefore, we were concerned that the SPEs were possibly wasting cycles while waiting for the next batch of event locations to be read from the disk. To address this, another layer of double-buffering was added to the code, this time to prefetch the next batch of event locations from the disk as the current batch is being processed. In fact, since the PPE is dual-threaded, reading and processing data sent to and from the hard disk can be done completely in parallel, together with a process that is responsible for marshaling and dispatching the SPEs.

4 Experimental Evaluation

To test our Cell Processor implementation, a data set of 500 earthquakes (events) with magnitudes greater than 7 on and around the island of Japan was generated (Figure 4). Seven batches of randomly generated exposures (locations in this region) ranging in size from 100

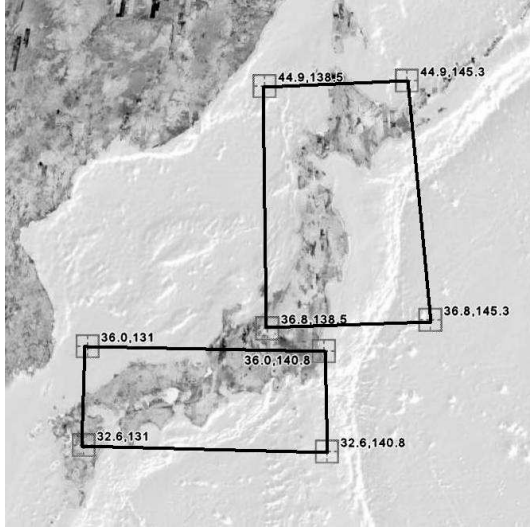


Figure 4: The earthquakes and exposures used as input in this study were generated within two bounding boxes around the island of Japan.

to 10,000 were created. For the largest input, 500 events by 10,000 locations, roughly 1.6 million out of a possible 5 million event-locations were processed because not every building was affected by every earthquake. These 1.6 million event-locations were used as input data to evaluate the performance of our implementation.

The program was benchmarked on this data set after each optimization stage described in the previous section. We measured the total wall clock time of the “inner loop” (event-location computations) for the sequential code running on the PPE and the parallel code using the six SPEs after each optimization described in the previous section.

The results for processing the entire dataset are given in Table 2. The “PPE Sequential” row shows the sequential time on the PPE. The “PPE & 6 SPEs” row shows the parallel time and speedup for the parallel code version implementing the streaming of data to the SPEs (Section 3.1) and caching (Section 3.2) approaches. The speedup obtained for the event-location computations is 5.04, 84% of the theoretical maximum of 6 (number of available SPEs). This is a very positive result which shows that relatively little performance is lost to table look-ups and communication overhead,

due to the effectiveness of the local caching and double buffering respectively. The “PPE & 6 SPEs + Vectorized” row in Table 2 shows the parallel time and speedup when vectorization (Section 3.3) is added. The speedup over the “PPE & 6 SPE” version for the EL computation time is approx. 3.3 or 82.5% of the theoretical maximum of 4. It shows that our vectorization effort was very successful. The “PPE & 6 SPEs + Vectorized + I/O double buffer” row in Table 2 shows the parallel time and speedup when I/O double buffering (Section 3.4) is added. Unfortunately, it did not yield any measurable decrease in running time due to the difficulty of finding an optimal buffer size. While the disk I/O time remains constant, the computation time of the inner-loop is highly data dependent and hard to predict. As a consequence, any buffer size chosen will be either too large or too small for a large portion of the event-location computations, potentially introducing as much idle time for the PPE as saving time for the SPEs through latency hiding. We tried many buffer sizes without much success and ended up using a size of 512 event-locations (59392 bytes) which offered the best average performance for our data.

Figure 5 shows the same wall clock times as Table 2 but for different data sizes (number of event-locations). We observe that the running times for all versions of our code grow linearly with respect to the input size. Figure 6 shows the same speedup values as Table 2 but again for different data sizes (number of event-locations). We observe that a speedup of 16 is achieved from approx. 400,000 event-locations onwards, and then remains very steady at that value, growing slowly to approx. 16.5.

5 Conclusions

In this paper we studied and parallelized a critical section of catastrophe modelling, the so called “inner loop”, and implemented it on a Cell Processor running on a regular Playstation 3 platform. The Cell Processor is known to be a challenging environment for software development. Our parallel solution is a combination of multiple techniques: streaming data to the SPEs, building caches on the SPEs to store

PPE Sequential
Wall Clock Time: 1282.56 s
PPE & 6 SPEs
Wall Clock Time: 254.59 s
Speedup: 5.04
PPE & 6 SPEs + Vectorized
Wall Clock Time: 77.74 s
Speedup: 16.50
PPE & 6 SPEs + Vectorized + I/O Double Buffer
Wall Clock Time: 77.74 s
Speedup: 16.50

Table 2: Speedup achieved for computing all 1.6M event locations from the Japanese earthquake data set on a PlayStation 3 (with six active SPEs).

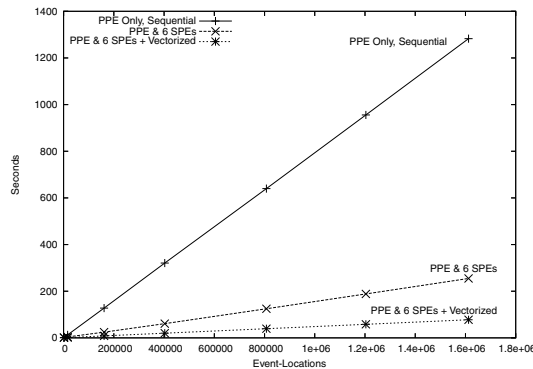


Figure 5: Wall clock time (seconds) for computing event-locations from the Japanese earthquake data set on a PlayStation 3 as a function of the number of event-locations.

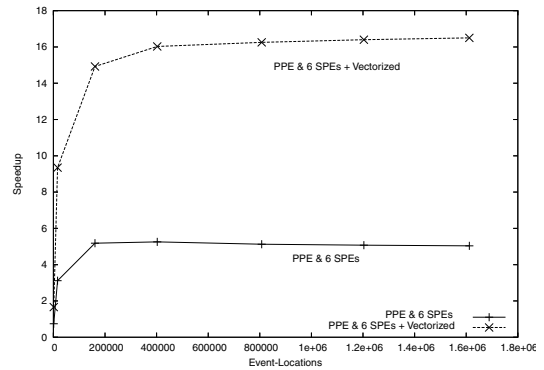


Figure 6: Speedup achieved for computing event-locations from the Japanese earthquake data set on a PlayStation 3 as a function of the number of event-locations.

parts of the catastrophe modelling lookup tables, vectorizing the computation on the SPEs, and double-buffering the file I/O. While vectorization and caching had a significant positive impact on performance, double-buffering of the file I/O did not yield much improvement. On a (Playstation 3) Cell Processor with six active SPEs and 4-way vectorization on each SPE, our parallel system provided for real life Japanese earthquake data a sustained 16x speedup, 66% of the maximum theoretically possible 24x speedup.

References

- [1] RapidMind software development toolkit. <http://www.rapidmind.net>.
- [2] M. Byrne, N.G. Kishi, and A. Rau-Chaplin. *Asian Catastrophe Insurance*, chapter Innovating Technology for Catastrophe Risk Modelling and Reinsurance.
- [3] T. Chen, R. Raghavan, J. N. Dale, and E. Iwata. Cell broadband engine architecture and its first implementation - a performance view. *IBM Journal of Research and Development*, 51(5):559–572, 2007.
- [4] C. Scawthorn (Ed.). Special issue: Multi-hazards loss estimation and HAZUS. *Natural Hazards Review*, 7(2), 2006.

- [5] P. Grossi and H. Kunreuther, editors. *Catastrophe Modeling: A New Approach to Managing Risk*. Springer Verlag, 2005.
- [6] C. R. Johns and D. A. Brokenshire. Introduction to the cell broadband engine architecture. *IBM Journal of Research and Development*, 51(5):503–519, 2007.
- [7] M.W. Riley, J.D. Warnock, and D.F. Wendel. Cell broadband engine processor: Design and implementation. *IBM Journal of Research and Development*, 51(5):545–558, 2007.
- [8] C. Scawthorn. *Risk Assessment, Modeling and Decision Support*, chapter A Brief History of Seismic Risk Assessment. Springer Verlag, 2006.