Accelerating Earthquake Simulations on General-Purpose Graphics Processors

Prasenjit Sengupta[*], Jimmy Nguyen[†], Jason Kwan[‡], and Padmanabhan K. Menon[§]
Optimal Synthesis Inc., Los Altos, CA 94022

and

Eric M. Heien[**] and John B. Rundle[††]
University of California, Davis, CA 95616

Parallelization strategies are presented for Virtual Quake, a numerical simulation code for earthquakes based on topologically realistic systems of interacting earthquake faults. One of the demands placed upon the simulation is the accurate reproduction of the observed earthquake statistics over 3-4 decades. This requires the use of a high-resolution fault model in computations, which demands computational power that is well beyond the scope of off-the-shelf multi-core CPU computers. However, the recent advances in general-purpose graphic processing units have the potential to address this problem at moderate cost increments. A functional decomposition of Virtual Quake is performed and opportunities for parallelization are discussed in this work. Computationally intensive modules are identified and these are implemented on graphics processing units, significantly speeding up earthquake simulations. In the current best case scenario, a computer with six graphics processing units can simulate 500-years of fault activity in California at 1.5 km $\times$ 1.5 km element resolution in less than 1 hour, whereas a single CPU requires more than 2 days to perform the same simulation.

[*] Research Scientist, 95 First Street Suite 240, E-mail: sengupta@optisyn.com.
[†] Research Engineer, 95 First Street Suite 240, E-mail: jnguyen@optisyn.com.
[‡] Research Engineer, 95 First Street Suite 240. E-mail: jason@optisyn.com.
[§] Chairman and Chief Scientist, 95 First Street Suite 240, E-mail: menon@optisyn.com.
[**] Scientist, Computational Infrastructure for Geodynamics, Department of Earth and Physical Sciences, One Shields Ave, E-mail: emheien@ucdavis.edu.
[††] Professor, Department of Physics, E-mail: jbrundle@ucdavis.edu

I. Introduction

Numerical simulations play a crucial role in the study of the weather, global climate and complex interconnected solid-Earth processes. There are important similarities between the behavior of the atmosphere and the earth’s crust. Both systems are chaotic and highly unpredictable [1]. Because of this fact, computer simulations are invaluable in better understanding how atmospheric and solid-Earth systems operate. The increased resolution and complexity of the simulation models and their associated analysis systems are driving the requirements for NASA’s High-End Computing (HEC) resources program.

NASA has supported the development of the Virtual Quake (VQ) earthquake model. The specific implementation considered here is Virtual California (VC) [2], a topologically realistic numerical simulation (boundary element code) of earthquakes occurring on the fault systems in California. The core of the simulation code can be used for simulating earthquakes based on any fault system [3]. VC includes all the major strike-slip faults in California and has now been extended to depth-dependent boundary elements, dipping faults, as well as various other new features. VC allows the study of questions relating to the physics of earthquakes such as: 1) precursory failure process of major earthquakes on complex fault systems; 2) timing and statistics of major earthquakes on complex fault systems; and 3) origin of space-time correlations between major earthquakes. Another major application of VC simulations lies in earthquake forecasting. By systematically comparing simulation results to observed data, a series of spatial probability density functions can be assembled that describe the probable locations of future large earthquakes. These forecasts can yield fault-based locations for the next earthquake, as well as the most-probable locations for earthquakes over the next 30 years.

Among their more practical uses, earthquake forecasts are employed to set earthquake insurance rates, set the value of catastrophe bonds, aid in planning for disasters, and for seismic research. The current earthquake forecast for California was developed by the Working Group on California Earthquake Probabilities [4], [5], [6], [7]. The latest forecast was completed in April 2014 [7]. Typically these forecasts have been based largely on ‘expert opinion’ [5], [7] as well as a variety of statistical measures. The latest forecast called, for the first time, the use of numerical earthquake simulators as an independent means of estimating earthquake probabilities [7]. Plans call for future forecasts to be based largely on numerical simulators such as VC [7] or any one of the five numerical simulators currently undergoing development [8]. Of major importance will be the need for the capability of real-time earthquake forecasting, particularly of aftershocks, which can cause structures weakened by the main shock to collapse. Long run times preclude real-time forecasting with numerical earthquake simulators. However, computational acceleration methods such as those proposed in this paper can circumvent this and other problems.

The central idea in this approach is to generate empirical (numerical) statistics for earthquake faults, which take into account the physics of the fault system. Thus the reliance on generic statistical distributions is removed. This procedure is often used in numerical weather forecasting [9]. These empirical statistics can then be used to construct conditional probabilities for major earthquakes at various earthquake faults in the model [4], [5], [6]. Additional methods to construct conditional probabilities are described in detail by van Aalsberg *et al* [10].

One of the demands placed upon the simulations is the accurate reproduction of the Gutenberg-Richter and Omori statistics over 3–4 decades of linear scale. This requires grid sizes from the smallest at about 100 m, to the largest anticipated at about 400 km. The major active faults in California comprise roughly 10,000 km in aggregate length, so the number of fault elements required is about $n{\sim}10^{10}$. Current models use 3 km resolution, so an increase to 100 m resolution amounts to a 1000-fold increase in the number of elements. As a consequence, there is a million-fold increase in the number of Green's function evaluations. With the Barnes-Hut algorithm [11], run times scale as $O(n \log n)$, so the run times are expected improve by a factor of approximately 50. However, a 50,000-year VC simulation (order of 1 million events) running on a 32-core machine still consumes about 1536 core-hours. Therefore, a different computational approach is needed if simulations are to be performed closer to real time.

The computational complexity of earthquake simulation for high-resolution fault models can be addressed by new technologies being developed in the area of parallel computing, arising from the introduction of programmable Graphics Processing Units (GPUs). The GPU is a rapidly maturing technology offering a high level of parallel computing power for computationally demanding applications. Over the past few years, GPUs have evolved from a fixed-function processor built around computer graphics into a full-featured parallel processor which can be programmed through several high-level and low-level computing languages. Historically, GPUs were programmed by mapping mathematical operations to a graphics language such as OpenGL. NVIDIA's Compute Unified Device Architecture (CUDA), [12], [13], is a set of software tools for managing computations on the GPU as a data-parallel computing device while maintaining the structure of a high-level language. GPUs are now used by the research community in order to address a broad range of computationally demanding and complex problems [14]. One of the attractive features of the GPU-enabled workstation is that it provides performance that rivals supercomputing clusters at desktop price-points.

GPU technology has been used in related prior work. For example, Komatitsch *et al* [15] implemented a finite-element-based earthquake simulation on a single GPU using CUDA. A finite-differenced wave equation, known as the Anelastic Wave-Propagation-Olsen-Day-Cui model, was accelerated using GPUs by Unat *et al* [16]. Without discussing specific theoretical differences between prior work and this paper, it is worth noting that the work by Unat *et al* used Mint [17], an automated tool for C-to-CUDA conversion. It was also limited to implementation on several CPUs with a single GPU device each. Work by In comparison with [15] and [16], the present work uses customized CUDA code and is generalized to clusters of machines with multiple GPUs.

This paper is organized as follows: Section 2 provides a functional overview of VC code and describes its computational components. Section 3 discusses the methods used to implement VC on GPUs. The computational experience and acceleration results are presented in Section 4. Conclusions and avenues for future work are discussed in Section 5.

II. Functional Decomposition of Virtual California

As described earlier, Virtual California is a model that includes stress accumulation and release, as well as stress interactions on the San Andreas and other adjacent faults (Figure 1 [32]) [18], [19], [20], [21]. The model is based on a set of mapped faults with estimated slip rates, a prescribed plate tectonic motion, potential earthquakes on all faults, and elastic interactions [18], [22], [23], [24]. Earthquake activity data and slip rates on these model faults are obtained from geologic databases. In contrast to the recent past, VC simulations now include dipping strike slip, thrust, and normal faults. Similar types of simulations have been developed by others [25], [26], [27], [28], [30].

Loading of each VC fault segment occurs due to the accumulation of a slip deficit at the prescribed slip rate of the segment (“backslip model”). The vertical rectangular fault segments interact elastically. Earthquake initiation is controlled by friction coefficients along with the space- and time-dependent stresses on fault segments which are computed by means of boundary element methods. Historical earthquakes are used that have moment magnitudes $M \geq 6.0$ in California during the last 200 years to prescribe the friction coefficients. A consequence of the fault segmentation is that the simulations do not generate earthquakes with magnitudes less than about $M \approx 5.8$.

Using VC, it is possible to construct simulated interferograms associated with major simulated earthquakes (Figure 2). These can be quantitatively compared to actual interferograms [31].

A. Computational Overview of the Simulation

VC is an efficient, parallel object-oriented C++ numerical code that runs on NASA HEC's Columbia system and JPL's COSMOS computer using Message Passing Interface (MPI) protocols. Figure 3 shows the path of execution in a parallel simulation running on multiple processors either on a cluster, multi-core machine or GPUs. Note that the figure shows the splitting of the execution across multiple parallel processors and the global reduction operations.

The execution is divided into three distinct phases:

1. Initialization: The initialization phase begins by parsing the specified model and the simulation parameters. If the simulation is running on multiple processors, the initialization process partitions the fault elements. This partitioning tries to ensure that each processor is responsible for roughly an equal number of elements and that elements on the same processor are on the same fault or geographically close to each other. Next, each processor calculates stress influences by all model elements upon the local elements. The core of the simulation involves cycling between two main phase: i) determining long-term stress buildup in the system, and ii) propagation of a rupture through the system. These are shown by the lower two-thirds of Figure 3.
2. Long-Term Stress Interaction: The simulation begins in the long-term stage by calculating the rate of long-term stress buildup for each element. In a parallel simulation each processor determines when each of the local elements will rupture. This is then globally reduced to finding when and where the first rupture will occur. Once this is determined, the rupture is propagated through the system.
3. Rupture Propagation: First, the ruptured elements are processed and their new stresses are communicated through the system. Each processor recalculates the effects of this change

on the stresses of their local elements and determines which, if any, have ruptured. If any of the processors experience further ruptures, the rupture propagation phase continues. This phase generally involves multiple propagation steps until all the earthquake computations are complete. Once there are no more ruptures, the simulation returns to the long-term stress calculation. The simulation is terminated after the specified number of simulation years have elapsed.

B. VC Code Profiling

A parallel version of VC has been implemented on a cluster using MPI and C++; this work is described by Heien *et al* [32]. The performance of this MPI implementation was evaluated using the northern California fault model shown in Figure 1 with $1\,\text{km} \times 1\,\text{km}$ segments in 6 layers. The simulation was run on a 16-node cluster, with each node having 24 GB of RAM and two quad core Intel Xeon processors running at 2.4 GHz and with all nodes connected by a Gigabit Ethernet switch. The VC simulation was performed for a simulation time of 100,000 years. For a single processor, most of the time is spent in stress calculation (69.1%) and Green's function calculation (27.5%). Very little time is spent in rupture propagation (3.2%) or communication functions. As the number of processors increases, communication dominates the simulation time. By 16 processors the simulation spends the most time in communication functions (61.9%). Because of this the simulation fails to scale favorably with increasing number of processors.

This study also noted that as the number of elements increases, the following behaviors are note: i) the proportion of time spent in performing the matrix vector multiplication increases, and ii) the proportion of time spent in creating the Green's function matrix and the rest of the functions in the application decreases.

III. Acceleration of VC on GPUs

Based on profiling the VC code on CPUs, it was found that the two segments of the simulation that perform the best when accelerated are the Green's function matrix construction, followed by the stress propagation using matrix-vector multiplication. Prior to the Green's function matrix calculation, fault elements, obtained from the user-specified fault models, need to be partitioned across different nodes using a partitioning scheme. Let n be the total number of fault elements and m be the number of nodes in a cluster. Each node is assigned a subset of fault data, which is given by n/m, where m is the number of nodes. Using MPI, fault assignment to each individual node is also broadcast to all other nodes so that each node can keep track all assignments. Maps are also maintained on each node to help keep track of other nodes' fault assignment.

Acceleration of Green's function matrix calculation and stress vector propagation are discussed in the following subsections.

A. Acceleration of Green's Function Matrix Calculations

A two-level nested loop which iterates of rows and columns is used to generate the entries of the Green's function matrix in the VC model. These matrix elements are calculated using closed-form analytical expressions given by Okada [33]. There are 232 Okada functions, which were re-implemented using CUDA for execution on the GPU.

The features that makes Green's function matrix generation suitable for GPU implementation are:

1. Independence: Calculation of each entry of the Green's function matrix is independent of the others and hence they can be generated concurrently in parallel.
2. Degree of Parallelism: The total number of parallel threads for the problem is equal to number of entries in the matrix (n^2), where n is the number of fault elements. For instance, an all-California problem at 3 km resolution consists of 11,291 elements, providing potentially 127 million parallel threads. It may be observed that the degree of parallelism can be very high.
3. Arithmetic Intensity: Arithmetic intensity is defined as the number of computations performed per memory transaction. For every entry of the matrix, $14{,}000$ to $43{,}000$ mathematical operations are performed to obtain the output values starting with as few as 17 input double-precision floating point numbers.

On each multi-GPU supporting node, Green's function matrix (also called the stress influence matrix) is generated from the fault data on multiple GPU devices. This step consists of calculating two matrices, one each for shear and normal stress coefficients. The function that generates the Green's function matrix first determines the correct matrix sizes to allocate for each GPU device, taking into account odd data sizes. Hence each GPU on a node should have approximately two matrices of size $(n/m/g_m) \times n$, one for shear and one for normal stress. In the foregoing formula, g_m denotes the number of GPU devices on the mth node.

The stress influence matrices on each GPU only contain a portion of the fault data. The function then concurrently performs Green's function evaluations on all of these GPU devices. Parallelization is achieved using the OpenMP API, which provides support for shared-memory parallel thread management and programming within a machine. Thus the function creates as many computing parallel threads as the number of GPU devices, where each thread handles its own Green's function calculation on a single GPU, using its own portion of the assigned fault data.

Green's function matrices are stored in column-major format and allocated with pitch memory to ensure global memory coalescing. Pitch memory refers to a CUDA practice when memory is allocated with padding to ensure coalesced memory access. This is because GPU memory access is most efficient when each kernel accesses contiguous memory blocks of the same size. This efficient memory management also speed up the matrix multiplication which is discussed in the next section.

The computation of Green's function matrices is also parallelized across multiple rows. Kernels are executed concurrently using the concept of streams and asynchronous memory transfers between host and the device. NVIDIA Fermi and later versions of the GPU support the launch of multiple kernels on the same GPU and concurrent execution of kernels on different streams. Consequently, the kernel can be invoked on different streams to achieve parallelization, with the kernel being responsible for computing one row of the Green's function matrices. Each row of the matrix corresponds to one fault element that is assigned to the GPU.

The calculation of each element in each row of the Green's matrices is also independent of all the others and hence they can be generated concurrently in parallel.

B. Acceleration of Stress Propagation Calculations

After the Green's function evaluations are complete on multiple GPU devices, the stress influence matrices (normal and shear) are stored on each GPU. Stress propagation calculations in VC are performed by using matrix-vector multiplication to determine the vector of stresses $\sigma_{ij}^{A}(t)$ on the various fault elements, given the vector of strains $s_B(t)$ on each element and the stress influence matrix T_{ij}^{AB}:

$$\sigma_{ij}^{A}(t) = T_{ij}^{AB} s_B(t) \tag{1}$$

Matrix-vector multiplication in VC is parallelized by dividing the matrix columns into blocks, multiplying them with portions of the vector and accumulating the resulting sums into an output vector. These steps constitute a 2-D partitioning process of the matrix [34]. Note that in a matrix vector multiplication, each row of the matrix multiplies the entire input vector to generate one element of the output vector. This is implemented in parallel over the rows of the matrix, where thread i multiplies row i of the matrix T_{ij}^{AB} with input vector $s_B(t)$ to generate the ith entry in the $\sigma_{ij}^{A}(t)$ vector. The input vector $s_B(t)$ is stored in shared memory to enhance runtime performance.

During the rupture propagation phase, the strain vector is typically sparse. The matrix multiplication then operates only on the non-zero elements of $s_B(t)$ to lower memory and computing requirements. The check for non-zero values in $s_B(t)$ introduces a conditional statement which can potentially cause warp divergence. In other words, each parallel kernel can require different amounts of time, which can reduce efficiency. However, divergence between parallel operations occurs in at most one addition and one multiplication. Moreover the check is performed on the vector only, since the block matrices are typically dense. Thus the benefits of performing limited number of computations on a sparse vector greatly outweigh any loss in efficiency due to the presence of conditional statements.

At every stress evaluation stage, the strain vector $s_B(t)$ is copied from the host memory to the GPU global memory. Then the strain vector is divided into smaller segments B1, B2, B3, etc., depending on the maximum size that will fit in shared memory (Figure 4). The first segment B1 is next copied into the shared memory. The kernel multiplies the corresponding block A1 from the T_{ij}^{AB}matrix with the shared memory segment B1 and the result is accumulated at the output. The second segment B2 is copied next, and multiplied with A2 to yield the next element of the output. This process is continued until all the elements of the output stress vector $\sigma_{ij}^{A}(t)$ have been computed.

In VC, both the normal and shear stresses are calculated from the strain vector using the process described above. The normal and shear stress calculations are independent of each other and can be performed concurrently. As noted before, GPUs of the Fermi type and later support simultaneous launch of multiple kernels on the same GPU. While kernels in the same stream execute in order, one after the other, kernels on different streams can execute concurrently.

Given enough hardware resources, this process achieves task-parallelism in addition to the data-parallelism.

In the current GPU implementation, the kernels for calculating the normal and shear stresses are instantiated in separate streams on the GPU. The shear and normal stress calculations are then performed in parallel, if enough hardware resources are available. The foregoing process can be readily divided among multiple GPUs. Each GPU is apportioned a predefined number of blocks and segments and matrix-vector multiplications are performed independently on each GPU. The last step is that of merging the resulting vectors obtained from each GPU device. The function uses stream synchronization as a mechanism to wait for all GPU devices to complete their own matrix multiplication tasks. Once these are complete, the results may be merged in a straightforward manner.

IV. Runtime Performance Evaluation

A number of problems of varying resolutions were tested to assess the acceleration offered by implementation of VC on GPUs. These are listed in Table 1.

The problems listed in Table 1 are arranged in order of the increase in the number of elements used to represent the faults in the state of California. For example, the ‘Parkfield’ model refers to a single fault in the Central California in the Carrizo Plain, modeled using 48 elements, while ‘SAF’ refers to all faults of the main strand of the San Andreas Fault. The models starting with ‘AllCal’ consist of all faults in the state of California, but at different resolutions. For instance, ‘AllCal_3.0km’ consists of a model where fault elements are of the size 3 km × 3 km.

The third column in Table 1 lists the memory required to store the Green’s function matrices for each problem. Device memory on the GPU is the constraint which determines how many GPUs are required to perform computations on a model of a given size. For example, an all-California problem at 3 km × 3 km can be simulated on a single NVIDIA Kepler GPU but a finer resolution model such as the 2.1 km × 2.1 km problem requires at least two GPUs.

The CUDA implementations of VC on single-GPU and multiple-GPU configurations were evaluated by comparing the results against the C++ implementation. The validation of results consisted of three steps. First, a direct comparison of the entries of the Green’s function matrices was performed. For small matrices (e.g. the Parkfield problem), the comparison was performed by inspection. For larger matrices, comparison was performed by writing the matrix contents to files, loading them in an external scripting environment (Python), and calculating the Frobenius norm of the difference matrix.

Second, stress values at the end of an iteration were compared. For small matrices, the elements of the stress vector obtained from the GPU and CPU implementations were manually compared. For larger matrices, the contents of the vectors were written to files, which were then loaded in an external scripting environment. The vector norm of the difference was then calculated. This test was performed at the end of every iteration to ensure that rounding errors did not accumulate.

Finally, VC simulation stores the time of event occurrence and the magnitude of an event at all the faults in a simulation. The values from the VC simulation of the CPU code and GPU code were compared in order to ensure accuracy.

Both implementations utilized double-precision floating point operations. Note that this requires that all GPUs on a node offer at least Compute Capability 2.0. It is possible to use VC with single-precision arithmetic, which can potentially enable the simulation of larger problems on fewer GPUs. However, this option can have implications on the accuracy of long-term simulations, especially when the statistical nature of the model is considered.

A. Computational Platforms

The foregoing problems were tested on four different configurations detailed in the following:

1. Configuration 1: this configuration consists of two Intel Xeon E5620 CPUs with four cores each, operating at 2.4 GHz. The system has 72 GB of RAM available. A GPU is also available on this system, which was disabled when this configuration was used as the CPU benchmark run against which GPU implementation speedups were evaluated.
2. Configuration 2: This configuration consists of one desktop computer equipped with two NVIDIA GPUs: a Tesla K20x (server-class GPU consisting of a Kepler GK110) and a Titan, each with 2688 cores, operating at 732 MHz, with 6 GB of device memory each. This system is suitable for problems of resolution up to 2.1 km × 2.1 km (AllCal_2.1km).
3. Configuration 3: this is a node from the NVIDIA cluster, equipped with six Tesla K40 GPUs (server-class GPU consisting of one Kepler GK110B) each with 2880 cores and 12

GB memory. This configuration can solve problems of the size associated with AllCal_1.5km.

4. Configuration 4: consists of two nodes from the NVIDIA cluster; each node has four Tesla K40 GPUs. This configuration was used to test the cluster-with-multiple-GPU version of VC.

B. Acceleration Results

Execution times for a 500-year simulation were calculated using the configurations discussed in Section 4.1. The execution times from Configurations 2-4 were compared with those from Configuration 1 in order to assess the speedup obtained using the single- and multi-GPU implementations of VC. These are reported in Table 2. Shaded cells represent configurations that were not tested. In a small subset of cases, either no speedup was observed, or using a GPU produced slower performance. This is because the benefit of a large number of GPU cores is outweighed by its slower clock speed and the overhead associated with memory transfers between the CPU host and GPU devices. An example of the first case is given by Problem #1 with 48 fault elements, and an example of the latter case is seen when moving from 1 GPU to 2 GPUs in Problem #5 with 11,291 fault elements. However, in general, the speedup using a GPU implementation over a CPU implementation was observed to have an accelerating trend with the size of the problem. Results indicate that higher speedup can be expected for larger problems when implemented on clusters of CPU nodes with multiple GPUs each. Performance in some of the other larger problems could not be carried out because the CPU-GPU clusters were available only on a limited basis.

The acceleration trend for a single node with 1, 2, and 6 GPUs is shown graphically in Figure 5. In this figure, the solid black vertical line indicates the number of elements in an all-California simulation with a fault element resolution of $500\text{ m} \times 500\text{ m}$. This problem will be composed of 418,660 fault elements and is expected to require 2.6 TB of memory to store the stress influence matrices. Ongoing research is aimed at reducing the amount of memory required for storing matrices in very large problems.

The best performance in the present research was obtained for the $1.5\text{ km} \times 1.5\text{ km}$ resolution problem which consists of 46,033 fault elements. This problem, when executed on a CPU with 16 concurrent threads, requires more than 2 days for a 500-year simulation. The same problem on a system with 6 GPUs requires less than 1 hour. Utilization of a cluster of computers with multiple GPUs is therefore a key technology crucial for simulating higher-resolution models at 1 km $\times$ 1 km, 500 m $\times$ 500 m, or finer.

The VC speedup on GPUs reported in this paper is strongly dependent on the acceleration of Green's function matrix construction and the stress propagation. For the simulation examples considered in this paper, the stress propagation dominated the computation time since the matrices are only constructed once at the beginning of the simulation. For long-term simulations with parameter updates, the Green's function matrices may require periodic recalculation. This can potentially improve speedup because of the truly independent nature of element-wise calculation in the stress influence matrix.

V. Conclusion

This paper outlined the process of implementing the Virtual California earthquake simulation software on a system with one or more CPUs and GPUs to leverage the large number of parallel threads that can be executed at an instant on these architectures. The algorithms constituting the earthquake simulations are parallelizable but the native C++ implementation require several hours of computation time on clusters consisting of CPUs for modest problem sizes. The native code was analyzed and profiled to identify functions which required the greatest amount of computational resources based on the proportion of computing times consumed, relative to the overall execution time. These functions were then accelerated in order to achieve the biggest payoff in performance.

The CUDA-enabled software can be executed on a cluster of CPUs consisting of nodes equipped with one or more GPUs. The software utilizes MPI to communicate and share memory and tasks between nodes, and OpenMP and CUDA to utilize parallel processing on GPUs on each node.

The best performance achieved in the present research was in simulating the 1.5 km $\times$ 1.5 km resolution model of the faults in California. This higher resolution model yielded a speedup of 56$\times$, demonstrating that a 2-day simulation on a CPU-only machine with 16 concurrent threads can be performed in less than 1 hour on a computer equipped with 6 GPUs.

It is evident from the observed results that using a CPU cluster equipped with multiple GPUs has the potential for further speeding up computations associated with Virtual California earthquake simulation code. Virtual California is an implementation of Virtual Quake which uses

California fault models specifically. Therefore in general, simulation of fault activity for larger geographical regions will also benefit from cluster-wide implementations. Furthermore, cluster implementation is essential to the ability of solving problems at resolution 1 km × 1 km or finer. Further research is required in the GPU implementation of algorithms such as Barnes-Hut for exceptionally high resolution problems such as 100 m × 100 m, in order to exploit compressed forms of the stress influence matrices for stress propagation calculations.

Acknowledgment

This work was supported by NASA Headquarters under Grant NNX12AP98G with Dr. Tsengdar J. Lee as Technical Officer. This work has greatly benefited from frequent discussions on the CPU/GPU implementations of the VC code with him. The authors would also like to thank Mr. Greg Branch of NVIDIA Corp. for providing access to the NVIDIA GPU cluster machine, which enabled several additional software tests.

References

[1] Rundle, J. B., Turcotte, D. L., and Klein, W., *Geocomplexity and the Physics of Earthquakes*, Geophysical Monograph Series, Vol. 20, American Geophysical Union, 2000.

[2] Rundle, P. B., Rundle, J. B., Tiampo, K. F., Donnellan, A., and Turcotte, D. L., "Virtual California: Fault Model, Frictional Parameters, Applications," *Pure and Applied Geophysics*, Vol. 163, No. 9, Sep. 2006, pp. 1819—1846.

[3] Heien, E. M., and Sachs, M., "Understanding Long-Term Earthquake Behavior through Simulation," *Computing in Science and Engineering*, No. 5, Vol. 14, Sep.—Oct. 2012, pp. 10—20.

[4] Working Group on California Earthquake Probabilities www.wgcep.org, retrieved July 13, 2015.

[5] Field, E. H., Dawson, T. E., Felzer, K. R., et al., "Uniform California Earthquake Rupture Forecast, Version 2 (UCERF 2)," *Bulletin of the Seismological Society of America*, vol. 99, no. 4, pp. 2053-2107, Aug. 2009.

[6] Field, E.H., "A Summary of Previous Working Groups on California Earthquake Probabilities," *Bulletin of the Seismological Society of America*, vol. 97, no. 4, pp. 1033-1053, Aug. 2007.

[7] Field, E.H., Arrowsmith, R.J., Biasi, G.P. et al., "Uniform California Earthquake Rupture Forecast, Version 3 (UCERF3)—The Time-Independent Model", *Bulletin of the Seismological Society of America*, vol. 104 no. 3, pp. 1122-1180, June. 2014.

[8] Tullis, T. E., "2010 Earthquake Simulators Workshop Report," available at http://www.scec.org/workshops/2010/simulators /index.html, Jul. 2010.

[9] Leutbecher, M., Palmer, T. N., "Ensemble Forecasting," *J. Computational Physics*, vol. 227, no. 7, pp. 3515-3539, Mar. 2008.

[10] Van Aalsburg, J., Rundle, J. B., Grant, L. B., Rundle, P. B., Yakovlev, G., Turcotte, D. L., Donnellan, A., Tiampo, K. F., and Fernandez, J., "Space- and Time-Dependent Probabilities for Earthquake Fault Systems from Numerical Simulations: Feasibility Study and First Results," *Pure and Applied Geophysics PAGEOPH*, vol. 167, pp 967-977, 2010.

[11] Barnes J., and Hut, P., "A hierarchical $O(N\,log\,N)$ force-calculation algorithm," *Nature*, vol. 324, no. 6096, pp. 446–449, Dec. 1986.

[12] Kirk D. B., and Hwu W. W., "Programming Massively Parallel Processors: A Hands-on Approach," Burlington MA: *Morgan Kaufmann*, 2010.

[13] Sanders J., and Kandrot E., "CUDA by Example: An Introduction to General Purpose GPU Programming," Reading, MA: *Addison-Wesley Professional*, 2010.

[14] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., and Phillips, J. C., "GPU Computing," *Proceedings of the IEEE*, Vol. 96, No. 5, May 2008, pp. 879—899.

[15] Komatitsch, D., Michéa, D., and Erlebacher, G., "Porting a High-Order Finite-Element Earthquake Modeling Application to NVIDIA Graphics Cards using CUDA," *Journal of Parallel and Distributed Computing*, Vol. 69, No. 5, May 2009, pp. 451—460.

[16] Unat, D., Zhou, J., Cui, Y., Baden, S. B., and Cai, X., "Accelerating a 3-D Finite Difference Earthquake Simulation with a C-to-CUDA Translator," *Computing in Science & Engineering*, vol. 14, no. 3, pp. 48-59, May 2012.

[17] Unat, D., Cai, X., and Baden, S. B., "Mint: Realizing CUDA performance in 3D Stencil Methods," *Proc. 25th International Conference on Supercomputing (ICS '11)*, pp. 214-224, May 2011.

[18] Rundle, J. B., “A Physical Model for Earthquakes: 2. Application to Southern California,” *J. Geophysical Research*, vol. 93, no. B6, pp. 6255-6274, Jun. 1988.

[19] Rundle, J. B., Rundle P. B., Donnellan A., Turcotte, D. L., Shcherbakov. R., Li, P., Malamud, B. D., Grant, L. B., Fox, G. C., McLeod, D., Yakovlev, G., Parker, J., Klein, W., and Tiampo, K. F., “A Simulation-Based Approach to Forecasting the Next Great San Francisco Earthquake,” *Proc. National Academy of Sciences, USA.*, vol. 102, no. 43., pp. 15363-15367, Aug. 2005.

[20] Van Aalsburg, J., Grant, L. B., Yakovlev, G., Rundle, P. B., Rundle, J. B., Turcotte D. L., and Donnellan, A., “A Feasibility Study of Data Assimilation in Numerical Simulations of Earthquake Fault Systems,” *Physics of Earth and Planetary Interiors*, vol. 163, pp. 149-162, Aug. 2007.

[21] Yikilmaz, M. B., Turcotte, D. L., Yakovlev, G., Rundle, J. B., and Kellogg, L. H., “Virtual California Earthquake Simulations: Simple models and their Application to an Observed Sequence of Earthquakes,” *Geophysical Journal International*, vol. 180, no. 2, pp. 734-742, Feb. 2010.

[22] Rundle, P. B., Rundle, J. B., Tiampo, K. F., Martins, J. S., McGinnis, S., and Klein, W. “Nonlinear Network Dynamics on Earthquake Fault Systems,” *Physical Review Letters*, vol. 87, no. 14, pp. 148501-1—148501-4, Oct. 2001.

[23] Rundle, J. B., Tiampo, K. F., Klein, W., and Sa Martins, J. S., “Self-Organization in Leaky Threshold Systems: The Influence of Near-Mean Field Dynamics and its Implications for Earthquakes, Neurobiology, and Forecasting,” *Proc. National Academy of Sciences, USA.* vol. 99, pp. 2514-2521, Feb. 2002.

[24] Rundle, J. B., Rundle, P. B., Donnellan, A., and Fox, G. C., “Gutenberg-Richter Statistics in Topologically Realistic System-Level Earthquake Stress-Evolution Simulations,” *Earth Planets and Space*, vol. 56, no. 8, pp. 761-771, Aug. 2004.

[25] Ward, S. N., “An Application of Synthetic Seismicity in Earthquake Statistics: The Middle America Trench,” *J. Geophysical Research*, vol. 97, no. B5, pp. 6675-6682, May 1992.

[26] Ward, S. N., “A Synthetic Seismicity Model for Southern California: Cycles, Probabilities, and Hazard,” *J. Geophysical Research*, vol. 101, no. B10, pp. 22393-22418, Oct. 1996.

[27] Ward, S. N., “San Francisco Bay Area Earthquake Simulations: A Step toward a Standard Physical Earthquake Model,” *Bulletin of Seismological Society of America,* vol. 90, no. 2, pp. 370-386, Apr. 2000.

[28] Goes, S. D. B. and Ward, S. N., “Synthetic Seismicity for the San Andreas Fault,” *Annals of Geophysics*, vol. 37, no. 6, pp. 1495-1513, Dec. 1994.

[29] Robinson, R., “A Test of the Precursory Accelerating Moment Release Model on some recent New Zealand Earthquakes,” *Geophysical Journal International*, vol. 140, no. 3, pp. 568-576, Mar. 2000.

[30] Dieterich, J. H., Richards-Dinger, K. B., “Earthquake Recurrence in Simulated Fault Systems,” *Pure and Applied Geophysics PAGEOPH,* vol. 167, pp. 1087-1104, 2010.

[31] Wei, M., Sandwell, D., and Smith-Konter, B., “Optimal Combination of InSAR and GPS for Measuring Interseismic Crustal Deformation,” *Advances in Space Research*, vol. 46, no. 2, pp. 236-249, Jul. 2010.

[32] Heien, E. M., Yikilmaz, M. B., Sachs, M. K., Rundle, J. B., Louise H. Kellogg, L. H., Turcotte, D. L., “Parallelization of the Virtual California Earthquake Simulator,” *International Conference on Computational Science (ICCS)*, 2011.

[33] Okada, Y., “Internal deformation due to shear and tensile faults in a half-space,” *Bulletin of the Seismological Society of America*, Vol. 82, no. 2, pp. 1018-1040, Apr. 1992.

[34] Grama, A., Gupta, A., Karypis, G., and Kumar, V., *Introduction to Parallel Computing*, Essex, UK: Pearson Education Ltd., pp. 337-345, 2003.

List of Tables

Table 1. List of VC Problems

Model Name	**No. of Fault Elements**	**Size of Green's Function Matrix**
Parkfield	48	24 KB
SAF	1,508	17.48 MB
AllCal2_Trunc4905	4,905	183.82 MB
AllCal2_Trunc7453	7,453	423.96 MB
AllCal_3.0km	11,291	2 GB
AllCal_2.1km	25,700	7 GB
AllCal_1.7km	29,496	19 GB
AllCal_1.5km	46,033	33 GB

Table 2. Simulation Time and Speedup on CPU, single GPU, multiple GPUs, and Cluster

Elements	**1 node CPU**	**1 node × 1 GPU**	**1 node × 2 GPUs**	**1 node × 6 GPUs**	**2 nodes × 4 GPUs**
48	< 1s	1s (<1×)			
1,508	15s	5s (3×)			
4,905	2m 46s	15s (11×)			
7,453	6m 11s	28s (13×)			
11,291	1h 26m	2m 31s (32×)	2m 38s (32×)		2m 17s (37×)
25,700	3h 11m	**Out of Memory**	5m 32s (35×)	4m 39s (41×)	
29,496	12h 17m	**Out of Memory**	**Out of Memory**	14m 58s (49×)	
46,033	48h 33m	**Out of Memory**	**Out of Memory**	52m 15s (56×)	

List of Figures

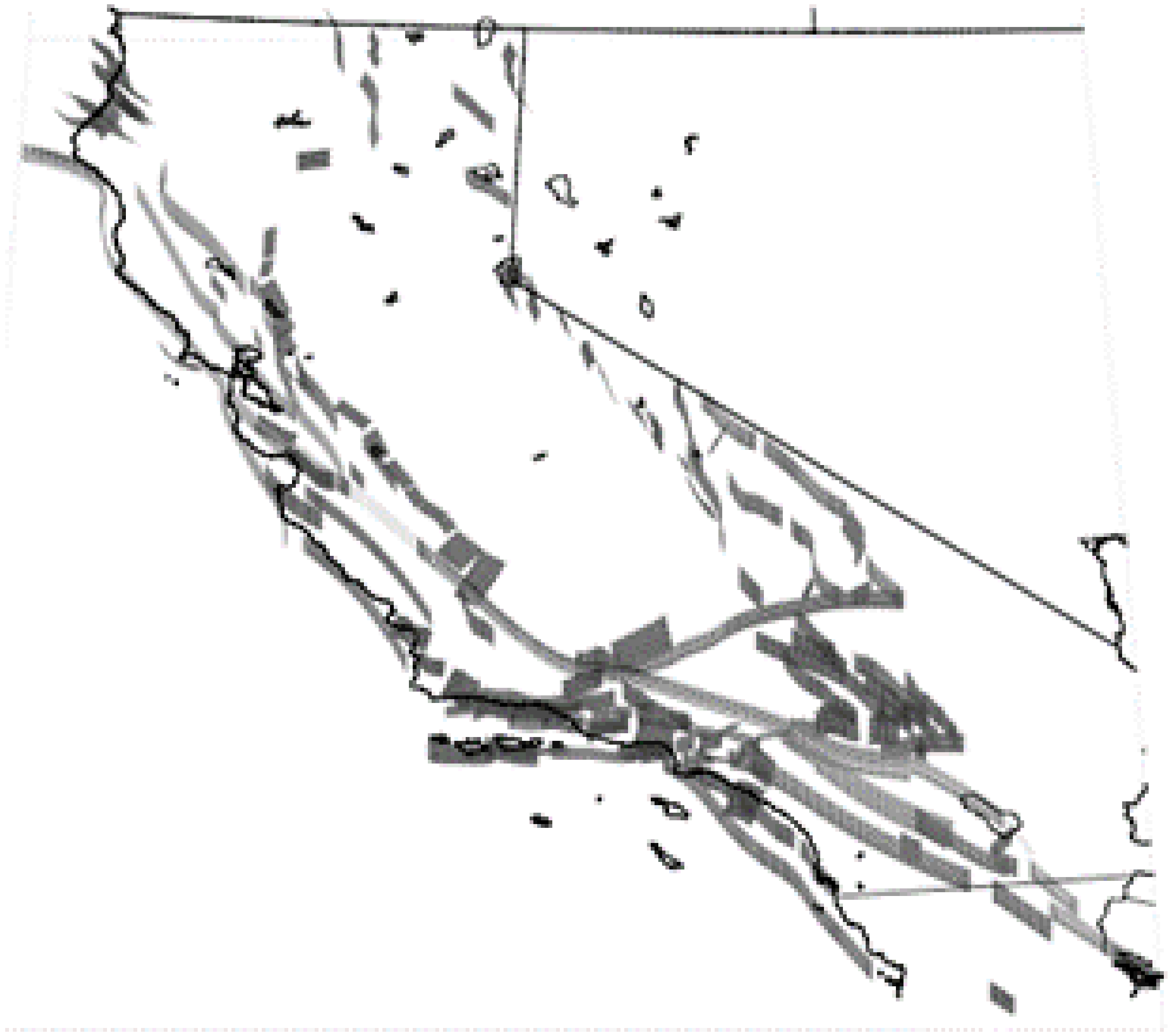

Figure 1. Fault Model used in Virtual California Simulation

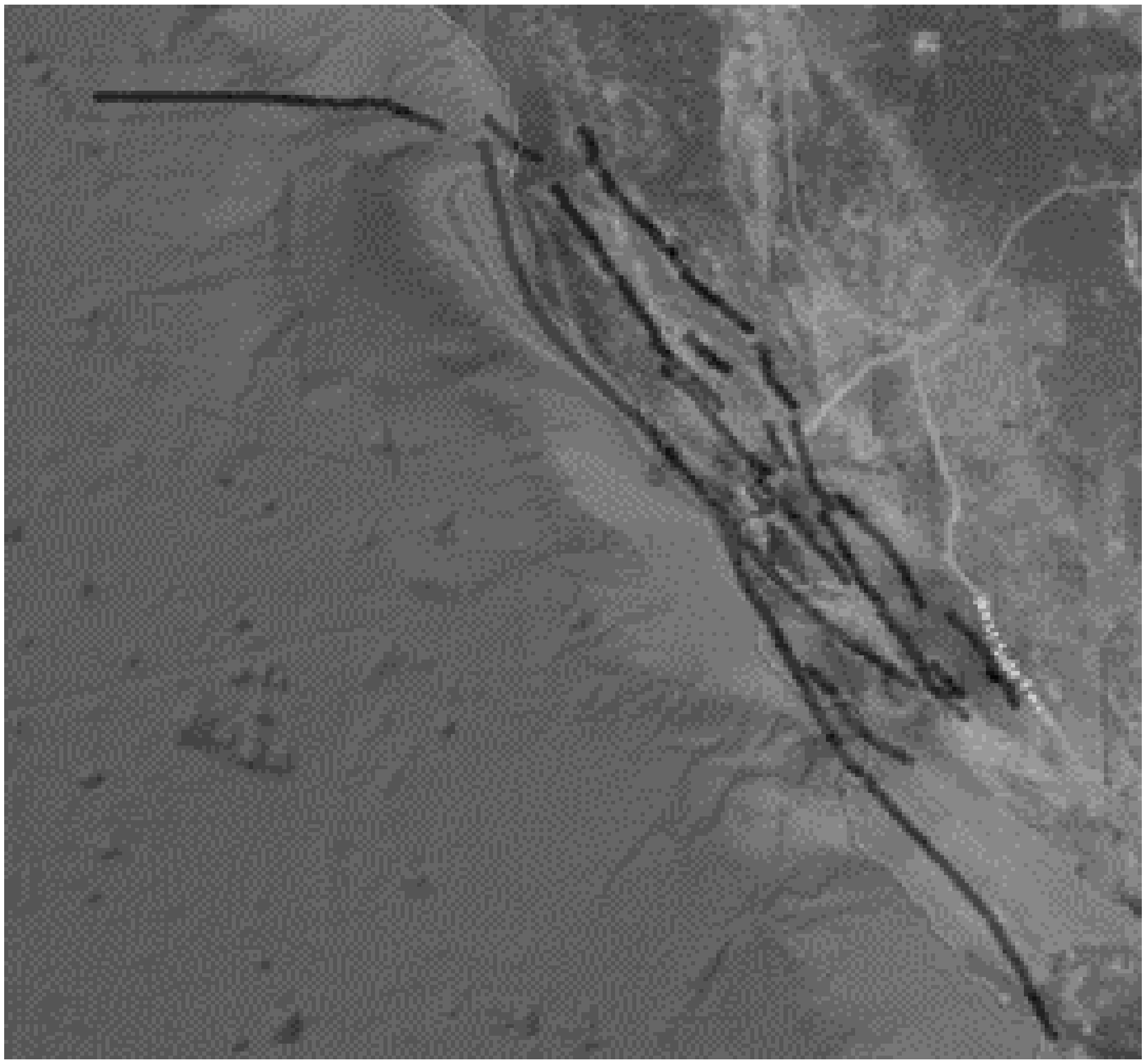

Figure 2. A Northern California VC Fault Model with Overlaid Interferogram from a Simulated 1906-Type Earthquake

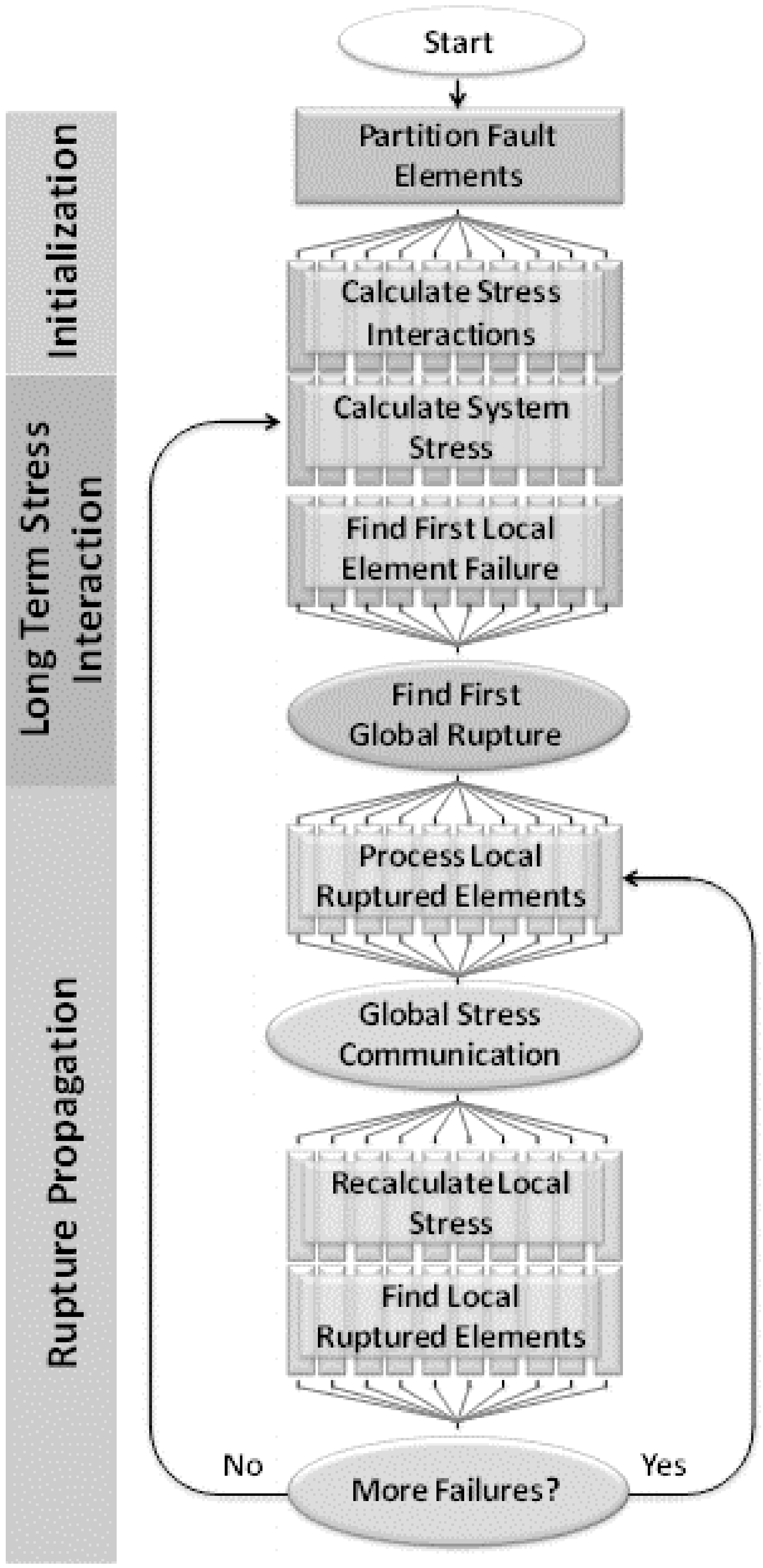
Start
Partition Fault Elements
Calculate Stress Interactions
Calculate System Stress
Find First Local Element Failure
Find First Global Rupture
Process Local Ruptured Elements
Global Stress Communication
Recalculate Local Stress
Find Local Ruptured Elements
No
More Failures?
Yes
Initialization
Long Term Stress Interaction
Rupture Propagation

Figure 3. Execution of Virtual California on a Parallel System

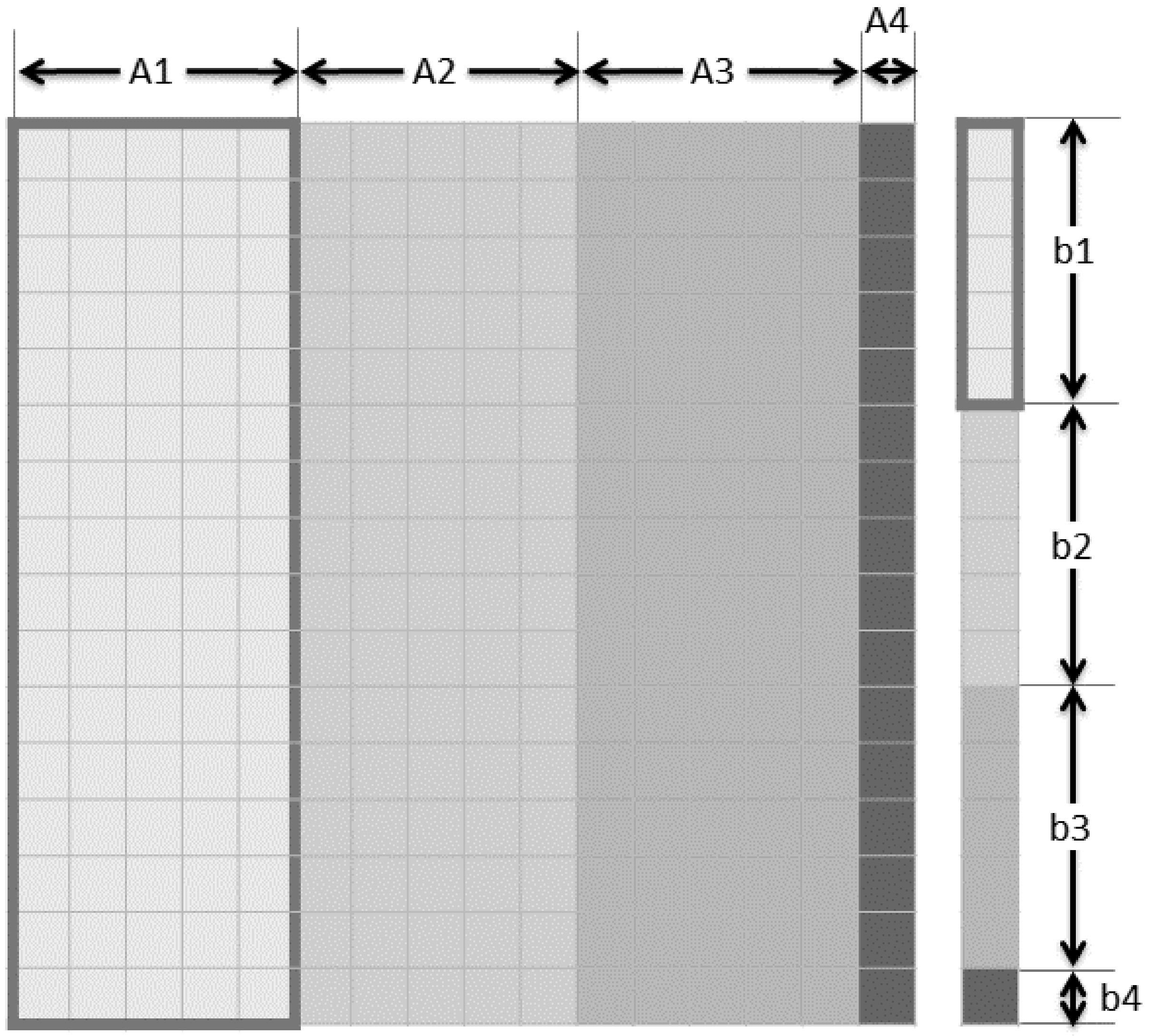

Figure 4. Parallel Matrix-Vector Block Multiplication and Accumulation

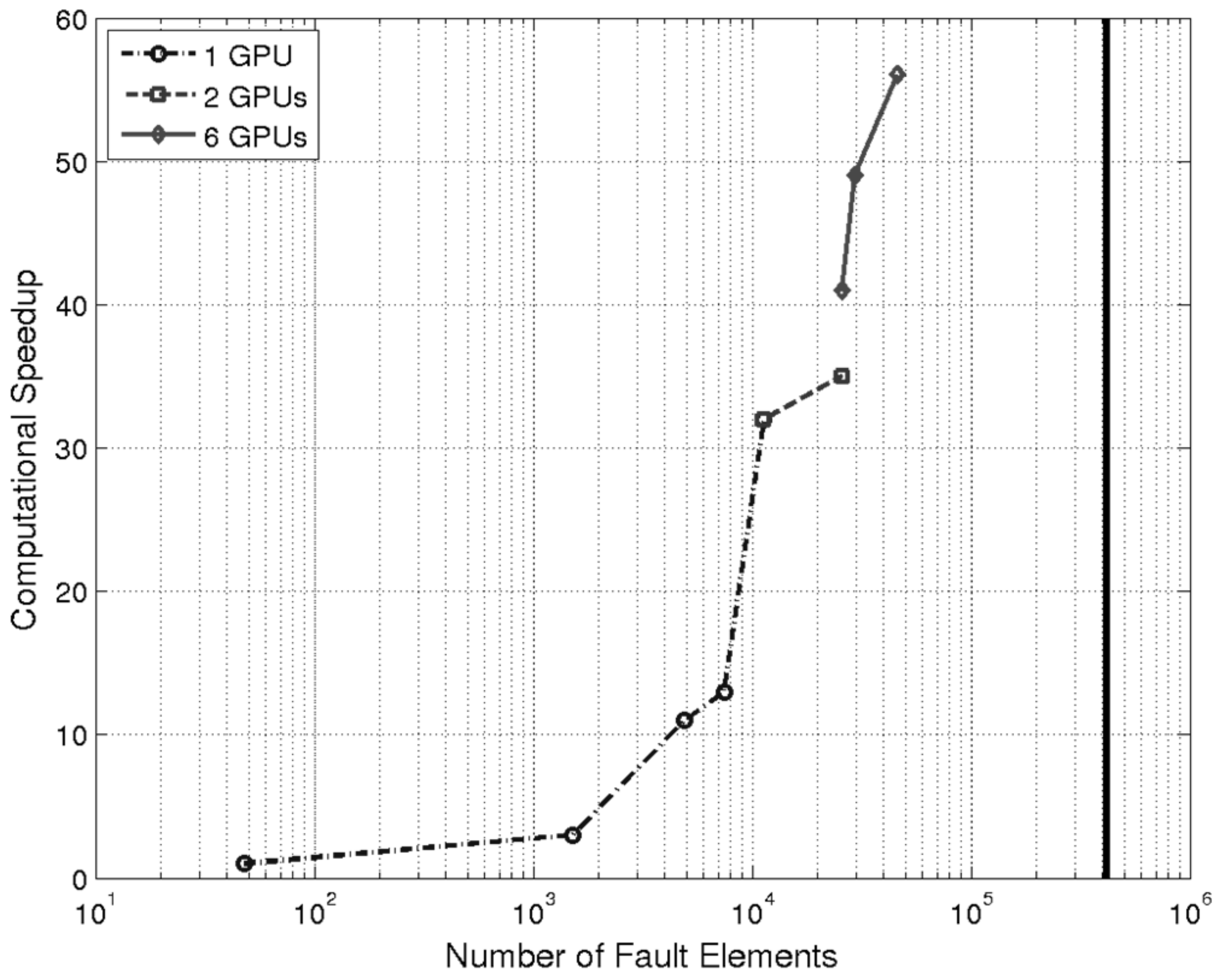

Figure 5. GPU Speedup as a Function of Problem Size and Number of Available GPU Devices