Relax-Miracle: GPU Parallelization of Semi-Analytic Fourier-Domain solvers for Earthquake Modeling

Sagar Shrishailappa Masuti, Sylvain Barbot
Earth Observatory of Singapore
Nanyang Technological University
Singapore
{smasuti,sbarbot}@ntu.edu.sg

Nachiket Kapre
School of Computer Engineering
Nanyang Technological University
Singapore
nachiket@ntu.edu.sg

***Abstract*—**
Effective utilization of GPU processing capacity for scientific workloads is often limited by memory throughput and PCIe communication transfer times. This is particularly true for semi-analytic Fourier-domain computations in earthquake modeling (Relax) where operations on large-scale 3D data structures can require moving large volumes of data from storage to the compute in predictable but orthogonal access patterns. We show how to transform the computation to avoid PCIe transfers entirely by reconstructing the 3D data structures directly within the GPU global memory. We also consider arithmetic transformations that replace some communication-intensive 1D FFTs with simpler, data-parallel analytical solutions. Using our approach we are able to reduce computation times for a geophysical model of the 2012 Mw 8.7 Wharton Basin earthquake from 2 hours down to 15 minutes (speedup of $\approx$8$\times$) for grid sizes of 512·512·256 when comparing NVIDIA K20 with a 16-threaded Intel Xeon E5-2670 CPU (supported by Intel-MKL libraries). Our GPU-accelerated solution (called Relax-Miracle) also makes it possible to conduct Markov-Chain Monte-Carlo simulations using more than 1000 time-dependent models on 12 GPUs per single day of calculation, enhancing our ability to use such techniques for time-consuming data inversion and Bayesian inversion experiments.

I. Introduction

The surface of the Earth constantly deforms due to the action of earthquakes, volcanoes and hydrological processes. Monitoring this deformation offers an opportunity to constrain the mechanical properties of the subsurface through modeling. One way to understand the deep properties of the lithosphere is to study the slow relaxation that follows large earthquakes or rapid changes of surface loads. The stress induced by these sudden disturbances spurs an accelerated visco-elastic transient that can deform rocks all the way down to the Earth's upper-mantle. Many analytical and numerical tools allow us to simulate scenarios of visco-elastic relaxation, including using the finite-element [1], boundary-integral [2] and Green's function methods [3].

The solution displacement or velocity can be obtained by convolving the forcing terms of the governing equation with an analytic Green's function requiring of the order of $O(N^2)$ operations, where N is the number of nodes in the mesh.

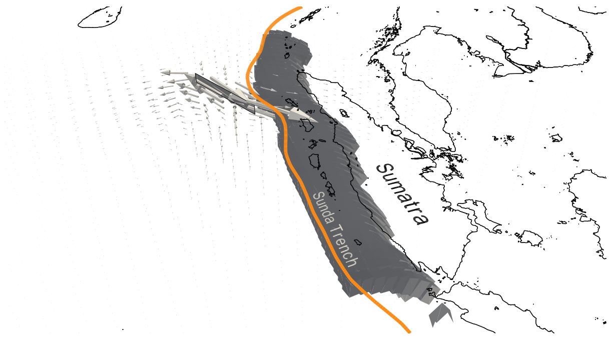

Fig. 1: Three-dimensional Relax simulation of the displacement caused by the 2012 Mw 8.7 Wharton Basin earthquake. The model incorporates a realistic source geometry for the quake and the Earth's mechanical properties around the Sunda trench offshore Sumatra, Indonesia.

When the elastic properties are assumed uniform inside the Earth, the convolution between the Green's function and the forcing terms can be evaluated in the Fourier domain, taking full advantage of the efficiency of the fast Fourier transforms. This method offers the best asymptotic scaling with mesh size, requiring only of the order of $O(N \log N)$ operations, which can be also efficiently parallelized on shared- and distributed-memory computers.

The Relax software implements a Fourier-domain Green's function [4], [5] to simulate three-dimensional models of visco-elastic relaxation following sudden stress changes induced by earthquakes, volcanoes and hydrological processes. The method is sufficiently flexible to incorporate realistic variations of visco-elastic properties and complex source models in space and time. For example, in Figure 1, we show a visualization of the result of Relax when analyzing the geophysical processes in action around the Sunda Trench off the Indonesian coast. The large, thick arrows represent a slip input into our model while the hundreds of smaller arrows are the resulting displacement fields generated by Relax. The method was also used to study the properties of the San

Andreas Fault around Parkfield, CA [6], and the mechanical properties of rocks in the Taiwanese accretionary prism [7]. Despite the relative efficiency of Relax compared to other numerical methods, multi-threaded computation of a single model still takes a several hours on a shared-memory computer for even the simplest datasets. This prevents us from using an automatized data inversion framework for realistic earthquake models. Acceleration of the Relax computation by large factors (i.e., 5–10×) would open the door to many applications, in particular Markov-chain Monte-Carlo methods for global parameter optimization and estimation of model uncertainties when fitting geophysical data.

Many other computational problems in science and engineering are structurally similar to Relax. They stress the capabilities of modern processing platforms in many ways: (1) raw compute-intensive nature of the arithmetic, (2) sheer volume of data being moved around (multi-GB active state), and (3) orthogonal data-access patterns. Promisingly, the numerics are inspired by natural phenomenon where parallelism, locality, reuse and spatial nature of the computational structures are abundant. GPUs are of particular interest in this context due to the availability of hundreds of data-parallel floating-point pipelines with large off-chip memory bandwidths unseen in general-purpose CPU-based hardware (see Section IV). GPUs have become relatively affordable and are increasingly easy to program through frameworks such as CUDA making them accessible to a broader community of users including geophysicists. In this paper, we describe the CUDA-parallelized GPU-implementation of the Relax computation (which we name Relax-Miracle – CUDA in Polish translates into "to work miracles") and associated bottleneck analysis and optimization.

The key contributions of this paper include:

- Parallelization analysis and optimization of OpenMP multi-threaded code for Relax – a semi-analytic Fourier-domain solver for earthquake simulations
- Optimized CUDA implementation of Relax (called Relax-Miracle) and detailed performance/scalability analysis of the complete Fortan code suitable for scalable parallelization.
- Demonstration of 7.7× speedup when comparing the performance of a 512·512·256-sized physical domain simulation between a 16-threaded Intel E5-2670 CPU implementation (MKL) versus an NVIDIA K20 GPU.
- Markov-Chain Monte-Carlo simulation of GPU-accelerated Relax-Miracle across 12 NVIDIA K20 GPUs to solve an inverse problem.

II. Background

We first describe the underlying mathematics of the Relax algorithm and motivate the challenges associated with its efficient parallelization.

A. *The Mathematics of Relax*

Relax evaluates the three-dimensional, time-dependent deformation that follows a stress perturbation. Displacement and velocity are obtained by solving the governing equations and boundary conditions for elastic deformation in a half space with a free surface. Static displacement and quasi-static instantaneous velocity are obtained by application of the elastic Green's function in the Fourier domain, after numerically Fourier transforming the body forces using a fast Fourier transform (FFT). Time evolution is simulated using a predictor-corrector Runge-Kutta explicit quadrature.

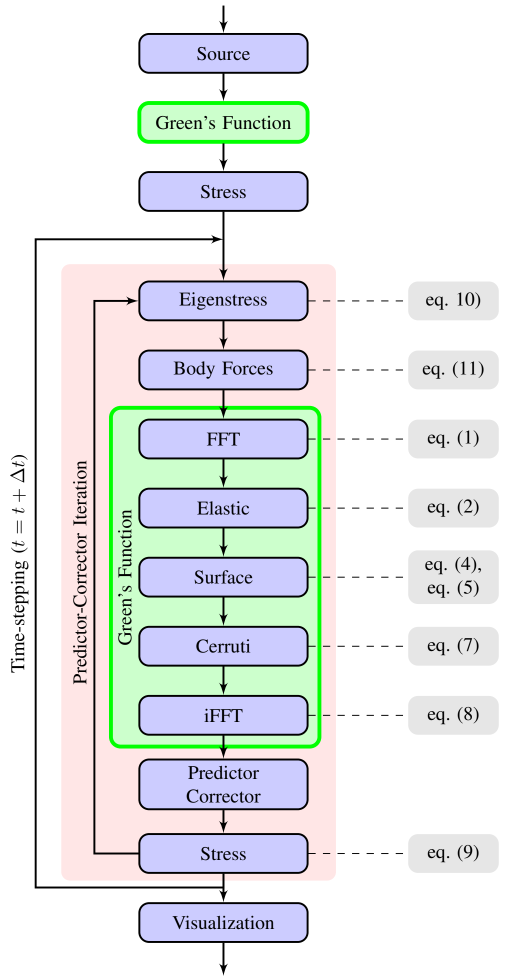

Fig. 2: Relax Flowchart showing the different numerical stages of the algorithm. Relax is an iterative computation that repeatedly computes solution to the elastic Green's function with nested loops for time integration using Runge-Kutta method and time-stepping

1) Elastic Green's function: Solution of the elastic equation in closed form is obtained with a pre-determined sequence of steps from prescribed body forces. We first Fourier transform the body forces $\mathbf{f}$ in three dimensions

$$\hat{\mathbf{f}}(\mathbf{k}) = \iiint_{\infty}^{\infty} \mathbf{f}(\mathbf{x}) e^{-i2\pi\mathbf{k}\cdot\mathbf{x}}\, d\mathbf{x} \tag{1}$$

where $\mathbf{x}$ and $\mathbf{k}$ are the coordinates and the wave numbers, respectively. This is approximated numerically using a discrete three-dimensional FFT. We then compute a particular solution analytically for every wave number using the matrix-vector product

$$\hat{\mathbf{u}}^p(\mathbf{k}) = \mathbf{M}\,\hat{\mathbf{f}}(\hat{\mathbf{k}}) \tag{2}$$

where we defined

$$\mathbf{M}(k_1,k_2,k_3) = \frac{1}{\mu(2\pi\beta^2)^2}\times \begin{bmatrix} (1-\alpha)\beta^2-\alpha k_1k_1 & -\alpha k_1k_2 & -\alpha k_1k_3 \\ -\alpha k_2k_1 & (1-\alpha)\beta^2-\alpha k_2k_2 & -\alpha k_2k_3 \\ -\alpha k_3k_1 & -\alpha k_3k_2 & (1-\alpha)\beta^2-\alpha k_3k_3 \end{bmatrix} \tag{3}$$

with $\beta^2 = k_1^2 + k_2^2 + k_3^2$. The dimensionless constant α is a function of the classic elastic parameters [4].

Eq. (2) requires a correction to satisfy the boundary condition at the surface. The correction has a closed-form representation in the Fourier domain [4] that can be estimated using

$$\hat{u}_3^p(k_1,k_2) = \int_{-\infty}^{\infty} \hat{u}_3^p(k_1,k_2,k_3)\,dk_3 \tag{4}$$

and

$$\hat{\mathbf{t}}^p(k_1,k_2) = \mu\int_{-\infty}^{\infty} \mathbf{S}\cdot\hat{\mathbf{u}}^p\,dk_3 \tag{5}$$

where we also defined

$$\mathbf{S}(k_1,k_2) = \begin{bmatrix} k_3 & 0 & k_1 \\ 0 & k_3 & k_2 \\ \omega\,k_1 & \omega\,k_2 & k_2+(1+\omega)\,k_3 \end{bmatrix} \tag{6}$$

with $\omega = -(1-2\alpha)/(1-\alpha)$. The correction is given in closed form by

$$\begin{aligned} \hat{u}_1^h &= \phi\big[-2B_1\beta^2 + \alpha\,\omega_1\,(B_1\omega_1+B_2\omega_2)\,(1-i\,\omega_3\,\phi) \\ &\qquad +\ \alpha\,i\omega_1\beta\,B_3\left(1-\alpha^{-1}-i\,\omega_3\,\phi\right)\big] \\ \hat{u}_2^h &= \phi\big[-2B_2\beta^2 + \alpha\,\omega_2\,(B_1\omega_1+B_2\omega_2)\,(1-i\,\omega_3\,\phi) \\ &\qquad +\ \alpha\,i\omega_2\beta\,B_3\left(1-\alpha^{-1}-i\,\omega_3\,\phi\right)\big] \\ \hat{u}_3^h &= \phi\,\alpha\,\beta\big[(\omega_1B_1+\omega_2B_2)\,\omega_3\,\phi \\ &\qquad -\ \beta\,B_3\left(\alpha^{-1}-i\,\omega_3\,\phi\right)\big] \end{aligned} \tag{7}$$

where the constants B_1, B_2 and B_3 depend on both $u_3^p(k_1,k_2)$ and $\hat{\mathbf{t}}^p(k_1,k_2)$. Finally, the solution in the space domain is obtained using the transform

$$\mathbf{u}(\mathbf{x}) = \iiint_{\infty}^{\infty} \hat{\mathbf{u}}(\mathbf{k})\,e^{i2\pi\mathbf{k}\cdot\mathbf{x}}d\mathbf{k} \tag{8}$$

which is performed using a FFT.

2) Time integration: Relax implements a predictor-corrector adaptive time step integration procedure. The velocity is first evaluated at time t to obtain displacement at time $t+\Delta t/2$. The velocity at time $t+\Delta t/2$ is then used to integrate displacement ($\mathbf{u}$) and inelastic stress (τ) from times t to $t+\Delta t$.

At any time, the stress $\boldsymbol{\sigma}$ can be evaluated with

$$\boldsymbol{\sigma} = \begin{pmatrix} \lambda\epsilon_{kk}+2\mu u_{1,1} & \mu(u_{1,2}+u_{2,1}) & \mu(u_{1,3}+u_{3,1}) \\ \mu(u_{2,1}+u_{2,1}) & \lambda\epsilon_{kk}+2\mu u_{2,2} & \mu(u_{2,3}+u_{3,2}) \\ \mu(u_{3,1}+u_{1,3}) & \mu(u_{3,2}+u_{2,3}) & \lambda\epsilon_{kk}+2\mu u_{3,3} \end{pmatrix} - \boldsymbol{\tau} \tag{9}$$

where ϵ_{kk} is the dilatation and $\boldsymbol{\tau}$ is the stress that was reduced by viscous flow. The rate of decay of stress is a function of the current stress and time

$$\dot{\boldsymbol{\tau}} = \mathbf{m}(\boldsymbol{\sigma},t) \tag{10}$$

and the body force can be directly obtained with

$$\dot{\mathbf{f}} = -\nabla\cdot\mathbf{m} \tag{11}$$

The velocity is obtained from the body-force rates of eq. (11) using the Green's function method. Keeping track of the evolution of $\boldsymbol{\tau}$ and $\mathbf{u}$ allows us to model the time-dependent evolution of displacement and stress due to viscoelastic relaxation.

B. Contemporary Literature Review

The use of high-performance computing in earthquake sciences is not new [8], [9], [10]. GPUs are particularly promising [1], [11] due to the parallel nature of the geophysics problems and accuracy-driven choice of floating-point intensive arithmetic calculations.

In [1], the authors describe the GPU parallelization of AWP-ODC (Anealstic Wave Propagation-Olsen-Day-Cui) real-world earthquake simulator. Their study performs 3D finite-differences that are implemented as highly-parallel stencil computations on GPUs. This tool is closely related to Relax, but it performs calculations on hyperbolic equations while Relax uses parabolic and elliptic ones. Furthermore, Relax operates in the Fourier domain to perform the simulation arithmetic for (1) lowering errors/inaccuracies, and (2) FFT compute speed over of the slower finite-difference or finite-element approaches. The focus of their paper is Mint - a source-to-source translator for stencil computations - while we emphasize the parallelizability limits and GPU optimizations of a broader set of kernels. Their tool flow only considers a simulation volume of 192·192·192 in contrast with our volume of 512·512·256. We run our experiments on newer NVIDIA K20 GPU in contrast with the NVIDIA C2050 GPU used in their study. A similar study based on finite-differences was also reported in [12].

In [11], the authors accelerate a 3D Fourier-migration solver which extensively involves 3D FFTs much like our solver. However, their approach emphasizes the development of strategies for managing 3D FFT runtimes as it is their bottleneck computation. In our solver, FFTs and inverse FFTs are an important component but they do not dominate overall runtime (see Table I). Our Relax-Miracle solver handles many other types of parallel computations. Their smart use of compression (type change from float to integer) may only be applicable in scenarios with specific accuracy requirements.

TABLE I: Asymptotic and Implemented behavior of functions in Relax

Funtion	**Description**	**Complexity**[*]		**Runtime**
		Sequential	Parallel	Sequential (ms)
`eigenstress`	compute tensorial forcing term	$O(N)$	$O(N_{xy}^{2/3})$	6
`bodyforce`	evaluate the divergence operator	$K^1 \cdot O(N)$	$K \cdot O(N_{xy}^{2/3})$	22
`stress`	compute 6 independent components of stress tensor	$K \cdot O(N^2)$	$K \cdot O(N_{xy}^{2/3})$	31
Green's Function				
`FFT/iFFT`	compute fast Fourier Transform (or inverse)	$O(N \cdot \ln(N))$	$O(\ln(N))$	3.5
`elastic`	computes the particular solution	$O(N)$	$O(N_{yz}^{2/3})$	1.1
`surface`	summation in the 3-direction	$O(N)$	$O(\ln(N))$	1.9
`cerruti`	add homogeneous solution	$O(N)$	$O(N_{yz}^{2/3})$	6.1
			1 Relax Iteration $\rightarrow$	71.6 ms

^1K is the size of the finite impulse response filter used for derivatives in strain and divergence calculations.
$^2N = N_x \cdot N_y \cdot N_z$

TABLE II: Data-Structures

Data-Structure	**Description**	**Dimension**	**Size**	**Memory**[1]
u_1, u_2, u_3	3 components of cumulative displacement	3D	$3 \cdot N_x \cdot N_y \cdot N_z$	0.75 GB
v_1, v_2, v_3	3 components of instantaneous velocity or forcing term[2]	3D	$3 \cdot N_x \cdot N_y \cdot N_z$	0.75 GB
t_1, t_2, t_3	3 components of surface traction[2]	2D	$3 \cdot N_x \cdot N_y$	3 MB
sig	6 independent components of stress	3D	$6 \cdot N_x \cdot N_y \cdot N_z \ / \ 2$	0.75 GB
$moment$	6 independent components of instantaneous power density	3D	$6 \cdot N_x \cdot N_y \cdot N_z \ / \ 2$	0.75 GB
tau	6 independent components of cumulative stress	3D	$6 \cdot N_x \cdot N_y \cdot N_z \ / \ 2$	0.75 GB
$structure$	3 components of the viscosity structure	1D	$3 \cdot N_z \ / \ 2$	1.5 KB
			Total Memory $\rightarrow$	3.75 GB

[1]approximate calculations for $512 \cdot 512 \cdot 256$-sized problem
[2]calculation of displacement from equivalent body force or velocity from force per unit time is performed in place.

As such, our solver needs single-precision arithmetic to meet our accuracy goals.

In [13], the authors present a mechanism for handling large-scale 3D FFTs that exceed GPU main memory capacity using blocking and other performance-enhancing techniques. While not directly relevant for our problem sizes and spatial resolutions, their solutions presents an intriguing possibility for splitting the complete problem across multiple GPUs which we may consider in future work.

III. Analysis, Parallelization and GPU Optimization

We first attempt to understand the computational structures inherent in Relax to study its parallel potential, and suitability for GPU acceleration.

A. Understanding Computational Limits

In Table I, we list the key functions in Relax (similar to the ones shown in Figure 2. In Table II, we highlight the core data structures used throughout the Relax computations. From these results, we can make some preliminary observations and attempt to draw some (hasty) conclusions:

- Relax has an iterative computation where the bulk of time is spent in the `bodyforce` and `stress` functions (53 ms out of 71 ms).
  Conclusion 1: We only need to accelerate the most-frequently used portion of the code on the GPU while leaving the rest of the program running on the CPU host.
- For a problem size of 512·512·256, we need around 3.75 GB of actively used state. To a large extent, we can perform computations in place simplifying the address calculations and reducing the amount of memory bandwidth required.
  Conclusion 2: This can easily fit in the CPU main memory and we can stream this data over PCIe bus and/or offchip storage.
- Most of the computations are asymptotically linear in problem size ($O(N)$) as well as highly data parallel along each dimension ($O(\sqrt[3]{N^2})$).
  Conclusion 3: Writing and tuning parallel OpenMP and CUDA versions of such code should be relatively straight-forward.

However, these hasty conclusions are incorrect due to the **large-scale 3D organization** of the computed data. A deeper analysis of the code and first-cut programming effort reveals a nuanced picture of the computational trends and bottle-necks.

- When dealing with such physically derived 3D models, we frequently need to access data in multiple different ways (orthogonal dimension orders) making it challenging, if not impossible, to pre-fetch or pre-order data in one right way. This problem will be exacerbated if the data must be transferred over the long-latency, low-bandwidth PCIe bus instead of the superior local DRAM interface.

- Amdahl's law cautions us against selective parallelization of certain subsets of the computation as it limits overall application speedup. If we only accelerate `bodyforce` and `stress`, we are capping the peak theoretical speedup at ≈3.8 (from Table I). Additionally, we are ignoring the impact of this apparently sensible CPU-GPU partitioning on the cost of performing memory transfers over the PCIe bus. This strategy will require us to move a large chunk of the 3D dataset repeatedly back-and-forth between the host CPU and GPU in every iteration. From Table II we can estimate the cost of transferring 1.5 GB (cumulative input to function from Figure 3) over PCIe bus at ≈250ms assuming an ideal PCIe throughput of 5.8 GB/s (PCIe v3.0). Compare this to the sequential cost of the iteration ≈71ms (last row of Table I).
- GPUs present a completely different data-parallel substrate that is substantially different from multi-core CPUs. This allows us to rethink the nature of the underlying arithmetic in a manner that best exploits data-parallelism.

B. GPU Parallelization

The original Relax source is written in Fortran and enhanced to use OpenMP parallelization extensively. In our exploratory implementation, we only offloaded the Green's Function to the GPU and observed a slowdown instead of speedup. As discussed earlier in Section III-A, the size of data structures moved between host and device limits performance severely. In the same experiment, we noted the ability of PCIe 3.0-capable GPU cards with lower floating-point throughput (NVIDIA GT650M) to outperform a PCIe-2.0-capable GPU card that offered 2–3× higher floating-point processing capacity (NVIDIA K20). This curious result confirmed our suspicion that we have to reformulate the entire Relax computation to avoid (or minimize) memory transfers between the host CPU and the GPU over the PCIe bus. We subsequently made two significant changes to the GPU code:

- **Complete offload**: We knew we had to perform GPU parallelization to avoid data transfers entirely by constructing and storing the data structures exclusively in the GPU main memory itself. This was only possible if we converted most of Relax functions (barring file I/O and visualization) to CUDA kernels. We can convince ourselves from Table I that this is indeed possible due the data-parallel nature of the rest of the functions. As a software-engineering strategy, we re-wrote Relax functions incrementally (one-by-one) to target GPUs using CUDA while retaining rest of the program functionality intact. This allowed us to gradually move over the entire computation to the GPU while retaining confidence over accuracy and correctness of the underlying arithmetic.
- **Arithmetic Reformulation**: We also made algorithmic changes to enhance parallelism through arithmetic reformulation of the calculations. For the `cerruti` kernel, the original CPU implementation used FFT-based calculations. When considering the data-parallel GPU target, we were able to replace these with data-parallel analytical equations. This is not generally possible but in this specific instance, the arithmetic allows this change resulting in an ≈60% runtime improvement.

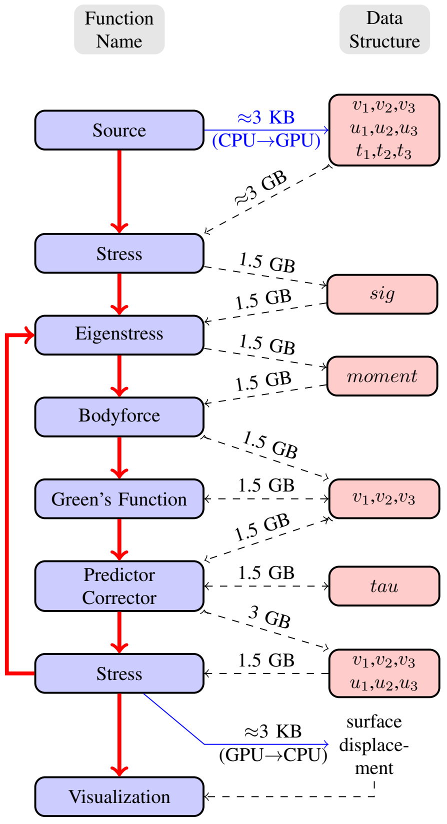

Fig. 3: Dataflow diagram for the Relax computation with data transfer size annotations for a $512 \cdot 512 \cdot 256$-sized problem

A flow diagram of the parallelized code annotated with data structure interactions is shown in Figure 3. As we can see, we have minimized the CPU↔GPU communication to a few kilobytes at the start and end of the computation. All internal multi-GB transfers stay local on the GPU.

IV. Experimental Setup

For our parallelization experiments, we consider both multi-core as well as GPU platforms. The key specifications of the platforms are listed in Table III. As we can clearly see, the motivation for considering GPUs is a combination of high floating-point throughput (≈10×) as well the substantially larger offchip memory bandwidth (≈4×). For the Monte-Carlo simulation, we parallelize the computation across a cluster of 12 NVIDIA K20 GPU cards. For software development and functional correctness, we focus on the cheaper GTX680

TABLE III: Computing Platforms

Platform	**Device**	**Technology**	**Die Size**	**Theoretical FLOPs**	**Off-chip Memory**		**Ratio**	
		(nm)	(mm^2)	(TFLOPs)	Capacity	Bandwidth	**FLOPs**	**Bandwidth**
Intel Multi-core	E5-2670	32	416	0.333	128GB	51.2 GB/s	1	1
NVIDIA GPU	GTX680	28	294	3.09	2 GB	192 GB/s	9.3	3.7
	C2075	40	520	1.03	4.68 GB	144 GB/s	3	2.8
	K20c	28	561	3.52	4.68 GB	208 GB/s	10.5	4

(and C2075) card while upgrading to the K20 for performance optimization.

We parallelize Relax for multi-cores using OpenMP for the data-parallel components and rely on 64-bit *Intel MKL 11.0* library (release 2013.5.192) and the 64-bit *FFTW3* (`libfftw3f-threads.so` compiled with *gcc-3.3.3*) library for the multi-threaded 3D FFT implementations on the CPU. For GPU programming we use *CUDA 5.0.35* toolkit along with the included *cuFFT* library. We also use the NVIDIA *Visual Profiler* for performance analysis and optimization. For the Markov-Chain Monte-Carlo simulations, we use *Matlab 2013.a* with a shell interface to Relax-Miracle. We use PAPI [14] to measure runtime on multi-core CPUs while using CUDA timers for GPU performance measurements. Our measurements are averaged across 100s of runs. We performed power measurements using the NVIDIA NVML library and recorded steady-state readings of 118–119W.

Accurate simulations of the deformation associated with earthquakes - or other sudden stress changes in the Earth - require a large 3D computational space. This is to offer simultaneously a fine sampling of the region considered, and to make this region as large as possible. We consider problem sizes 128·128·128, 256·256·256 and (preferred) 512·512·256 to consider varying degrees of accuracy. We are currently unable to model larger spaces due to offchip memory capacity limits of the NVIDIA K20 GPU but are aware of ideas [13] for solving this problem in the future. For visualizing the result of the Relax-Miracle simulations we use *Paraview 3.1* (used to create Figure 1).

V. Performance Results

We first describe the parallelizability of Relax computation using OpenMP on CPUs. Once we understand the limits of speedup possible using multi-threaded code, we discuss our performance results using GPUs. We will highlight the key causes of scaling issues and identify bottlenecks that prevent ideal performance through a series of questions and associated answers.

A. CPU Performance

Q: What is the impact of OpenMP multi-threading on Relax performance? Does the choice of MKL or FFTW3 affect scalability?

In Figure 4, we quantify the performance and scalability trends of parallel Relax computation running across multiple threads (1$\rightarrow$32) when using MKL and FFTW3 libraries. When

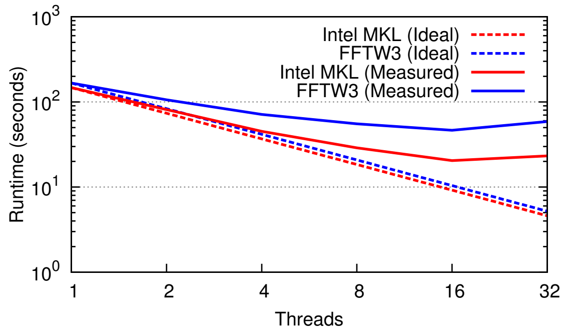

Fig. 4: Impact of parallelizing Relax using OpenMP across 1–32 threads when using MKL and FFT3 libraries

compared to ideal scaling behavior (T_{seq}/PE), we observe saturation in total Relax runtimes at $\approx$7$\times$ with 16 threads for the Intel MKL libraries. This is less than ideal but acceptable when compared the the FFT3 library. We encountered scalability challenges with the multi-threaded FFTW3 library, hence we focus on the optimized MKL library for our speedup calculations. Across both libraries, we observe a distinct slowdown when increasing total thread count from 16 to 32 . We attribute this systematic performance loss to the 2-chip Intel E5-2670 solution that can only support 16 hyper-threads per chip (but 32 total).

From Table I, we expect different functions in Relax to parallelize to different degrees on the multi-core platform. Certain data-parallel functions with high arithmetic intensity (flops/words ratio), should parallelize efficiently, while others should parallelize less well. In Figure 5, we observe the impact of scaling thread count on individual Relax functions. The bottleneck functions `bodyforce` and `stress` dominate parallel runtime across all threads. Observing the slope of the runtime scaling, we can conclude that the `stress` function starts to lose scalability beyond 2-4 threads due to limits of cache capacity and memory bandwidth. We see this behavior more clearly in Figure 6 where we separate out the individual speedups of the different functions.

Q: How does Relax performance change with varying simulation volume (problem size)?

In Figure 7, we show the impact of scaling problem size on single-thread and 16-thread performance on an Intel E5-2670 CPUs when using the Intel MKL library. As we can see the runtime scales close to linearly with problem size for the

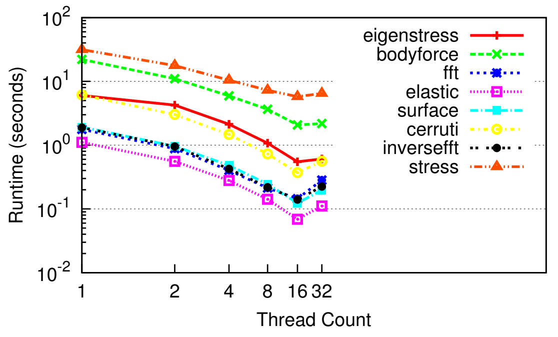

Fig. 5: Runtime Scaling of CPU Multi-Thread Performance (Intel MKL)

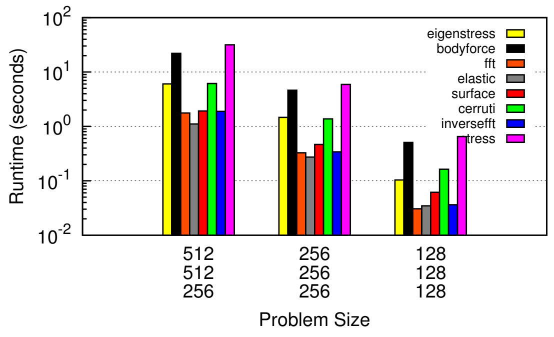

Fig. 8: Runtime breakdown of 1-Thread Performance (Intel MKL)

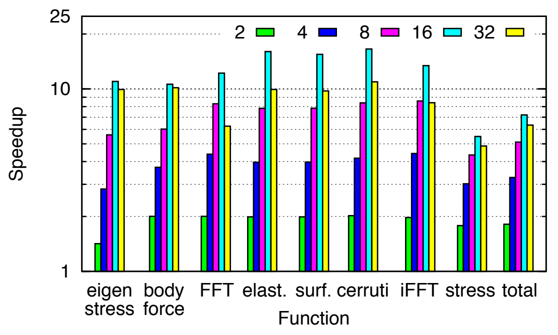

Fig. 6: Individual Speedups of different Relax functions as a function of thread count (Intel MKL)

single-thread solution but non-linearly when considering 16 threads. We should expect this due to the scaling trends of memory bandwidth for the 3D data structure accesses. A distinct runtime gap opens up above problem size of 256·256·128 due to low arithmetic utilization of small problems. As a consequence, we observe a speedup of 3.6× at problem size of 128·128·128 while it increases to 7.2× at the largest problem size we consider 512·512·256.

To better understand the impact of size on individual functions in Relax, we plot the breakdown of overall runtime in Figure 8 as a function of different problem sizes. We note that the `bodyforce` and `stress` function calls continue to take up the bulk of overall runtime at all sizes considered. The variation in runtimes between the fastest and slowest calls is as large as ≈28× at large problem sizes (512·512·256) while the gap is smaller ≈21× at smaller problem sizes (128·128·128) considering the asymptotics.

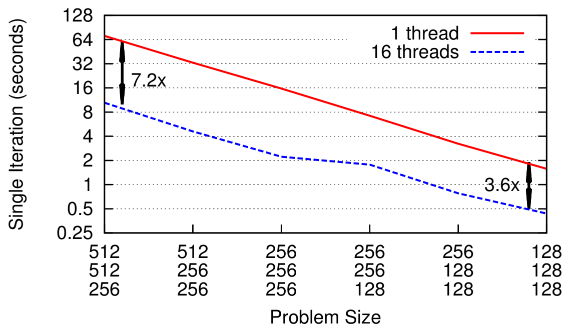

Fig. 7: Impact of Problem Size on 1-Thread and 16-Thread performance (Intel MKL)

B. GPU Performance

Q: When compared to the optimized 16-thread CPU implementation using Intel MKL library, how much speedup can the GPU offer?

In Figure 9, show the speedups for the CUDA GPU implementation on an NVIDIA K20 when compared to optimized Intel E5-2670 CPU implementations for the largest problem size of 512·512·256. If we consider single-thread MKL implementation, we achieve a speedup of 55.5× when using the GPU. This speedup becomes a more modest 7.7× when compared to the optimized 16-threaded implementation. The inherent parallelism in Relax is sufficiently high that multi-core CPUs are able to extract up to 7.2× speedup from the code, but the GPUs can go a step further to deliver a speedup of 7.7× over and above the optimized CPU mapping.

Q: How can we explain the nature of GPU speedups for Relax-Miracle? Why are we still able to achieve speedups higher than the multi-core CPUs?

We now show the relative improvements in performance due to GPU parallelization of Relax functions in Figure 10 (speedup) and Figure 11 (runtime).

- From the speedup plot, it is clear that bulk of the speedup is due to the high parallelizability of `stress` update computation on the GPUs. This is unsurprising as we have a large amount of data-parallelism that maps very well to GPUs. Unlike the multi-core platforms where the scalability of the `stress` update was limited, the CUDA implementation of this function scales particularly efficiently.
- The data-parallel `elastic` response and `surface` calculations also accelerate well on the GPU platform.

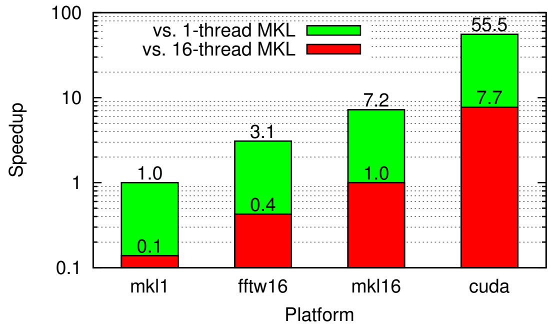

Fig. 9: Overall performance of various versions of Relax relative to the 1-thread and 16-thread OpenMP implementations with MKL FFTs for a simulation size of 512·512·256.

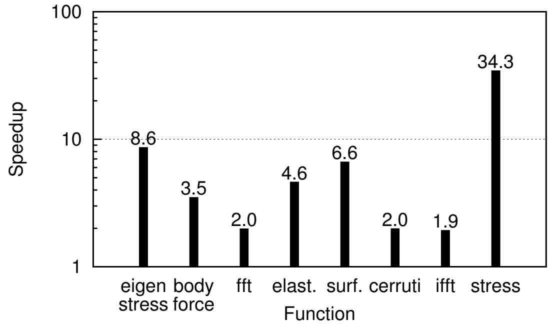

Fig. 10: Speedup breakdown of the CUDA implementation relative to the 16-thread OpenMP version of Relax with MKL FFTs.

- When inspecting runtime trends, the relative gaps between function performance stays unchanged except the `bodyforce` kernel, which now becomes the new critical bottleneck. These functions have a large degree of parallelism and scale in proportion to the relative extents of scalable parallelism.

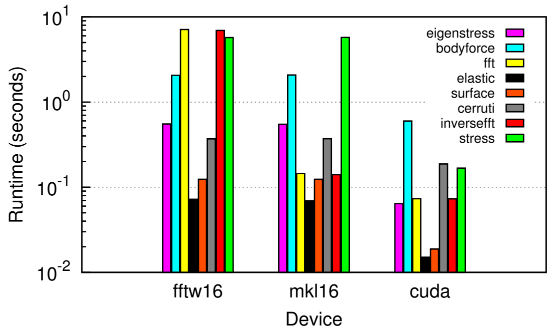

Fig. 11: Runtime breakdown of the OpenMP (with FFTW3 and MKL FFTs) and CUDA implementations.

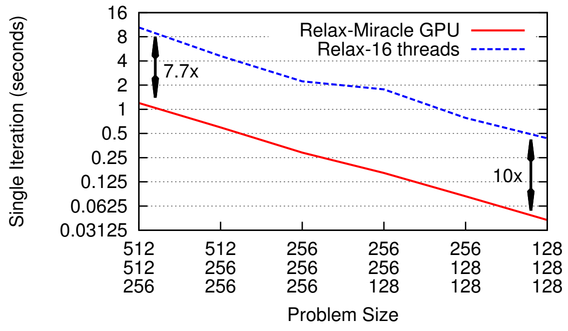

Fig. 12: Impact of Relax-Miracle input problem size on NVIDIA K20 GPU performance

Q: How robust are the observed GPU implementation speedups across different problem sizes?

As we saw previously in Figure 7, multi-core speedups for Relax shrink when considering smaller problem size due to low arithmetic utilization of the smaller datasets. In Figure 12, we now show the effect of increasing problem size on the performance of Relax (16-threads CPU) and compare it to Relax-Miracle (GPU). As we can see, high speedup 10× are possible for small problem sizes 128·128·128, which drops to 7.7× at the nominal problem size of 512·512·256. As observed earlier, we again see the distinct slowdown in 16-thread CPU performance when switching from 256·256·128 to 256·256·256. In comparison, the GPU performance scales smoothly with problem size suggesting fast and uniform memory access of the GPU memory subsystem.

Q: How does GPU performance change across GPU families?

We investigate if our results are repeatable across GPU platforms by running experiments on the GTX680, C2075 and the K20 platforms. In Figure 13, we observe the performance achieved (with individual function breakdown) across these diverse systems. As expected, the K20 beats both the GTX680 and the C2075 due to its higher offchip memory bandwidth and superior peak floating-point throughput. One curious observation is the slower runtime achieved by the `bodyforce` function on the K20 compared to the other devices. We are currently investigating this anomaly.

Q: Can we identify the source of GPU performance limits? Are there clear sources of performance bottlenecks?

In Figure 14, we show the result of profiling the execution of Relax-Miracle on the NVIDIA K20 GPU with the NVIDIA Visual Profiler. The top three functions by runtime take up a disproportionate 90% of overall GPU runtime. Unlike the multi-core implementation, `stress` drops to the third slowest function in the set. We observe that through our auto-tuning effort we are able to raise occupancy for the most critical function `bodyforce` to 0.7 but we are unable to further improve performance due to DRAM memory bandwidth limits. The other two functions `cerruti` and `stress` have low occupancies of ≈0.5 due to high registers/thread as well as

Function	Time (ms)	Occupancy	Warp Eff.	Instr. (M)	Reg	DRAM Write	DRAM Read	Load Eff.	Store Eff.
bodyforce	549	0.72	100	4,508	37	0	50	22	84
cerruti	238	0.46	98.85	2,634	60	53	38	400	0
stress	173	0.49	100	2,596	46	15	85	97	16
eigenstress	60	0.24	100	605	78	51	23	28	20
surface	21	0.5	87.21	434	55	59	96	45	0
elastic	17	0.5	87.37	445	54	63	64	45	90
FFT	4	0.49	101.79	40	41	72	72	0	100
iFFT	4	0.49	101.82	40	41	76	76	0	100

Fig. 14: Performance Analysis of CUDA Kernels
*DRAM Read/Write performance is in GB/s, Efficiency is in percentage

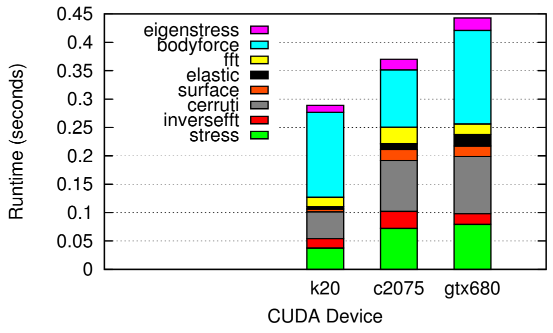

Fig. 13: Relax-Miracle Performance across different NVIDIA GPUs

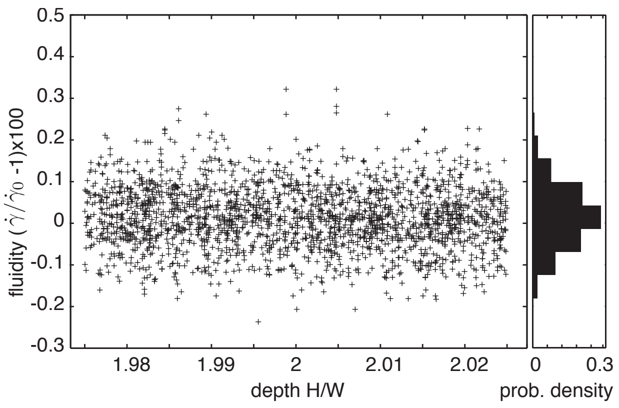

Fig. 15: Monte-Carlo sampling of the probability density function of Relax-Miracle models. Every cross corresponds to a time-dependent Relax calculation on a 256·256·256 grid. Viscosity of the visco-elastic substrate is constrained within 0.2%. Depth of the visco-elastic substrate is retrieved at the accuracy of numerical sampling.

DRAM bandwidth limits.

VI. Scientific Impact of Relax-Miracle

The performance obtained from the CUDA/GPGPU implementation enables applications of Markov-chain Monte-Carlo methods for Bayesian inversion of geophysical data or exploration of model uncertainties and trade-offs. To illustrate this capacity, we conduct a Metropolis-Hastings random walk of 5000 forward models to estimate the uncertainties of the depth and the fluidity of a viscoelastic substrate using postseismic relaxation data.

We construct a set of synthetic data using a simulation of 2 years of viscoelastic relaxation (relaxation time $t_m = 1/\dot{\gamma}_0 = 1\,\text{yr}$) following a strike-slip earthquake of length and width W occurring in an elastic lid of thickness $H = 2W$ (viscoelastic flow occurs underneath). We use a computational grid of 256·256·256 and a spatial sampling of $\Delta = 0.05\,W$. We simulate the time-dependent deformation at 13 points at the surface of the grid corresponding to the hypothetical location of GPS stations. We add a normally distributed signal representing a white noise of 5%. All models are simulated in parallel across the 12 NVIDIA K20 GPUs to achieve linear speedup. Roughly, each GPU can replace around 5–6 Intel multi-core systems in our compute cluster at same performance (or deliver 5–6× speedup over one CPU). Each simulation requires about 20 time steps on average and takes about 40 seconds. The sampling of the probability density function of the model parameters indicates that the depth parameter is retrieved to a level of uncertainty corresponding to numerical accuracy. The fluidity of the ductile rocks can be estimated within 0.2%.

The results shown in Figure 15 indicate that realistic models of Earth's deformation could be explored using Markov-chain Monte-Carlo methods to optimize the fit to geodetic data sets. Performance of our new GPU implementation allows us to conduct this type of analysis routinely, which will permit more thorough investigations of the mechanical properties of the lithosphere following large earthquakes.

VII. Conclusions

The ultimate aim of the Relax package is to enable large-scale and realistic simulations of the surface deformation due to physical phenomena such as earthquakes, volcanoes or hydrological processes on Earth. GPU-based systems offer the unique opportunity to exploit the spatial parallelism inherent in this simulation with less cost (equipment, maintenance, programming) that the currently used OpenMP-based multi-core platforms.

We show how to parallelize semi-analytic Fourier-domain calculations in Relax using the NVIDIA K20 GPU by as much as a factor of 7.7$\times$ when compared to optimized 16-threaded Intel E-52670 multi-core CPU supported by Intel MKL libraries for simulation sizes of 512·512·256. We are able to achieve this speedup by completely offloading 3D data structure construction and update to remain entirely within the global memory of the GPU while also transforming some of the numerical algorithms to prefer data-parallel analytical formulations instead of FFT computations. Furthermore, this only became feasible when all the functions that updated the large 3D data structures in Relax were converted into CUDA kernels to enable GPU-only storage of the data structures. We are currently limited in simulation speed and size by the GPU DRAM memory capacity as well as memory-bandwidth for the kernels with low arithmetic intensity.

The new GPU implementation of the Relax algorithm opens the door to more thorough explorations of the mechanical parameters of the Earth using statistical methods.

References

[1] D. Unat, J. Zhou, Y. Cui, S. B. Baden, and X. Cai, "Accelerating a 3d finite-difference earthquake simulation with a c-to-cuda translator," *Computing in Science & Engineering*, vol. 14, no. 3, pp. 48–59, 2012.

[2] A. Cochard and R. Madariaga, "Dynamic faulting under rate-dependent friction," *Journal of Pure and Applied Geophysics*, vol. 142, no. 3-4, pp. 419–445, 1994.

[3] F. F. Pollitz, "Gravitational viscoelastic postseismic relaxation on a layered spherical Earth," *Journal of Geophysical Research*, vol. 102, no. B8, p. 17921, 1997.

[4] S. Barbot and Y. Fialko, "Fourier-domain Green's function for an elastic semi-infinite solid under gravity, with applications to earthquake and volcano deformation," *Geophysical Journal International*, vol. 182, no. 2, pp. 568–582, 2010.

[5] ——, "A unified continuum representation of postseismic relaxation mechanisms: semi-analytic models of afterslip, poroelastic rebound and viscoelastic flow," *Geophysical Journal International*, vol. 182, no. 3, pp. 1124–1140, 2010.

[6] S. Barbot, Y. Fialko, and Y. Bock, "Postseismic Deformation due to the Mw 6.0 2004 Parkfield Earthquake: Stress-Driven Creep on a Fault with Spatially Variable Rate-and-State Friction Parameters," *Journal of Geophysical Research*, vol. 114, no. B07405, 2009.

[7] B. Rousset, S. Barbot, and J. P. Avouac, "Postseismic deformation following the 1999 Chi-Chi earthquake, Taiwan: Implication for lower-crust rheology ," *Journal of Geophysical Research*, vol. 117, no. B12, pp. 1–16, 2012.

[8] Y. e. a. Cui, "Scalable Earthquake Simulation on Petascale Supercomputers," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–20.

[9] T. Furumura and L. Chen, "Parallel simulation of strong ground motions during recent and historical damaging earthquakes in Tokyo, Japan," *Journal of Parallel Computing*, vol. 31, no. 2, pp. 149–165, 2005.

[10] K.-L. Ma, A. Stompel, J. Bielak, O. Ghattas, and E. J. Kim, "Visualizing Very Large-Scale Earthquake Simulations," in *Proceedings of the ACM/IEEE conference on Supercomputing*, Nov. 2003.

[11] J.-H. Zhang, S.-Q. Wang, and Z.-X. Yao, "Accelerating 3D Fourier migration with graphics processing units," *GEOPHYSICS*, vol. 74, no. 6, pp. WCA129–WCA139, Nov. 2009.

[12] D. Michéa and D. Komatitsch, "Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards," *Geophysical Journal International*, vol. 182, no. 1, 2010.

[13] J. Wu and J. JaJa, "High Performance FFT Based Poisson Solver on a CPU-GPU Heterogeneous Platform," *International Parallel and Distributed Processing Symposium*, pp. 115–125, 2013.

[14] P. Mucci, S. Browne, C. Deane, and G. Ho, "PAPI: A Portable Interface to Hardware Performance Counters," in *Proceedings of Department of Defense HPCMP Users Group Conference*, Jun. 1999, pp. 1–8.