Understanding Long-Term Earthquake Behavior through Simulation

Understanding earthquake behavior is crucial for predicting earthquakes' effects. One way of understanding long-term earthquake behavior is through simulation. Virtual California is a program developed to generate ensembles of earthquakes in a specified fault system. The theory underlying the program, some recent results, and several optimizations used to improve performance are presented.

Earthquakes are among the most devastating natural hazards faced by society. Many areas of the world are at high risk of experiencing potentially disastrous seismic events. Two regions at significant risk for disruption from catastrophic earthquakes are Japan and California. In 2011, we saw the effects of the Tohoku earthquake in Japan, resulting in large loss of life (about 15,867 lives) and extreme economic disruption (about US$235 billion). In California, the combination of one of the largest fault systems in the world and one of the world's most densely populated regions could prove particularly destructive. It has been estimated that a repeat of the 7.9 magnitude (M7.9) 1906 San Francisco earthquake could cause as much as $84 billion in damages.[1]

Earthquakes typically occur on preexisting faults. Displacements on these faults are governed by *stick-slip behavior* and *elastic rebound*. Because of the relative motion of the tectonic plates, stress on a fault increases. The fault sticks until the stress reaches the fault's frictional strength, then the fault slips and elastic rebound occurs. Fault systems, however, are difficult to study. They're affected by dynamics at many different scales in time and space, and they're physically inaccessible, making direct measurements of their properties problematic.

Earthquake fault systems are chaotic and highly unpredictable, similar to the atmosphere's behavior, for which weather forecasts and global climate models rely heavily on simulations. Likewise, computer simulations are an invaluable tool in better understanding how fault systems operate. Virtual California is a computer simulation that models the earthquake fault system in California,[2,3] although the simulation code's core can be used for any fault system. Here, we describe the Virtual California code's various elements and discuss results from a simulation. We also discuss details of implementing the code and optimizations to improve simulation performance.

Ensemble Domain versus Time-Domain Simulations

Earthquake fault simulations generally fall into one of two categories: *time domain* and *ensemble domain*. Before we describe Virtual California in detail, it's important to draw a distinction between these two simulation types. A time-domain fault simulation attempts to solve a set

1521-9615/12/$31.00 © 2012 IEEE
Copublished by the IEEE CS and the AIP

Eric M. Heien and Michael Sachs
University of California, Davis

of differential equations that govern the system's evolution. These simulations usually employ an approximation scheme such as finite-element analysis. The result of these solutions is a function that has time as an independent variable, so in principle the system's state at any given time is encoded in the solution. Aside from the considerable computational difficulty of applying the time-domain approach to fault systems, problems arise from the sensitivity of these solutions to initial conditions. As we mentioned, collecting information about the current state of a fault is difficult, which in turn makes precise definitions of initial conditions difficult.

An ensemble-domain simulation sidesteps the problem of initial conditions by looking for the system's most likely states, given some set of external parameters. Instead of generating a single "history" of the system, many histories are created; this *ensemble* is then the basis of statistical analysis. Because the ensemble is a combination of many different paths the system can take, it's less sensitive to where those paths begin. Ensemble-domain simulation is used in Virtual California, as well as in the well-known Monte Carlo algorithm and many other simulators.[4–6]

Virtual California's Components

Virtual California consists of three major components: a fault model, a set of quasistatic interactions or *Green's functions*, and an event model. Despite the name, the only component of Virtual California that's California-specific is the fault model. This model can be changed to any physically realistic fault model and still correctly work with the simulation physics and event model.

Fault Model

The fault model that's currently in use is based on the Uniform California Earthquake Rupture Forecast version 2 (UCERF2)[7] fault model developed by the Working Group on California Earthquake Probabilities. The model includes 65 sections, roughly corresponding to known faults in California, with some faults modeled by multiple sections. Each fault plane is meshed into square elements that are roughly 3 × 3 km (see Figure 1), for a total of 8,395 elements. Each element is then given a slip velocity along a fixed rake vector and a failure stress. The rake vector always lies in the element's plane; the vector's angle (which is relative to the Earth's surface) and the slip rate are measured quantities that are taken from the UCERF2 model. The failure stresses, however, aren't measured quantities—they're tuned to reproduce observed event recurrence times for the fault sections.

Figure 1. An example of a Virtual California fault model. This particular model is based on the Uniform California Earthquake Rupture Forecast version 2 (UCERF2) fault model,[7] which is the model used in this article.

The fault model's format follows the specification developed by the Southern California Earthquake Center earthquake simulators group. Details of the specification can be found at http://scec.usc.edu/research/eqsims.

Element Stress Interactions

Interactions between fault elements are governed by quasistatic Green's functions. The effect that one element has on another depends on their relative positions and orientation, and on the direction of their slip displacements. The interactions are considered quasistatic because the large-scale fault geometry doesn't change during the simulation, therefore the elements' orientation and position don't change and neither do the Green's functions. This is an idealized model—over long time periods the actual fault systems' geometry will change. However, because the rate of change in a fault system's geometry is so slow—the average slip velocity in the model described above is 1.76×10^{-10} meters per second (m/s)—Virtual California is designed to explore histories on time scales in which the slow slip won't cause major changes in the fault system. For example, 10,000 years of slip at the average rate results in approximately 55 m of displacement, which is about 2 percent of an element's size.

Because the faults don't actually move, a technique called *backslip* is used to model the effects of stress buildup and release along the fault plane. Elements are treated like leaf-springs, which are stretched away from equilibrium in the opposite directions of the actual long-range plate motions that cause stress buildup. This stretching creates stress on the elements; when the stress on an

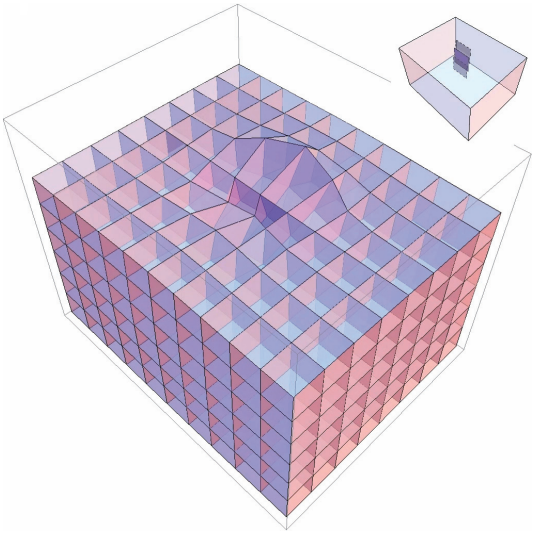

Figure 2. The 3D displacement field created by vertical backslip. This displacement field demonstrates how movement of a fault produces uneven changes in the surrounding area.

element increases to the failure stress point, the element fails, slip occurs, and the element returns to its equilibrium state. On an actual fault, this failure results in a change of the displacement on either side of the fault plane, which slightly changes the fault geometry. In a backslip model, the equilibrium position for a failed element is the same as its initial position, so failed elements don't actually change their position.

As the elements are stretched, they change the amount of stress that's applied to all other elements in the model—interactions that are governed by the Green's functions. Every element is moved a unit distance along their slip-velocity vectors and the stress changes are calculated for every other element. The stress changes on any element in the simulation due to changes on all other elements is given by the following:[2]

$$\sigma_{ij}(\mathbf{x},t) = \int dx'_k T_{ij}^{kl}(\mathbf{x}-\mathbf{x}')s_l(\mathbf{x},t), \quad (1)$$

where $s_l(\mathbf{x},t)$ is the 3D slip of element l and $T_{ij}^{kl}(\mathbf{x}-\mathbf{x}')$ is the interaction tensor.[8] In the case of Virtual California, the field is evaluated only at elements and slip is allowed only along the rake vector defined by the model (described in the previous section). Under these conditions, Equation 1 simplifies to

$$\sigma_{ij}^{A}(t) = T_{ij}^{AB}s_B(t), \quad (2)$$

where A and B run over all elements.

The actual values of the Green's functions are calculated using an implementation of Okada's half-space deformations.[8] Examples of the output generated by this implementation of Okada's interactions are shown in Figures 2 and 3.

Rupture Event Model

Virtual California's event model consists of two phases: *long-term slip* and *rupture propagation*. The long-term slip phase models the time between earthquakes when stress builds up on the faults because of long-term plate movement. This involves applying backslip to all of the elements at their model-defined slip velocities. The long-term slip phase ends when any element reaches a defined failure stress. Because the interactions are elastic, the relationship between slip and stress is known (see Equation 2), and it isn't necessary to evolve the system step by step during this phase. Rather, the simulation time is directly advanced to the point at which the next element fails, and then the rupture-propagation phase begins.

During the rupture-propagation phase, the system releases accumulated stress through a cascading series of fault-element failures. The first element to fail is allowed to slip back toward its equilibrium position. The amount the element slips, Δs, is related to stress drop defined for the element in the model, $\Delta\sigma$, by the following:[2]

$$\Delta s = \frac{1}{K_L}\frac{N_{ef}}{N_e}\Delta\sigma.$$

The factor N_{ef}/N_e is related to the rupture's current size: N_{ef} is the number of failed elements on a particular fault and N_e is the total number of elements on that fault. This factor is used to prevent small ruptures from slipping too much.

After the initial element slips, a new stress state is calculated for the entire system using Equation 2. Additional elements will now fail if their stress values exceed the model-defined failure stress. To encourage rupture propagation, a dynamic triggering mechanism is used. Elements on the same fault and physically close to a failed element are allowed to fail at a lower stress than the defined failure stress if the amount of stress accumulated during the rupture is greater than a predefined dynamic triggering factor η:

$$\frac{\sigma_f - \sigma_i}{\sigma_i} > \eta, \quad (3)$$

where σ_i is the stress on an element before the event began and σ_f is the new stress on the element.

The dynamic trigger factor approximates the stress intensity factor at the tip of a propagating rupture. This process continues until there are no more failures, at which point the event is over. If failed elements haven't slipped back to their equilibrium points because of their initial failures, they're allowed to fail again and release more stress into the system.

Simulation Computational Overview

We'll now outline the computational flow of Virtual California. Figure 4 shows the execution path in a parallel simulation running on multiple processors on either a cluster or multicore machine. The execution is divided into three distinct phases: initialization, long-term stress interaction, and rupture propagation. Because a naive implementation of this simulation would be heavily bound by computation and communication, Virtual California uses techniques (discussed later) to make this more tractable.

The initialization phase begins by parsing the specified model and simulation parameters. If the simulation is running on multiple processors, it partitions the fault elements. The goal of the partitioning is 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 all of the model elements' stress influences on local elements.

The simulation's core involves cycling between two phases—the first determines long-term stress buildup in the system and the second propagates a rupture through the system—indicated by blue and red, respectively, in Figure 4. 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

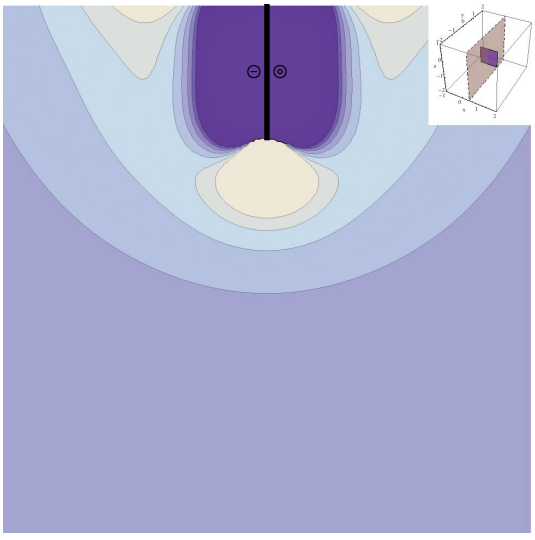

Figure 3. The shear stress field σ_{xy}, created by horizontal backslip, as viewed from an element's (a) front and (b) top. The arrows indicate the backslip's direction. The tan color indicates $\sigma_{xy} > 0$ and blue indicates $\sigma_{xy} < 0$.

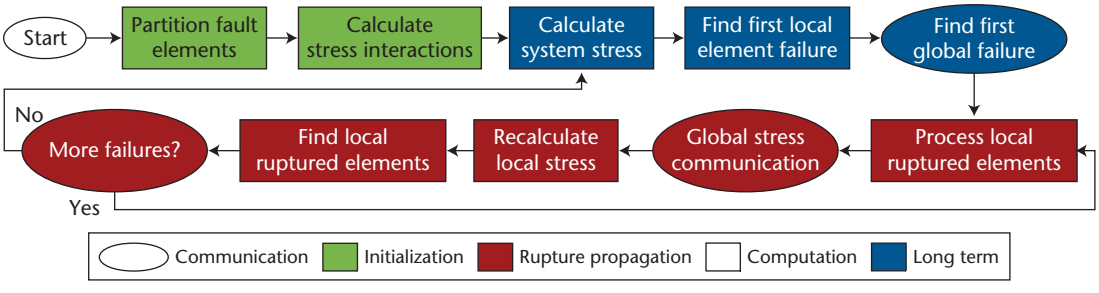

Figure 4. Execution flow of Virtual California on a parallel system. The execution is divided into three distinct phases: initialization, long-term stress interaction, and rupture propagation.

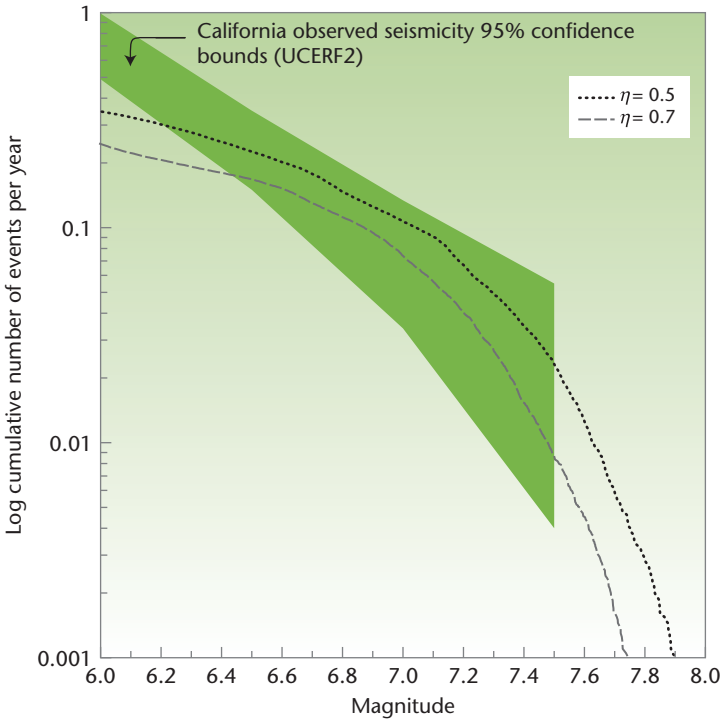

Figure 5. Frequency-magnitude statistics for two Virtual California simulations, each with a different value of the dynamic triggering factor η. The green area represents the 95 percent confidence bounds on the observed seismicity in California, as reported by UCERF2.[7]

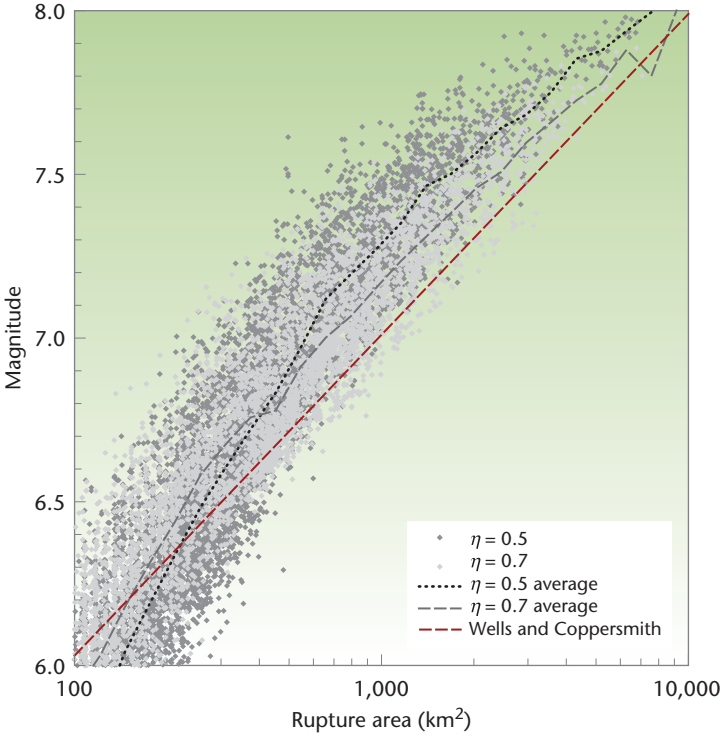

Figure 6. Magnitude versus rupture area for all events in two Virtual California simulations, each with a different value of the dynamic triggering factor η. The red dashed line is the empirical relationship, as reported in research by Donald Wells and Kevin Coppersmith.[10]

rupture, which is then globally reduced to find when and where the first rupture will occur.

Once this is determined, the rupture is propagated through the system using the mechanisms described earlier. 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) ruptured. If any processors experience further ruptures, the rupture-propagation phase continues. This phase generally involves multiple propagation steps until the earthquake is finished. Once there are no more ruptures, the simulation returns to the long-term stress calculation. After a specified number of simulation years have elapsed, the simulation ends.

A Virtual California Simulation

To illustrate the types of results that Virtual California produces, we ran two simulations on the model shown in Figure 1. Each simulation used a different value of the dynamic triggering factor η (see Equation 3), and both of them were run for 20,000 simulated years.

Figures 5 and 6 show the results. We used these plots because they allow a direct comparison with observed quantities. Thus, we can evaluate how well Virtual California represents seismicity in California.

A particularly important scaling relation in seismology is the Gutenberg-Richter frequency-magnitude relation:

$$\log N_c = a - bM,$$

where N_c is the total number of earthquakes with magnitude greater than M, b is a near-universal constant in the range $0.8 < b < 1.1$, and a is a measure of the level of seismicity. What Equation 4 tells us is that for roughly every 10 M5.0 earthquakes, for example, we can expect roughly one M6.0 earthquake. Figure 5 shows the Gutenberg-Richter relation for the results of the Virtual California simulations. We provide results for two values of the dynamic triggering factor η. For large η, rupture propagation is inhibited and there are fewer large earthquakes. For small η, ruptures propagate freely and large earthquakes dominate. This figure shows that an earthquake that's roughly M7.6 or larger is expected about every 100 years. Figure 5 also shows the observed seismicity in California (as reported by UCERF2).[7] The drop-offs for events below M ~ 6.3 and above M ~ 7.3 are related to limiting aspects of the fault

model's geometry. An earthquake's magnitude is related to the surface area of ruptured faults as follows:[9]

$$M \sim \log(SA), \quad (4)$$

where A is the event rupture area and S is the event slip. Hence, the fall-off of events at large magnitude is related to the maximum surface area of faults in the model. The fall-off at low magnitudes is related to the element size. Smaller elements would allow smaller magnitude events to occur.

Several other empirical relationships between earthquake observations have been reported.[10] As an example of how the output of Virtual California compares to these relations, Figure 6 shows the relationship of the event rupture area to event magnitude. There's some scatter in the Virtual California data—however, the relationship reported in research by Donald Wells and Kevin Coppersmith[10] is based on 148 events, while there are more than 100,000 events in the Virtual California database. It's encouraging that the simulation does so well in reproducing the empirical relation over such a large number of events.

Simulation Optimizations

The physics underlying Virtual California requires a significant amount of computing power to properly calculate. To make simulations tractable, we use a variety of optimizations to improve performance and allow fast simulation of fault systems.

Trading Memory for Cycles

The stress interaction calculation in Virtual California is similar to N-body calculations used in molecular dynamics and astrophysics simulations. Rather than determining the gravitational influence of one body on another, the calculations in Virtual California determine the stress caused by the movement of an element on other elements or on itself. However, unlike typical N-body calculations, which involve on the order of tens of floating-point operations per interaction, this calculation is highly computationally intensive—requiring tens of thousands of operations per interaction (see Table 1). The precise number of operations depends on the orientation of the faults. For example, calculating a single time step for a model with 100,000 elements would require up to 4.30×10^{14}, or roughly 36 CPU hours on a modern processor. Furthermore, it would be highly difficult to accelerate this calculation on GPUs or similar vector processors because of the code's numerous conditional branches.

Table 1. Operations required to calculate a single element–element stress interaction.

Operation	Minimum	Maximum
Addition	3,622	10,804
Multiplication	8,708	25,566
Square Root	260	780
Branch	1,148	3,924
Other	571	1,923
Total	14,309	42,997

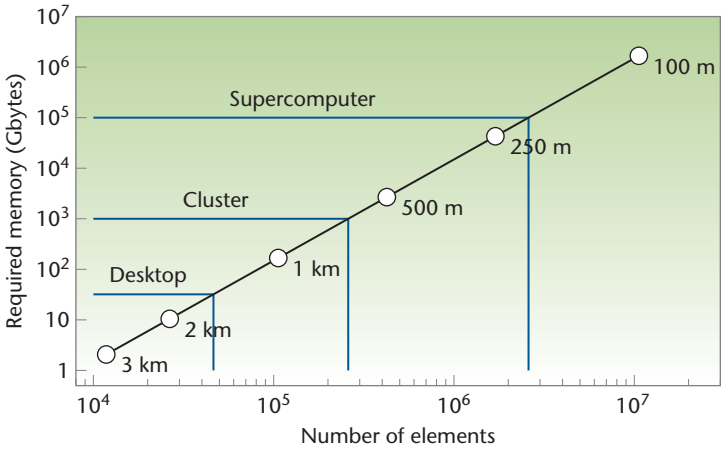

Figure 7. Memory requirements for a full California simulation at varying resolutions. Finer resolution greatly increases the number of elements involved, which in turn leads to much larger memory requirements.

Calculating the Green's function to determine the stress interaction between elements is computationally intensive; and, because the simulation uses a backslip model, there's a tight limit on how inaccurate these calculations can be. Taking advantage of this, the interactions are infrequently calculated and instead are stored as a distributed matrix $\boldsymbol{G}$, representing T_{ij}^{AB} from Equation 2. Element $\boldsymbol{G}_{a,b}$ of this matrix indicates the change in shear stress on element a, caused by a unit distance of movement by element b. Each processor calculates and records the stress influence of all the other elements on the locally owned elements. Using this technique, Virtual California effectively trades memory for calculation and can perform simulations much faster.

However, this optimization makes memory a limiting resource for larger models. Figure 7 shows how the memory requirements for a full California simulation grow as the model resolution increases. For elements of size 1.5×1.5 km, the simulation will require 43,000 elements and

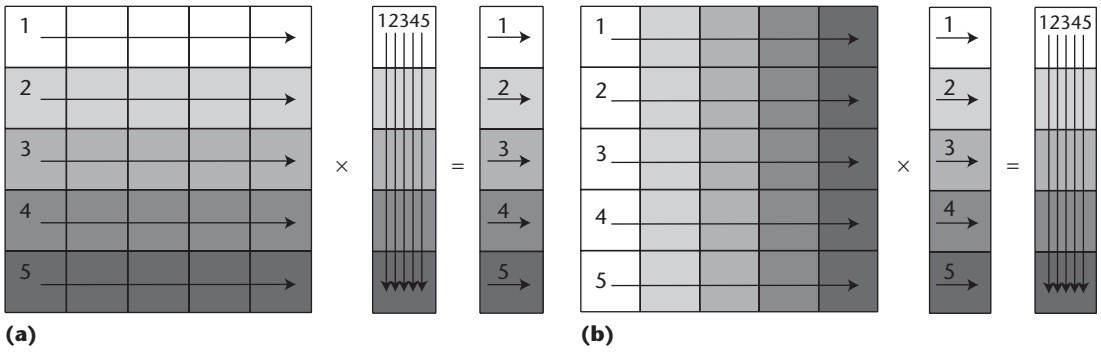

Figure 8. The (a) original matrix calculation and (b) transposed matrix calculation for sparse vectors. Arrows and numbers indicate the data used in each step.

Table 2. Speed improvement from sparse-dense optimizations.

Method	Long Term	Rupture	Total
Original	1.00×	1.00×	1.00×
Ignore zeros	1.00×	40.60×	8.44×
Transpose	1.11×	1.17×	1.17×
Both	1.11×	50.48×	9.50×

fit on a high-end desktop computer. If the element resolution drops to 700 × 700 m, though, it will require 250,000 elements and barely fit into the memory of a medium-sized cluster (1 Tbyte of memory). With elements of size 200 × 200 m, the simulation won't fit on most supercomputers (100 Tbytes of memory). As we describe later, we address this memory limitation using approximation algorithms that let the simulation scale well below 100 m of resolution.

Mixing Sparsity and Density

During a simulation, we calculate an element's stress using the distributed matrix $\boldsymbol{G}$. In the long term, nearly every element is moving and so stress is calculated as a matrix-vector multiplication, where both the matrix and vector are dense (mostly nonzero). However, during rupture propagation, only a few elements influence stress, and thus the vector is sparse (mostly zero) while the matrix remains dense.

In a typical simulation, there's between a 1:10 to 1:20 ratio of dense-dense to dense-sparse multiplications. This is because ruptures generally propagate over multiple elements, causing several rounds of stress calculation during an earthquake. The common strategy in dealing with sparse multiplication is to avoid operations on matrix and vector zero-value entries, because these will contribute nothing to the result.

Figure 8a shows an example of the standard matrix-vector multiplication method. The arrows and numbers indicate the order of operations and which elements they use—for example, to calculate the first entry in the result vector, we use the matrix's top row and the input vector's entire column. For a dense matrix-sparse vector multiplication, the technique of avoiding zero value entries in the input vector doesn't improve performance much. The reason is that, to calculate each entry in the result vector, the sparse input vector must be completely traversed and all corresponding matrix row entries must be examined.

A simple way of handling this is to test each element in the sparse input vector and perform only the associated multiplications and sums if the result is nonzero. The matrix data multiplied with a given input vector element is stored in a matrix column. Table 2 shows the results of this method. If this method is used with dense-dense multiplication, performance drops by a factor of 3.48 times, so a separate function must be used for dense multiplication that has the same performance as the original. For the sparse input vector, performance is drastically increased, running more than 40 times faster than the original method. However, there's still a performance penalty because of poor cache efficiency (due to accessing matrix columns).

The solution is to store the matrix in a transpose form and calculate the result in a different order. Rather than calculating each result entry at a time, the transposed method calculates the contribution of each input vector element to the solution. As Figure 8b shows, the transpose method accesses memory along the rows of the transposed matrix (columns of the original matrix) and therefore has much better cache efficiency. By storing the matrix in transpose form, there's a slight performance improvement in both dense-dense and dense-sparse multiplications.

Even better performance is achieved by combining the two methods. Storing the matrix in

transpose form and ignoring zero-value entries for dense-sparse calculations lets the matrix-vector multiplication avoid unneeded zero-value multiplications and still maintain good cache efficiency while accessing the matrix columns (rows in the transposed matrix). This method achieves a speedup of more than 50 times for the rupture calculations and improves total simulation speed by a factor of nearly 10 times.

Streaming SIMD Extensions (SSE) Optimizations

The majority of time in a Virtual California simulation is spent performing the matrix multiplication. Fundamentally, there are two issues with performing fast calculations: transferring the data from memory to the processor and performing the calculation on the processor. Virtual California can use the streaming single instruction, multiple data (SIMD) extensions, or SSE, to improve performance in each of these regards.

First, there are optimizations for calculation. As with most other SIMD instruction sets, SSE provides functionality for performing multiple mathematical operations with a single instruction. In the case of SSE, for example, a single instruction can perform four simultaneous 32-bit (or two simultaneous 64-bit) floating-point multiplications. Virtual California uses 64-bit floating-point instructions to multiply (`_mm_mul_pd`) and accumulate (`_mm_add_pd`) values in the matrix multiplication.

The second optimization is related to memory and caching. SSE provides instructions (`_mm_prefetch`) to prefetch data from memory to a cache closer to the processor. By carefully prefetching data, the processor can perform calculations on element X while elements $X + 1, \ldots, X + N$ are being moved from memory into cache, improving overall performance. In combination with this, the multiplication loop over matrix entries can be unrolled to perform several multiplications while prefetching data elements that will be used soon.

Table 3 shows the effect of these improvements on performance, using a single core of a 2.2-GHz Intel Core i7 with 1,333-MHz double data rate, type 3 (DDR3) RAM. Calculation speeds were determined by clock timings, and the transfer rate was inferred based on the amount of data required. As the table shows, Virtual California achieves a nearly 41 percent speedup just by switching to SSE instructions. Combining loop unrolling with data prefetching yields up to an additional 15 percent speedup.

Table 3. Speed improvement from Streaming SIMD Extensions (SSE) optimizations.

Method	Calculation (megaflops)	Memory (Gbytes/s)	Improvement
Original	999	7.44	–
SSE No Unroll	1,412	10.52	41.3%
SSE Unroll x2	1,461	10.89	46.2%
SSE Unroll x4	1,429	10.65	43.0%
SSE Unroll x8	1,482	11.05	48.3%
SSE Unroll x2, Prefetch	1,484	11.06	48.5%
SSE Unroll x4, Prefetch	1,514	11.28	51.6%
SSE Unroll x8, Prefetch	1,556	11.60	55.8%

It's worth noting that these results indicate that the multiplication performance is bound by memory bandwidth, with most of the transfer rates around the theoretical maximum of the DDR3 RAM (10.4 Gbytes/s). Values higher than this are likely because some of the data are already in the processor cache. The ratio of operations to input data (64-bit doubles) is roughly 1:0.95, which is close to the expected value from this sort of multiplication (1:1 using a fused multiply-accumulate). This memory bottleneck could be addressed by using 32-bit floating-point values or by compressing the matrix data.

Speculating on Communication

The program flow in Figure 4 shows three main communication points. The global nature of these communications is a serious bottleneck to simulation performance on a parallel system. To decrease the frequency of communication and improve parallel performance, Virtual California uses a type of speculative execution during rupture propagation.

Figure 9 shows simulation runtimes for a 13,398-element, 100,000-year simulation on a simple cluster system using different numbers of processors. The cluster has 16 nodes, with two 2.4-GHz quad-core Xeon processors per node and a Gigabit Ethernet network. For a single processor, most of the time is spent in the long-term stress calculation and the initial Green's function calculations. As the number of processors increases, these calculations scale well, but the time spent in communication significantly rises. Indeed, by 16 processors, the majority of simulation time is spent communicating, resulting in poor scalability.

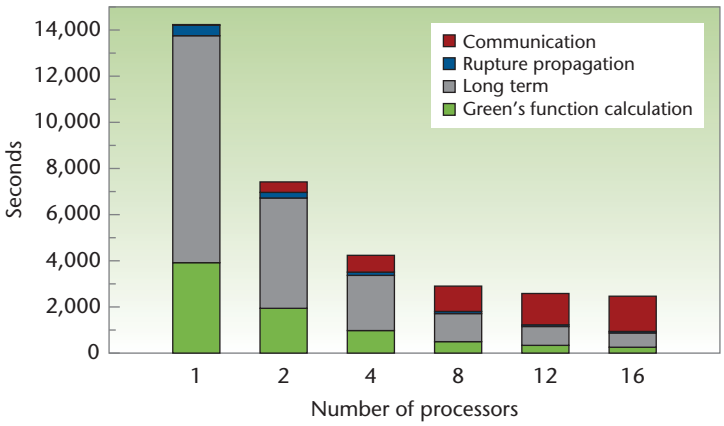

Figure 9. Parallel simulation time breakdown. As the number of processors increases, the calculations scale well, but the time spent in communication significantly rises. Indeed, by 16 processors, most of the simulation time is spent communicating, resulting in poor scalability.

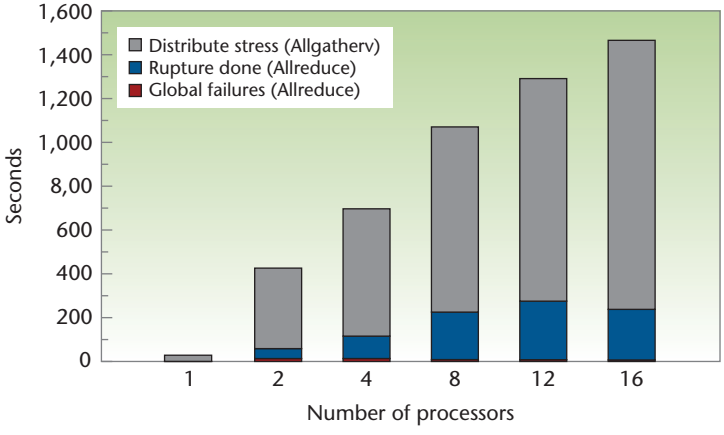

Figure 10. Parallel communication-time breakdown. For large numbers of processors, the majority of time is spent distributing stress values.

Figure 10 shows the breakdown in time for different types of communication. For all numbers of processors, the majority of time is spent distributing stress values over the processors. This is required because the changes in stress during a rupture potentially lead to other elements rupturing. However, at a given rupture propagation step, it's highly unlikely that the rupture will spread a significant distance past where it already has traveled.

Parallel simulations can take advantage of this by only communicating stress changes once a rupture is likely to pass over a processor boundary. Figure 11 details the scheme's flow. When rupture propagation begins, the simulation starts in standard mode, where it communicates stress changes at every rupture propagation step. If the simulation predicts that the rupture won't pass a processor boundary in the next step, it changes to localized mode, where it doesn't communicate. Once the ruptures have ended or are predicted to pass a boundary, the processor in localized mode does a global stress communication for all ruptures that occurred and the simulation determines whether the rupture would have moved across a processor boundary. If the rupture remained localized as predicted, the simulation continues as normal. However, if the prediction was incorrect and the rupture spread past a boundary, the simulation is rewound to the rupture propagation's start and rerun entirely in standard mode.

In this technique, the key parameter is the distance of the rupture to the processor's model boundary, which determines when the processor will enter localized mode. If this distance is too close, ruptures will pass the boundary more than expected and the rupture propagation will frequently require rewinding. However, if the distance is too far, the simulation will rarely enter localized mode and there will be the same scalability problems as the original simulation.

Figure 12 shows the results of using this technique with the fault model and cluster for 8, 16, and 32 processors. In this case, the optimal boundary distance is roughly 5 km, such that speculation works but localized mode isn't entered too frequently. Using this boundary distance decreases communication time up to 66 percent and earthquake simulation time up to 47 percent. The optimal boundary distance was determined empirically by testing different ranges (see Figure 12). The optimal value will depend on the simulation's physics, specifically on the strength of element interaction based on the Green's function calculation. Future work will investigate how this distance changes, depending on the element size, and could involve adaptive boundary distances that change depending on previous success rates and quickly learning the optimal strategy for any input model.

Saving Memory with Barnes-Hut

Earlier, we noted the problem of memory-usage-restricting high-resolution simulations. To address this problem, Virtual California can use a variation on the classic Barnes-Hut algorithm common in N-body simulations.[11] This algorithm lets simulations approximate the forces between N bodies in a simulation using $O(N \log N)$ operations rather than $O(N^2)$. To do this, it hierarchically divides the simulation space and treats groups of distance bodies as a single representative body. In N-body simulations, this reduces computation—that is, rather than calculating the

interactions with many distant bodies, it calculates the interaction with a single representative body. If the bodies are reasonably distributed, then many interactions can be approximated by a single calculation.

Instead of minimizing calculations—which Barnes-Hut is commonly used for—Virtual California applies this algorithm to minimize memory usage such that the memory required for a model with N elements grows as $O(N \log N)$. We accomplish this by averaging the interactions from multiple distant elements into a single interaction. Recording the single interaction in the matrix uses less memory, which in turn makes smaller-resolution simulations feasible. The distance at which grouping occurs depends on the ratio of the grouped elements' bounding-box size to the distance from the target element. This ratio is set as parameter θ, described in detail elsewhere.[11] Virtual California uses a default θ of 0.1—well below the common value of near 1, and thus higher in accuracy.

The Barnes-Hut algorithm first subdivides the model space, typically by using a quad tree for 2D or an octree for 3D. Figure 13 shows a visualization of part of the octree for the California model with 1-km elements. Blue boxes indicate unrefined octree nodes, while the smaller red boxes indicate well-refined sections containing a single element.

Figure 14 shows a comparison between the original memory requirements and those for Barnes-Hut. These requirements don't increase linearly because halving the element size results in four times the number of elements. Even so, memory requirements grow much slower using the Barnes-Hut algorithm. At an element resolution of 1 km, the original method requires 10 Gbytes of memory while the Barnes-Hut method requires 3.8 Gbytes. With 100-meter elements, the memory requirements are 100 Tbytes versus 4 Tbytes—a difference that enables simulations of this resolution on medium-sized clusters, which previously wasn't possible even on most supercomputers.

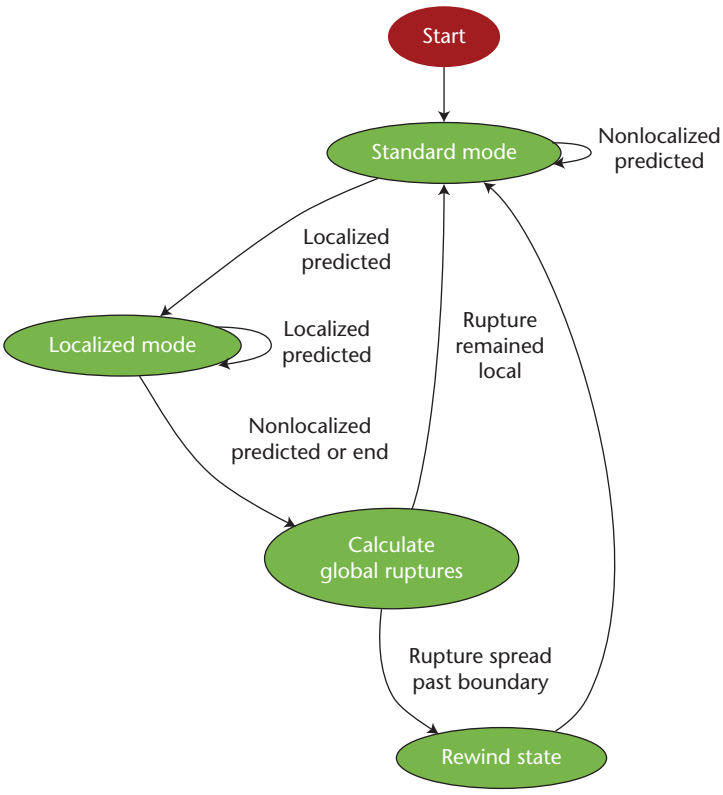

Figure 11. Speculative execution flow. Communication between processors occurs in standard mode, when ruptures are predicted to spread across processors.

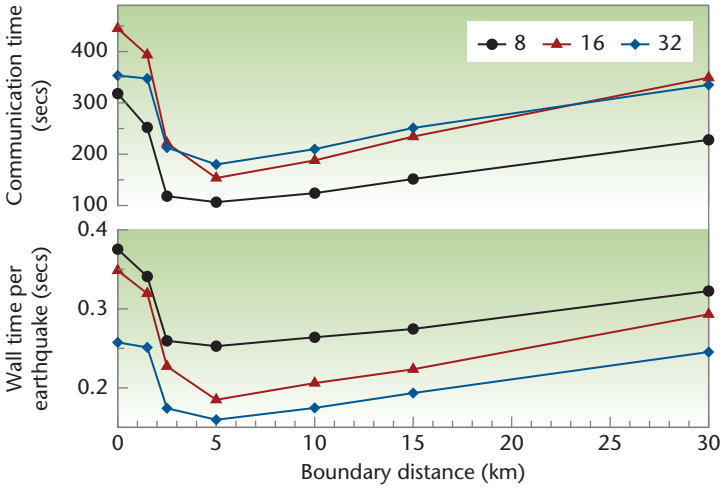

Figure 12. Effectiveness of speculative execution in reducing communication time and improving simulation speeds. The results are shown for 8, 16, and 32 processors.

Understanding earthquake behavior is important for disaster prevention and management. The Virtual California program helps perform ensemble domain simulations that are statistically similar to actual earthquakes and will aid in predicting and understanding earthquakes in the future. These simulations use a variety of advanced computer science techniques, such as speculative execution and approximation algorithms, to enable fine-resolution, high-performance simulations for improved accuracy and better science. This work will be extended in the future to include more advanced physical models, such as long-term

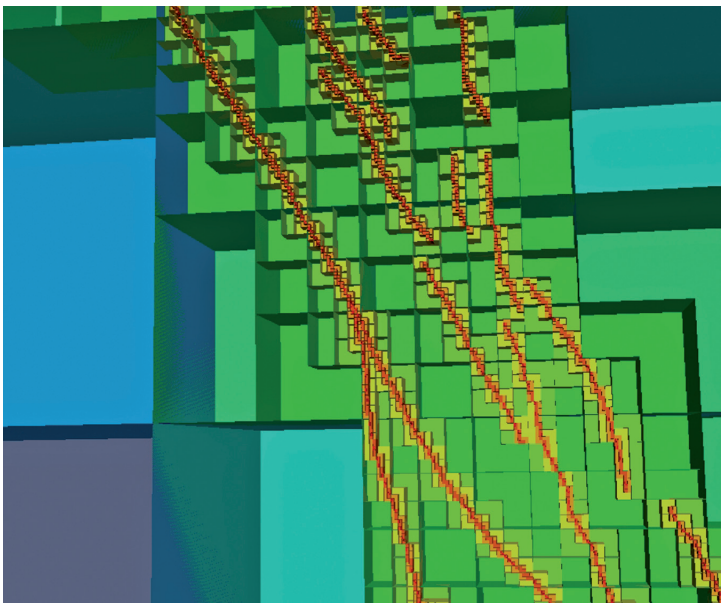

Figure 13. Visualization of octree for California 1-km model. The fork in the long fault (San Andreas) near the center is close to San Francisco.

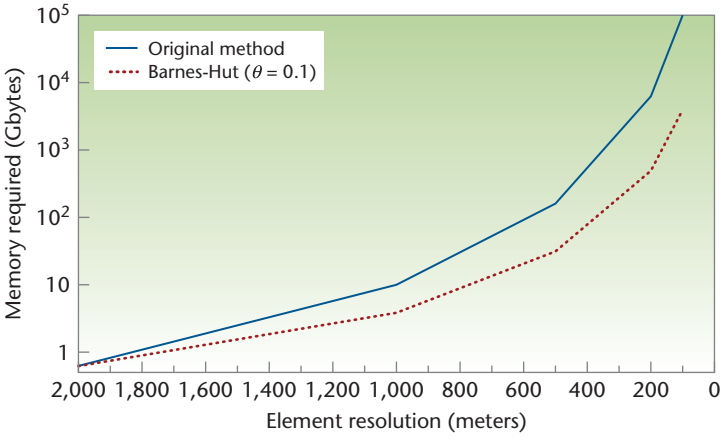

Figure 14. Memory requirements for the Northern California model at different element sizes. The requirements don't increase linearly because halving the element size results in four times the number of elements.

evolution of faults and complex heterogeneous crustal models, to help improve our understanding of the Earth.

References

1. R. Chen, D. Branum, and C.J. Wills, *California Geological Survey—2009 Earthquake Loss Estimation*, Calif. Geological Survey, 2009; www.consrv.ca.gov/cgs/rghm/loss/Pages/2009_analysis.aspx.
2. P. Rundle et al., "Virtual California: Fault Model, Frictional Parameters, Applications," *Pure and Applied Geophysics,* vol. 163, no. 9, 2006, pp. 1819–1846.
3. M. Yıkılmaz et al., "A Fault and Seismicity-Based Composite Simulation in Northern California," *Nonlinear Processes in Geophysics,* vol. 18, no. 6, 2011, pp. 955–966.
4. S.N. Ward, "A Synthetic Seismicity Model for Southern California: Cycles, Probabilities, and Hazard," *J. Geophysical Research,* vol. 101, no. B10, 1996, pp. 22393–22418.
5. J. Dieterich and K. Richards-Dinger, "Earthquake Recurrence in Simulated Fault Systems," *Pure and Applied Geophysics,* vol. 167, no. 8, 2010, pp. 1087–1104.
6. F. Pollitz, "Epistemic Uncertainty in California-Wide Synthetic Seismicity Simulations," *Bulletin Seismological Soc. America*, vol. 101, no. 5, 2011, pp. 2481–2498.
7. E.H. Field et al., *The Uniform California Earthquake Rupture Forecast, Version 2 (UCERF 2),* US Geological Survey Open File Report 2007-1437, 2008; http://pubs.usgs.gov/of/2007/1437.
8. Y. Okada, "Internal Deformation Due to Shear and Tensile Faults in a Half-Space," *Bulletin Seismological Soc. America*, vol. 82, no. 2, 1992, pp. 1018–1040.
9. H. Kanamori and D. Anderson, "Theoretical Basis of Some Empirical Relations in Seismology," *Bulletin Seismological Soc. America*, vol. 65, no. 5, 1975, pp. 1073–1095.
10. D.L. Wells and K. Coppersmith, "New Empirical Relationships among Magnitude, Rupture Length, Rupture Width, Rupture Area, and Surface Displacement," *Bulletin Seismological Soc. America*, vol. 84, no. 4, 1994, pp. 974–1002.
11. J. Barnes and P. Hut, "A Hierarchical *O*(*N* log *N*) Force-Calculation Algorithm," *Nature*, vol. 324, 1986, pp. 446–449.

Eric M. Heien *is a researcher in the Computational Infrastructure for Geodynamics (CIG) group at the University of California, Davis. His research interests include parallel and distributed computing, volunteer computing, and high-performance computing for scientific simulation. Heien has a PhD in computer science from Osaka University. He's a member of IEEE. Contact him at emheien@ucdavis.edu.*

Michael Sachs *is a PhD student in the Department of Physics at the University of California, Davis. His work focuses on building and analyzing computer simulations of complex physical systems. Sachs has an MS in physics from the University of California, Davis. He recently received a NASA Earth and Space Science Fellowship for research related to Virtual California. Contact him at mksachs@ucdavis.edu.*