International Journal of Parallel, Emergent and Distributed Systems
Vol. 25, No. 5, October 2010, 401–410

Parallel catastrophe modelling on a Cell/BE

Frank Dehne[a]*, Glenn Hickey[b1], Andrew Rau-Chaplin[c2] and Mark Byrne[c]

[a]*Carleton University, Ottawa, Canada;* [b]*McGill University, Montreal, Canada;* [c]*Flagstone Reinsurance, Halifax, Canada*

(*Received 15 July 2009; final version received 16 November 2009*)

In this paper, we study the potential performance improvements for catastrophe (CAT) modelling systems that can be achieved through parallelisation on a cell processor [Cell Broadband Engine Architecture (Cell/BE)]. We studied and parallelised a critical section of CAT modelling, the so-called inner loop and implemented it on a Cell/BE running on a regular Playstation 3 platform. The Cell/BE is known to be a challenging environment for software development. In particular, the small internal storage available at each synergistic processing element (SPE) of the Cell/BE is a considerable challenge for CAT modelling because the CAT-modelling algorithm requires frequent accesses to large lookup tables. Our parallel solution is a combination of multiple techniques: streaming data to the SPEs and parallelising inner loop computations, building caches on the SPEs to store parts of the large CAT modelling lookup tables, vectorising the computation on the SPEs and double-buffering the file I/O. On a (Playstation 3) Cell/BE with six active SPEs and four-way vectorisation on each SPE, we were able to measure a sustained 16 × speedup for our parallel CAT-modelling code over a wide range of data sizes for real-life Japanese earthquake data.

Keywords: parallel algorithms; catastrophe modelling; cell processor

1. Introduction

Over the past three decades, catastrophe (CAT)-modelling technology has become a vital tool for quantifying, managing and transferring risk in the insurance industry. The first CAT 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, CAT risk models are the standard for quantifying CAT 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 [3,6,7].

Natural CAT 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. CAT 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 organisations, now use the models to estimate

*Corresponding author. Email: frank@dehne.net

ISSN 1744-5760 print/ISSN 1744-5779 online

DOI: 10.1080/17445760903492086
http://www.informaworld.com

total national risks. An outline of a CAT 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 CATs, 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 CAT-modelling systems because it defines how much work can be produced in a given time budget. Speed improves quality in that a fast CAT-modelling engine allows the designers of region-peril models more cycles in a given window of time to generate higher resolution results. More precisely, CAT-modelling systems benefit from increased modelling speed in the following ways [1]. Improved speed allows running the CAT 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 increasing model confidence by running multiple scenarios in the same amount of time. Faster turnaround of analysis due to improved

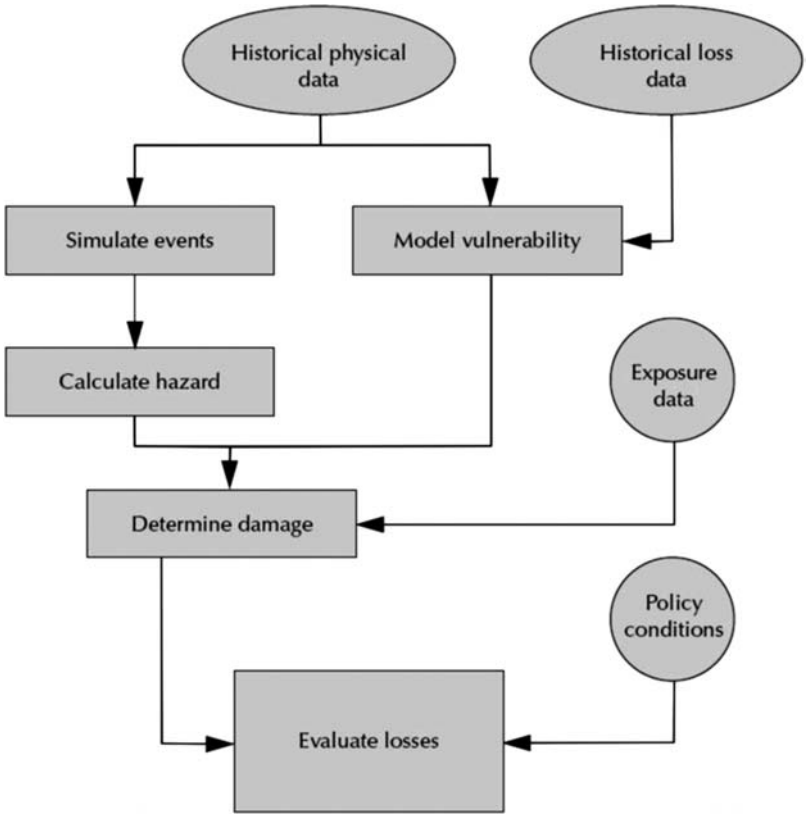

Figure 1. CAT risk modelling procedure.

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 CAT 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 CAT-modelling system's performance can benefit from being parallelised and executed on a Cell Broadband Engine Architecture (Cell/BE) [2,5]. To our knowledge, this is the first such study. We studied and parallelised a critical section of CAT modelling, the so-called inner loop, and implemented it on a Cell/BE running on a regular Playstation 3 platform. The large aggregate computational power of the Cell/BE is our main motivation for studying its use to improve the performance of CAT modelling. However, the Cell/BE is known to be a challenging environment for software development. In particular, the small internal storage available at each synergistic processing elements (SPE) of the Cell/BE is a considerable challenge for CAT modelling because the CAT-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 CAT modelling lookup tables for faster table lookup, vectorising the computation on the SPEs and double buffering the file I/O. On a Playstation 3 Cell/BE with six active SPEs and four-way vectorisation on each SPE, we were able to measure a sustained 16 × speedup for our parallel code over a wide range of data sizes for real-life Japanese earthquake data.

The remainder of this paper is organised as follows. In the following Section 2, we discuss some features of the Cell/BE which are important for this project. Section 3 outlines our parallel CAT modelling method for the Cell/BE 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/BE

The Cell/BE is a microprocessor architecture jointly developed by Sony Computer Entertainment, Toshiba and IBM. The Cell/BE emphasises efficiency/watt, prioritises bandwidth over latency and favours peak computational throughput over simplicity of program code. The vastly superior computation speed of the Cell/BE (204.8 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 CAT modelling. The Cell/BE is, however, widely regarded as a challenging environment for software development. IBM provides a Linux-based Cell/BE development platform but software adoption remains a key issue on whether Cell/BE ultimately delivers on its performance potential. The Cell/BE consists of four components (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 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 multi-threading. However, they can perform load, store, shuffle, channel or branch operation in parallel with computation operations. They have a reduced Single Instruction Multiple Data-Reduced Instruction Set Computer (SIMD-RISC) instruction set, a 128-entry 128-bit unified register file for all data types and four-way SIMD vector capability. SPEs can complete up to two instructions per cycle but have no branch prediction logic in hardware. Instead, they require software-controlled

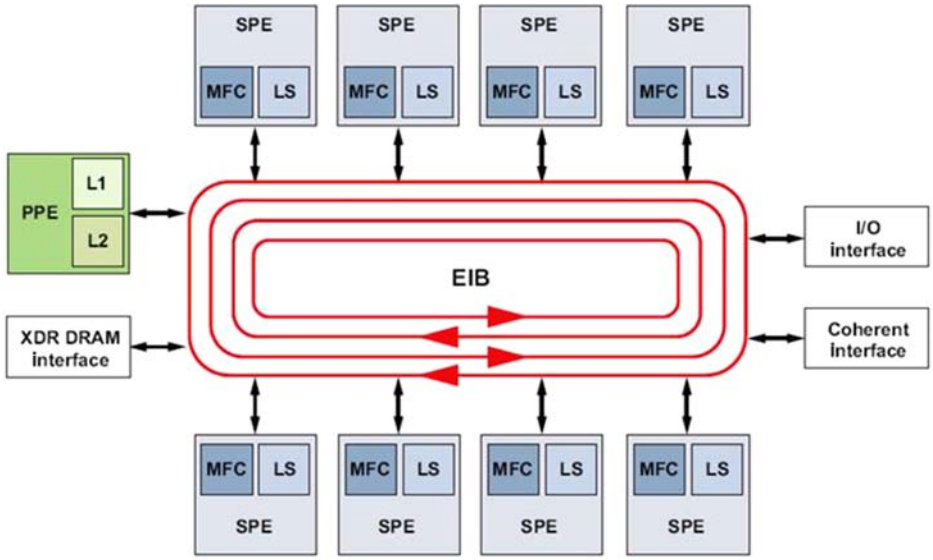

Figure 2. Cell/BE diagram from [4]. (Note: only six SPEs were available on our Playstation 3 platform.)

branch prediction through branch hint instructions. Each SPE's memory consists of a 256 KB local store with six 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 Direct Memory Access (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 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 dataset 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 (http://www.rapidmind.net). 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/BE

The input to the 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 peak ground velocity (PGV) values, 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 these data on an event-by-event basis, by first performing a Geographic Information System (GIS) query to determine the affected locations and 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 (EL) 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,

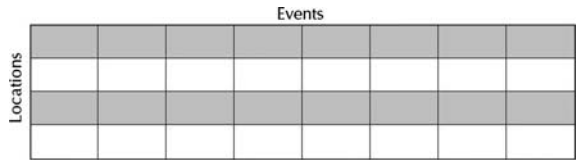

Figure 3. EL matrix.

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 EL computation.

Algorithm 1 'Inner Loop' for a single *EL* computation.

```
1:  Look up the distribution of PGV values for the event at this location.
2:  for all each point i in the PGV distribution do
3:      Look up corresponding distribution of location vulnerability values.
4:      for all 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 PGV distribution and add it to the
        total loss.
9:  end for
10: return total loss.
```

3.1 Streaming data to the cell SPEs

We used the RapidMind API (RapidMind software development toolkit, http://www.rapidmind.net) to first port the existing sequential code to the Cell/BE's PPE and then to make a series of optimisations in order to take full advantage of the SPEs. The first optimisation was to write an SPE version of the code described in Algorithm 1 that computes the expected loss from a single 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. 2 KB) 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 256 KB. Since the loss computations make heavy use of pre-computed (offline) lookup tables through random access, and these tables are too large to fit in the

Table 1. Pre-computed lookup tables required for computation of ground-up loss.

Table name	Dimensions	Size (KB)
Ground motion	PGV × dist. index × event type	25.9
Ground motion dist.	PGV × index point × event type	25.9
Vulnerability	Coverage index × PGV	154.7
Vulnerability dist.	MDR index × dist. index × coverage type (4)	170.0
Total		376.5

Notes: Full financial model requires several additional tables. PGV, peak ground velocity; MDR, mean damage ratio.

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 256 KB is available for the lookup 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 implemented our own caching mechanism which keeps a local cache of recently accessed table entries on each SPE using a FIFO replacement strategy. As shown in Section 4, this strategy proved to be extremely effective in reducing the overhead for accessing the lookup tables.

3.3 Vectorisation on the Cell SPEs

The above version of the Cell/BE code only stores one value per register on the SPE. However, the Cell/BE's SPE has 128-bit vector registers. The full power of the Cell/BE can only be harnessed if the code is adapted to use each register as a four-value vector. Therefore, we changed each operation on the SPE to operate on four tupler instead of values. The control flow involved in computing ground-up loss is loop based and the majority of the code could be efficiently vectorised. Vectorising the computation of offsets in the various lookup 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 were 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 four tupler (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/BE

3.4 Double buffering for file I/O

For the Cell/BE, 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 pre-zfetch 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 marshalling and dispatching the SPEs.

4. Experimental evaluation

To test our Cell/BE implementation, a dataset 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 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 ELs were processed because not every building was affected by every earthquake. These 1.6 million ELs were used as input data to evaluate the performance of our implementation.

The program was benchmarked on this dataset after each optimisation stage described in the previous section. We measured the total wall clock time of the 'inner loop' (EL computations) for the sequential code running on the PPE and the parallel code using the six SPEs after each optimisation 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

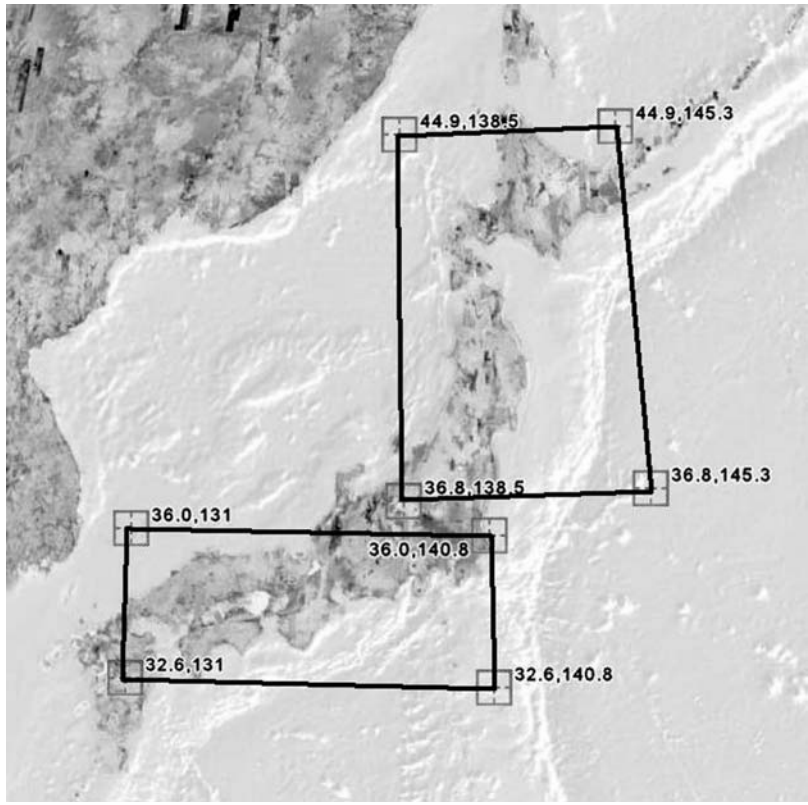

Figure 4. The earthquakes and exposures used as input in this study were generated within two bounding boxes around the island of Japan.

Table 2. Speedup achieved for computing all 1.6 M event locations from the Japanese earth quake dataset on a Playstation 3 (with six active SPEs).

Code version	Wall clock time (s)	Speedup
PPE sequential	1282.56	1
PPE & 6 SPEs	254.59	5.04
PPE & 6 SPEs + Vectorised	77.74	16.50
PPE & 6 SPEs + Vectorised + I/O double buffer	77.74	16.50

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 EL computations is 5.04. This is a very positive result which shows that relatively little performance is lost to table lookups and communication overhead, due to the effectiveness of the local caching and double buffering, respectively. The 'PPE & 6 SPEs+Vectorised' row in Table 2 shows the parallel time and speedup when vectorisation (Section 3.3) is added. The speedup over the 'PPE & 6 SPE' version for the EL computation time is approximately 3.3 or 82.5% of the theoretical maximum of 4. It shows that our vectorisation effort was very successful. The 'PPE & 6 SPEs + Vectorised + 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 EL 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 ELs (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 ELs). 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 ELs). We observe that a speedup of 16 is achieved from approximately 400,000 ELs onwards, and then remains very steady at that value, growing slowly to approximately 16.5.

5. Conclusions

In this paper, we studied and parallelised a critical section of CAT modelling, the so-called inner loop, and implemented it on a Cell/BE running on a regular Playstation 3 platform. The Cell/BE 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 parts of the CAT-modelling lookup tables, vectorising the computation on the SPEs and double buffering the file I/O. While vectorisation 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/BE with six active SPEs and four-way vectorisation on each SPE, our parallel system provided for real-life Japanese earthquake data a sustained 16 × speedup.

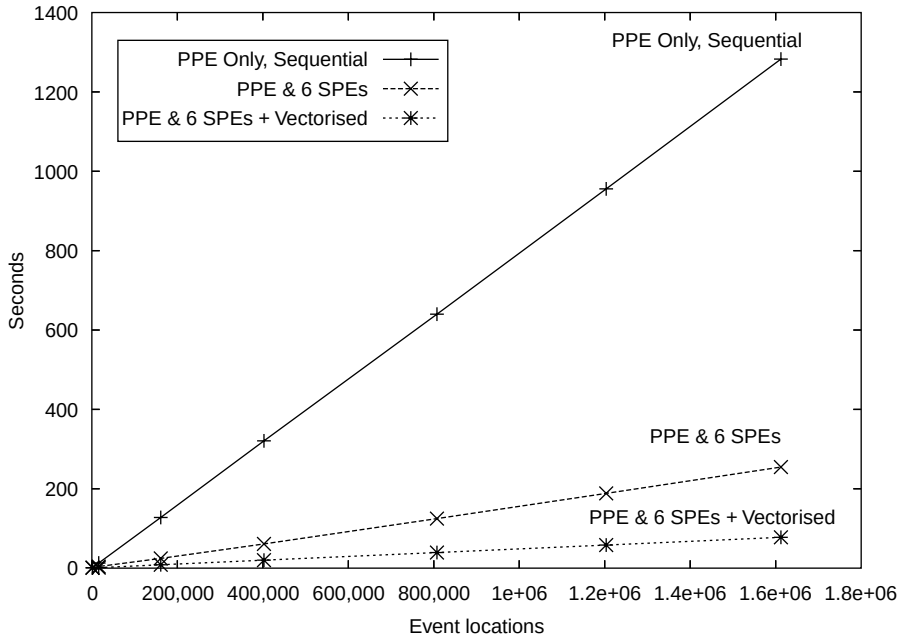

Figure 5. Wall clock time (seconds) for computing ELs from the Japanese earth quake dataset on a Playstation 3 as a function of the number of ELs.

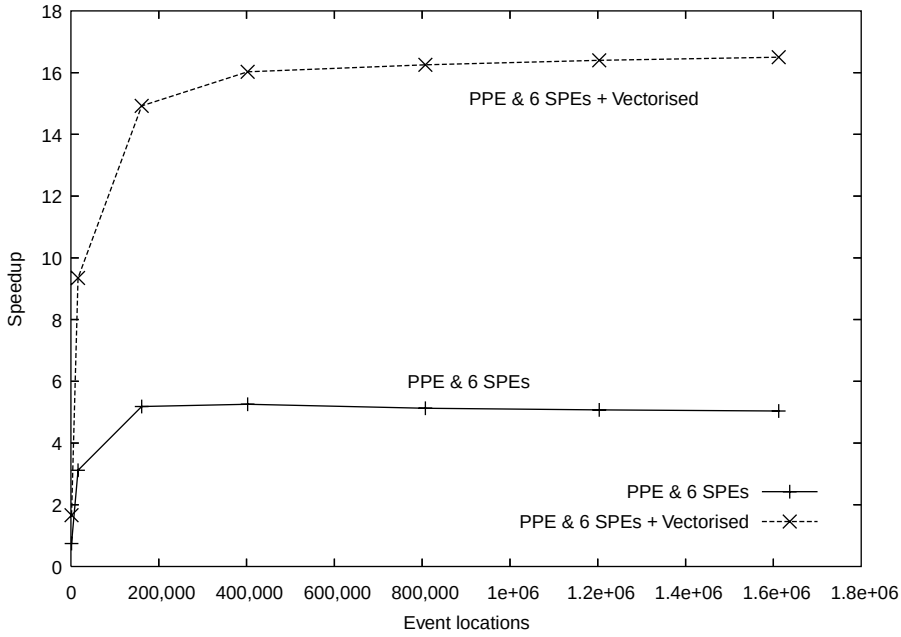

Figure 6. Speedup achieved for computing ELs from the Japanese earth quake dataset on a Playstation 3 as a function of the number of ELs.

Notes

1. Email: glenn.hickey@gmail.com
2. Email: arau-chaplin@flagstonere.bm

References

[1] M. Byrne, N.G. Kishi, and A. Rau-Chaplin, *Asian Catastrophe Insurance*, chapter, Inovating Technology for Catastrophe Risk Modelling and Reinsurance, Flagstone Reinsurance, Halifax, Canada, 2007.
[2] T. Chen, R. Raghavan, J.N. Dale, and E. Iwata, *Cell broadband engine architecture and its first implementation – A performance view*, IBM J. Res. Dev. 51(5) (2007), pp. 559–572.
[3] P. Grossi and H. Kunreuther (eds.), *Catastrophe Modeling. New Approach to Managing Risk*, Springer-Verlag, New York, 2005.
[4] C.R. Johns and D.A. Brokenshire, *Introduction to the cell broadband engine architecture*, IBM J. Res. Dev. 51(5) (2007), pp. 503–519.
[5] M.W. Riley, J.D. Warnock, and D.F. Wendel, *Cell broadband engine processor: Design and implementation*, IBM J. Res. Dev. 51(5) (2007), pp. 545–558.
[6] C. Scawthorn, *Risk Assessment, Modeling and Decision Support*, chapter, A Brief Histrory of Seismic Risk Assessment, Springer-Verlag, New Year, 2006.
[7] C. Scawthorn (ed.), *Special issue: Multihazards loss estimation and HAZUS*, Nat. Hazards Rev. 7(2) (2006), pp. 39–103.