Limits of FPGA Acceleration of 3D Green's Function Computation for Geophysical Applications

Nachiket Kapre, Jayakrishnan Selva Kumar, Parjanya Gupta
School of Computer Engineering
Nanyang Technological University
Singapore, 639798
nachiket@ieee.org

Sagar Masuti, Sylvain Barbot
Earth Observatory of Singapore
Nanyang Technological University
Singapore, 639798
sbarbot@ntu.edu.sg

***Abstract*—**
FPGA-based accelerators can outperform multi-core, GPU and Xeon Phi based platforms by at as much as 2.8× for 3D Green's Function processing in geophysics while delivering superior energy efficiency. FPGAs can efficiently implement a complex mixture of compute patterns that include data-parallelism, reductions, dataflow and streaming computations using spatial parallelism to deliver these speedups and power benefits. Since 3D Green's Function is highly-parallel but communication bound, we optimize the FPGA implementation by considering loop restructuring and tiling optimizations to minimize and regularize off-chip accesses. Furthermore, we configure the FPGA to implement the key compute intensive kernels at double-precision as well as single-precision to exploit the uncertainty in measurements of earthquake monitoring sensors. For 512×512×512 problem size, the Xilinx SX475T (Maxeler MAX3) outperforms the fastest architecture by 1.1–1.4× (double-precision), 2.2–2.8× (single-precision) with 1.2× better energy efficiency.

I. Introduction

Physically-derived computational problems from geophysical simulations often handle large volumes of data and perform complex mathematical operations on these datasets. The underlying computations are an abundant source of parallelism which makes it possible to easily map these computations to large clusters of multi-core CPUs with relatively modest programming effort using MPI and OpenMP. However, the cost of 3D data access and communication between the different constituent blocks limits overall achievable performance. For large problems with a 512^3 mesh, a single simulation can take at least 2–3 hours on parallel 16-thread Xeon architectures (see an example simulation result in Fig. 1). For our calculations, inverse modeling of physical parameters using Simulated Annealing would require about 15 weeks of calculations on the 16-thread Xeon or occupy a 1K-core cluster for 2–3 weeks. Accelerators such as NVIDIA GPU, Intel Xeon Phi and FPGA-based systems offer the potential to improve performance by keeping data locally on the accelerator cards and operating on them in parallel. However, in presence of significant memory access (communication) requirements for large 3D structures, even highly-parallel problems may challenges these accelerators.

In this paper, we implement 3D Green's Function acceleration using various accelerators. This is a common kernel in many geophysical modeling computations that involve estimating the elastic deformation of the earth's crust around earthquake events and also for modeling hydrological features such as the impact of rainfall or dam construction on geological stability. Proper sampling of geological structure require small sampling. So to encompass the whole study area in our numerical simulation requires a large number of points. In our experience these requirements are achieved for meshes with 512^3 or more elements. To meet scientific requirements, these large 3D structures (512^3) impose significant constraints on memory capacity and communication within the system. The specific computational blocks exhibit a rich mixture of compute patterns such as data-parallelism, reduction that can be supported well on parallel hardware. The data collected for geological monitoring stations that is used in this analysis is typically delivered with a 5% uncertainty,

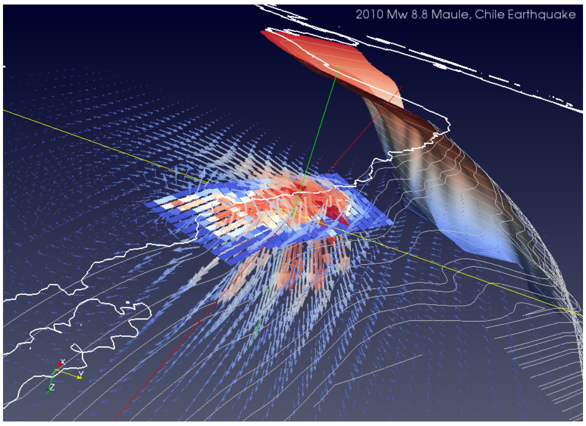


Fig. 1: Perspective of three-dimensional modeling of the 2010 Mw 8.8 Maule, Chile earthquake. The arrows represent the amplitude and direction of displacement occasioned by the quake. The central coloured plane represent the amplitude of slip of the subduction fault during the event. The megathrust, in the background, represents the plate boundary between the Nazca plate and the South American Plate. The simulation is on a 512^3 mesh and takes 2–3 hours on 16-thread Xeon system

which enable cheaper single-precision implementations. The specific challenges we explore in this paper include composing and co-ordinating applications as complex as Green's Function evaluation that easily exceed the logic capacity of a single FPGA card, managing the communication of data between the sub-functions and optimizing performance.

Maxeler DFE [5] (data flow engine) is an accelerator platform with an associated Java-based programming environment (MaxCompiler) that permits rapid development of FPGA accelerated computations. A single MAX3 Vectis FPGA card contains a Xilinx Virtex SX475T FPGA and 24 GB DRAM that is large enough to hold intermediate state for multiple instances of problems as large as 512^3. When programming this board, the computation is split into several *kernels* that are offloaded to the FPGA in a manner mostly similar to GPU parallelization. We consider single-FPGA designs where we explore loop restructuring, tiling and streaming optimizations that allow us to use fit the design into a single FPGA and optimize memory behavior.

The key contributions of this paper include:

- Implementations of 3D Green's Function kernels for elastic deformation for multi-core CPUs (OpenMP), GPU (CUDA), Xeon Phi (OpenMP pragmas) and FPGA (Maxeler).
- Evaluation of single-FPGA designs with optimizations for loop tiling, loop restructuring, dataflow streaming and precision optimization to support real-world problem sizes of scientific interest.
- Quantification of performance and power usage across various problem sizes and physical system configurations.

II. Background

A. Green's Function for Elastic Deformation

The deformation of elastic materials under infinitesimal strain is described by the Navier's partial differential equation, whereby the displacement field is related to internal body forces and surface tractions. For time-dependent deformation, an equation of the same form relates the velocity field to body forces and surface tractions per unit time. Many problems of geophysical interest including poroelasticity, viscoelasticity and faulting can be modeled in this framework. Solving the Navier's equation efficiently is therefore of great practical significance. Many numerical methods can be used to solve the partial differential equations in three dimensions, including the finite element or the finite difference methods. Solving the partial differential equation in the Fourier domain scales with N, therefore the efficiency of spectral methods grows with $N \log N$. A fast solution to the Fourier domain solution opens the door to many practical applications, including the modeling of time-dependent nonlinear visco-elasto-plastic deformation or aseismic creep on faults in three dimensions with hundreds of time steps. Another application of great significance is the Bayesian inversion of geophysical data using Monte Carlo methods. The Green's function is a mathematical operator that provides the displacement and stress due to a distribution of forces in the domain using the fast Fourier approach.

For the geophysical applications considered here, we are interested in solving for the elastic deformation due to internal body forces and surface traction in a half space. Mathematically, this implies that we need to incorporate the free surface boundary condition (the vertical components of stress are zero at the surface). In practice the boundary condition at the surface is of great importance because this is where most field measurements are usually carried out and the simulation results must be accurate there for effective comparison with data.

The spectral solver works as shown in Figure 2. First, we apply a fast Fourier transform of the body forces $\mathbf{f}(\mathbf{x})$ as shown:

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

For certain kinds of earthquake simulations we can skip the FFT step entirely with time-independent data in the context of multiple inverse Monte-Carlo simulations.

We then multiply the three components of the body forces by a matrix of wavenumber components. Here $\mathbf{M}(\mathbf{k})$ represents the transfer function between forces and displacements ($\hat{\mathbf{u}}$) in the Fourier domain.

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

The result is a displacement field that satisfies the conservation of momentum, but not the free-surface boundary condition.

Next, we compute the stress $\hat{\mathbf{t}}^p(k_1, k_2)$ at the surface. Typically, geophysical observations are carried out at the Earth's surface, so dealing with the free surface boundary condition is important to simulate data.

$$\hat{\mathbf{t}}^p(k_1, k_2) = \int_{-\infty}^{\infty} \mathbf{T}(\mathbf{k}) \cdot \hat{\mathbf{u}}^p\, dk_3 \tag{3}$$

and we add an correction that satisfies the partial differential equation but cancels out the surface stress. Here, $\mathbf{T}(\mathbf{k})$ represents the transfer function between displacements ($\hat{\mathbf{u}}$) and stress in the Fourier domain.

The displacement field that solves the elastic equations with given stress at the surface is referred to as Cerruti's solution. Conveniently, the analytic solution for displacement $\hat{\mathbf{u}}(\mathbf{k})$ can be formulated in the Fourier domain, so this step can be done in place.

$$\hat{\mathbf{u}}(\mathbf{k}) = \hat{\mathbf{u}}^p(\mathbf{k}) + \hat{\mathbf{u}}^h(\mathbf{k}; \hat{\mathbf{t}}^p) \tag{4}$$

Only after this correction do we apply the inverse Fourier transform to obtain a result in the space domain.

$$\mathbf{u}(\mathbf{x}) = \iiint_{-\infty}^{\infty} \hat{\mathbf{u}}(\mathbf{k})\, e^{i2\pi\mathbf{k}\cdot\mathbf{x}} d\mathbf{k} \tag{5}$$

The advantage of this technique is that the solution is obtained immediately with algebraic accuracy. No iterations are involved or convergence tests required. The solution is obtained with an arbitrary distribution of body forces and surface tractions.

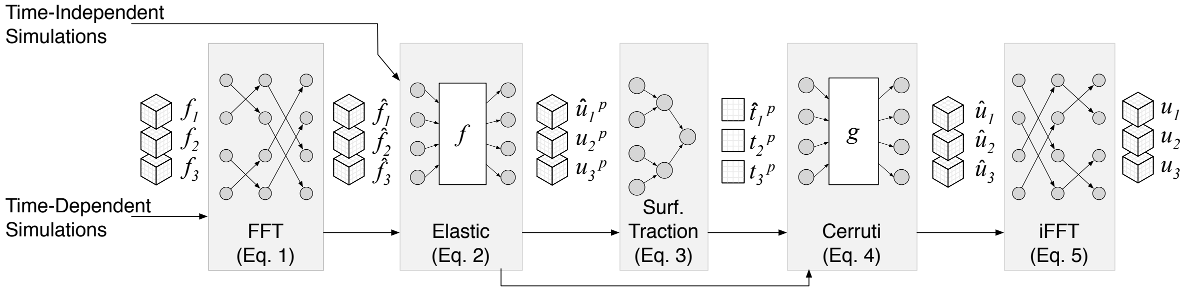


Fig. 2: 3D Green's Function Block Diagram
(Time Independent simulations do not need an explicit FFT, Equation numbers correspond to those in Section II)

TABLE I: Runtime breakdown of 1-thread performance on Xeon E5-2620 (512×512×512)

Function	**Time**	**Instructions**		**Memory Miss Rates**	
	(s)	FP Ops	Ld-Str	L1 Miss	L2 Miss
FFT	2.78 (13%)	11 M (0.15%)	2.1 G (11%)	20.7 (43%)	7.6 (44%)
Elastic	2.42 (11%)	1.4 G (20%)	2.7 G (15%)	1 (2%)	0.1 (0.5%)
Surf. Traction	3.61 (17%)	1.5 G (21%)	5.1 G (27%)	0.6 (1%)	0.1 (0.5%)
Cerruti	10.02 (46%)	4.5 G (59%)	6.2 G (33%)	3.3 7(%)	3.3 (18%)
iFFT	2.94 (13%)	12,M (0.15%)	2.4 G (13%)	22.2 (47%)	6.3 (36%)
Total	21.77 (100%)	7.5 G (100%)	18.5 G (100%)	47.8 (100%)	17.5 (100%)

B. Related Work

There is broad and vast existing literature of FPGA acceleration of scientific computing applications in geophysics. [2] shows the superiority of FPGA-based accelerators over GPUs and other competing conventional platforms for complex geophysics applications. The authors map a large, physically-inspired dataflow operations to the FPGA accelerator and deliver speedup and power benefits by exploiting spatial parallelism and high-capacity DRAMs to hold intermediate state. Recent work has also investigated geophysics acceleration on GPUs [7], [4] by extensive use of optimized CUDA libraries and auto-tuned CUDA routines. In our approach, we consider a considerably harder problem for FPGA mapping that presents a combination of challenges: (1) large size and orthogonal access patterns of the data being handled, (2) scale of the computation being implemented exceeding FPGA logic capacity, (3) consideration for I/O and memory bandwidth limited performance, and (4) fair evaluation of performance and power across multi-core CPU, GPU, Xeon Phi and FPGA accelerators.

C. Performance and Bottleneck Analysis

We first profile the Green's Function code on a single-thread implementation on an Intel Xeon CPU to identify performance bottlenecks. This code is written in Fortran and makes extensive use of FFTW and Intel MKL libraries. We also perform multi-core parallelizability analysis of the different blocks to understand how performance scales with threads on multi-core CPUs. For this experiment, we use the same Fortran code with OpenMP pragmas applied to suitable loops in the different stages of the computation. We tabulate the distribution of runtime and other statistics across the various Green's Function operators in Table I and the multi-thread scaling trends in Figure 3. We observe that the `Cerruti` kernel takes up around 50% of total sequential time with the rest equally distributed across the remaining functions. As expected, the FFTs and iFFTs are performance-limited by high L1 and L2 miss rates and very low floating-point operation efficiency. In contrast, the rest of the function have 20–60% floating-point occupancy which is favorable for parallelism. This bears out in Figure 3, where we see smooth linear improvements in runtime for most functions up to 16 threads with the exception of the parallel FFTs which start to saturate somewhat beyond 8 threads. Overall, the high floating-point occupancy and smooth parallel scalability of the bulk of the Green's Function code makes it ideal for parallelization. However, when solving large inverse problems that use the Green's Function kernel at 3D sizes such as 512^3, despite the abundant parallelism in the problem, overall performance can still be limited by I/O and memory bandwidth. The abundance of parallelism in presence of serious I/O concerns shifts the research focus in this acceleration study towards system-level strategies and composition of a larger and complex computation on the accelerator.

III. 3D Green's Function on Accelerators

We compare the key capabilities of the different accelerators evaluated in this study and discuss system-level issues and parallelization of the Green's Function on these accelerators.

A. Accelerators

In Table II, we identify the key datasheet specifications and metrics of the FPGA accelerators. As we see, when considering the peak floating-point capacity of the different accelerators, the GPU and the Xeon Phi are able to dominate the comparison with the CPU and the FPGA by almost 10×. In contrast, when comparing power usage of the platforms, FPGAs retain a 5× advantage over the GPU and Xeon Phi. Overall, the capability of a platform depends on other factors

TABLE II: Datasheet specifications and key metrics of the FPGA, GPU, Xeon Phi and Multi-Core systems

	Intel Multi-core CPU	Xilinx FPGA	NVIDIA GPU	Intel Xeon Phi
Processor	Xeon E5-2620	Xilinx Virtex-6 SX475T	K20c	Xeon Phi 5110P
Technology	32nm	40nm	28nm	22nm
Cores	6 x86 cores	N/A	2.5K CUDA cores	60 x86 cores
Clock Frequency	2 GHz	200 MHz	732 MHz	1.053 GHz
Peak FLOPs (64b)	96 GFLOPS	116 GLOPS[1]	1.17 TFLOPS	1.01 TFLOPs
Power[2]	95 W	40 W	225 W	225 W
On-Chip Memory				
Size L1 (per-core)	32 KB L1	38.3 MB (FMEM[3])	64 KB L1	32 KB L1
Size L2 (total)	1.5 MB L2		1.25 MB L2	30 MB L2
Size L3 (total)	15 MB L3			
B/W L1 (per-core)	256 GB/s L1	10 TB/s (FMEM[3])	180 GB/s L1[4]	80 GB/s L1[5]
B/W L2 (total)	190 GB/s L2		750 GB/s L2[4]	40 GB/s L2[5]
B/W L3 (total)	92 GB/s L3			
Off-Chip Memory				
Capacity	32 GB	24 GB (LMEM[3])	6 GB	8 GB
Interface	32b DDR3-1333	48b DDR3-400	324b GDDR5	32b GDDR5
Bandwidth	42.6 GB/s	38.4 GB/s	208 GB/s	320 GB/s

[1] From http://www.xilinx.com/support/documentation/white_papers/wp375_HPC_Using_FPGAs.pdf [2] We measure power as dataheet TDP numbers for the PCIe cards or the chip as advertised [3] FMEM refer to FPGA Block RAMs and LMEM is offchip DRAM in Maxeler terminology [4] From http://gpu.cs.uct.ac.za/Slides/Kepler.pdf (Slide 12) [5] From https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner (Fig. 14)

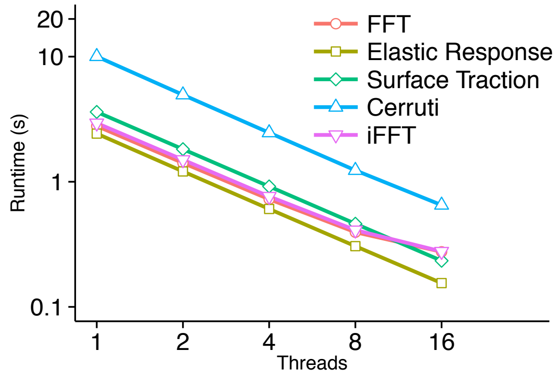


Fig. 3: Multi-thread performance scaling on 16-threads Intel Xeon E5-2620 with Intel MKL 2013.5.192 (512×512×512)

such as onchip memory capacity and memory bandwidth. On this front, the on-chip FPGA BlockRAMs deliver a staggering ≈10 TB/s bandwidth that exceeds the on-chip bandwidths of the competing platforms by 5–10×. The large 30 MB L2 cache of the Xeon Phi enables caching larger portions of the FPGA accelerator for fast local access. For larger problems that need to be stored in the off-chip DRAM, the GPU and Xeon Phi again offer as much as 8× higher bandwidth. Given this imbalance, for the parallel Green's Function computation, it may appear that the accelerators with faster memory interfaces will better support larger 3D data accesses. We investigate this assumption in following sections.

B. GPU Parallelization

First, we consider GPU-based parallelization of the Green's Function code. GPUs are highly-parallel SIMT (Single Instruction Multiple Thread) architectures that are organized into many SMs (symmetric multi-processors). Each SM runs a warp of threads which take turns executing code on the ALUs. For our application, we rewrote the individual Fortran routines of 3D Green's Function package [1] in CUDA to exploit the parallel potential of GPUs. This meant the code was split into separate kernels – one kernel per Green's function stage shown in Figure 2. A key system-level design strategy we chose was to keep the 3D structures resident in the GPU DRAMs between consecutive kernel invocations. This allows us to exploit the 208 GB/s (See Table II) bandwidth that is available. For the FFT and iFFT routines, we simply used the optimized `cuFFT` libraries supplied by NVIDIA. In the table below, we show the speedup achieved for each of the individual functions.

FFT	**Elastic**	**Surface Traction**	**Cerruti**	**iFFT**
14	72	82	20	15

In Figure 4, we show the speedup achieved for each of the individual functions. As expected, acceleration for memory intensive FFTs and iFFTs was limited to about 2× despite using optimized routines. However, other functions such as `Elastic Response` and `Surface Traction` were accelerated by greater factors due to available data-parallelism in the problem. The `Cerruti` kernel, despite available parallelism, saturated at a peak speedup of 2× as well due to excessive register usage per GPU thread.

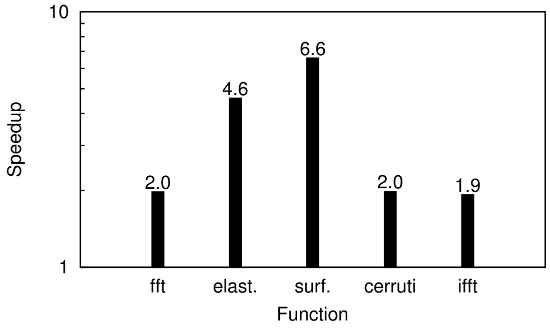


Fig. 4: Speedup Breakdown of NVIDIA K20 acceleration of 512×512×512 Green's Function vs. 16-thread Intel Xeon E5-2620

C. Xeon Phi Parallelization

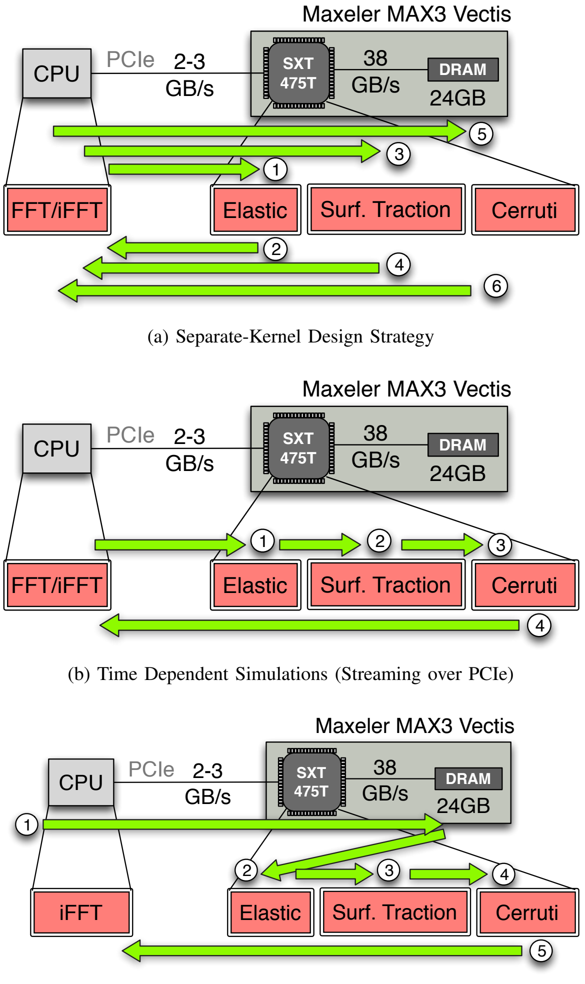
Next, we investigate the potential for Xeon Phi-based parallelization of the Green's Function code. The Xeon Phi accelerator is organized as multiple x86 cores interconnected with a ring supported by a large 30 MB L2 cache. Each core offers compatibility with existing x86 software but requires careful parallelization through suitable use of OpenMP pragmas or specific MKL functions. For parallelizing Green's Function Fortran code, we simply added suitable OpenMP pragmas for appropriate loops to encourage and guide parallelization on the Xeon Phi. To ensure the intermediate data resides on the accelerator memory space, we introduced suitable data persistence pragmas thereby avoiding a PCIe roundtrip. In the table below, we shown the speedup breakdown across the various functions.

FFT	Elastic	Surface Traction	Cerruti	iFFT
1	2.3	4.7	2.7	1.3

D. FPGA Parallelization (System-Level)

When parallelizing the Green's Function computation on the GPU and Xeon Phi, we split the design down in to individual kernels while retaining intermediate state on the accelerator DRAMs. For the FPGA dataflow design, we are (1) unable to fit the complete compute graph into a single FPGA configuration, and (2) unable to keep reliably data resident in the DRAM between multiple FPGA reconfigurations. We run the FFT and iFFT computations on the multi-core CPU as the resulting kernels are memory-bound and run fast enough on the host multi-core CPU. FPGA-based 3D FFTs [3] are possible but occupy a large portion of the FPGA by themselves (80% DSPs and RAMs on Virtex-7 XC7V200T) while operating on smaller sizes (64^3 single-precision vs. 512^3 double-precision required here) as constrained by the device characteristics. We represent our two system-level approaches in Figure 5.

- **Multiple-Bitstream FPGA Design with PCIe Streaming**: A naive implementation that mimics the GPU/Xeon Phi offload model adapted to the FPGA is shown in Figure 5a. Each kernel is configured on the FPGA in sequence thereby needing only a single physical card for the entire Green's Function computation. We optimize the kernels to occupy as much of the FPGA as possible by replicating the design multiple times until we exhaust capacity. Here, we copy back intermediate results to the host CPU as the Maxeler FPGA card is unable to retain DRAM contents across reconfiguration cycles. Since we expect this design to perform poorly due to the need to repeatedly transfer data back/forth to the host CPU, we do not explore this further but mention the strategy as a consideration for future FPGA board designs. Apart from keeping DRAMs in self-refresh state, we will have to be aware of DRAM controller training phase at bootup.
- **Time Dependent Simulations (PCIe Streaming – Figure 5b)**. In the time-dependent simulation case, we map all three functions `Elastic`, `Surface Traction` and `Cerruti` directly within a single FPGA. We optimize



(a) Separate-Kernel Design Strategy

(b) Time Dependent Simulations (Streaming over PCIe)

(c) Time Independent Simulations (Streaming over DRAM)

Fig. 5: Various System-Level Design Strategies for Composing the Accelerator

dataflow between these kernels using loop restructuring and tiling optimizations. We choose to stream the data directly over the PCIe interface from the host CPU, through the various compute blocks on the FPGA and back to the host CPU in a streamlined fashion. Since one transfer over PCIe is unavoidable for time-dependent simulation, we avoid the DRAM entirely and stream the data to the FPGA fabric.

- **Time Independent Simulations (DRAM Streaming – Figure 5c)**. For the time-independent scenario for earthquake simulations, we can skip the FFT step and load data directly from the DRAM for various simulations. In this case, we again map all three functions directly within a single FPGA but stream the inputs from the DRAM instead of the host CPU (PCIe). The input structure $\mathbf{f}(\mathbf{x})$ only needs to be copied to the DRAM once across multiple Monte-Carlo simulations. This allows us to exploit the fast bandwidth of the local DRAMs on the accelerator card. We still need to perform the iFFT step on the CPU which requires that we transfer data back over the PCIe interface overlapped with the FPGA compute.

E. FPGA Optimization (Kernel-Level)

The key performance challenge with 3D loops nested over x, y and z dimensions, is the poor memory performance due to the orthogonal nature of memory accesses in different iterations. While all platforms are ultimately limited by this memory access, there are still opportunities to minimize the number of accesses on suitable architectures. On the FPGA, it is possible to run each of three core kernels separately while exchanging data over the DRAM like the GPU, Xeon Phi and CPU implementations, but instead we can stream the inter-kernel code directly within the FPGA using dataflow composition. A similar optimization has poor effectiveness on the GPU, CPU and Xeon Phi due to limited on-chip register and cache capacity.

As shown in Fig. 6, we describe the fused kernel in MaxJava using special Java primitives that are synthesizable directly to the Maxeler FPGA card without needing to write low-level RTL code. We compose the three kernels using a Maxeler `Manager` that allows flexible integration with PCIe or DRAM-based modes. We express loop counter chains for the 3D iteration to feed the memory address generation logic. The dataflow expressions of the arithmetic capture spatial parallelism within the numerical evaluations. We reduce the number of off-chip memory accesses by keeping the data within the chip using streaming communication. We provide a high-level dataflow sketch of the various kernels in Listing 7.

We optimize the Maxeler FPGA implementation using two loop transformations:

- *Loop Restructuring*: The optimized, fused code is obtained through manual loop restructuring optimization that ensures the three sequential kernels can be combined in this manner. While various orders of x, y, and z indices are possible, the nature of the computation yielded y-x-z as the performance-optimal order. Here, we are required to wait one iteration of the z loop before executing `Cerruti` to balance buffering and ensure correct evaluation.
- *Loop Tiling*: Due to the 3D→2D reduction operation in `Surface Traction` phase, we use loop tiling optimization to ensure efficient on-chip memory access for the accumulation. Loop tiling splits the innermost loop over z into a nested loop and re-organizes memory accesses to use on-chip buffers with a reuse distance of C (multiple of 25 cycles). This increases FMEM usage to fit these buffers to significantly improves performance by improving memory access time and keeps the datapath occupied.

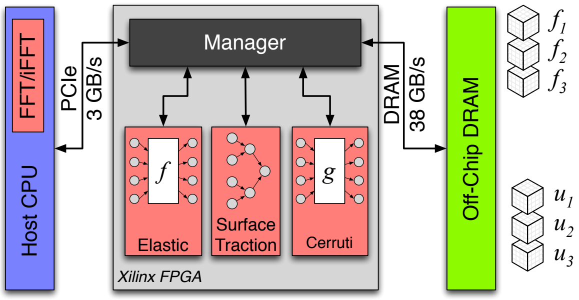


Fig. 6: Maxeler FPGA Design Composition

```
SUBROUTINE fused(REAL f1, f2, f3)
  DO y = 0, N
    DO x = 0, N
      DO z = 0, N
        ! do elasic here...
        ! do surface here...
      END DO
      DO z = 0, N
        ! do cerruti here...
      END DO
    END DO
  END DO
```

Fig. 7: Fortran pseudocode for three compute stages in 3D Green's Function calculation

IV. Methodology

We use Relax open-source software package [1], written in Fortran, for our experiments. Relax is already multi-threaded using OpenMP pragmas for use on multi-core CPUs. The CUDA version (Section III-B) is performance tuned with block-size and threads-per-block exploration, explicit register allocation control and memory transfer overlapping optimizations. The Xeon Phi version (Section III-C) is performance optimized with pragmas for vectorization, multi-threading, page buffering and offload preloading options. We tabulate the various environment configurations for the different platforms in Table III. The FPGA MaxJava implementation is optimized to run at 120 MHz through loop restructuring, tiling and pipelining optimizations. We tabulate FPGA resource utilization for the various design configurations (precision, interface) in Table IV. We use PAPI counters, CUDA timers events to record the time taken by the accelerated computations averaged across thousands of iterations. We calculate all speedups when compared against `ifort` optimized CPU implementations supported by tuned Intel MKL libraries. We also optimize GPU and Xeon Phi implementations through a design space exploration of various tuning parameters (*e.g.* block and thread configurations, offload pragmas).

V. Results

We discuss the overall performance and power utilization results for the different accelerators and simulation scenarios.

In Figure 8, we show overall speedups for the various accelerators including PCIe transfer times and find that neither the GPU nor the Xeon Phi outperforms multi-core CPUs for time-dependent simulations, with the GPU 10% faster for the time-independent simulations. The FPGA design operates close to the streaming throughput of the PCIe link bandwidth offering speedups between 10–40% over the 16-threaded CPU implementation under both scenarios respectively. The GPU implementation suffers for poor thread occupancy (less than 50%) due to high number of registers used per thread. Poor performance of the Xeon Phi is a known issue for non-regular workloads like 3D FFTs and 3D-problems [6]. For time-independent simulations, the first transfer is amortized over multiple Monte-Carlo instances resulting in slightly better results. The FPGA is now able to exploit the faster DRAM bandwidth to improve performance when compared to the PCIe-limited design. Overall, despite abundant parallelism, when facing communication limits, only the FPGA accelerator is able to offer a competitive solution that exploits available link bandwidth.

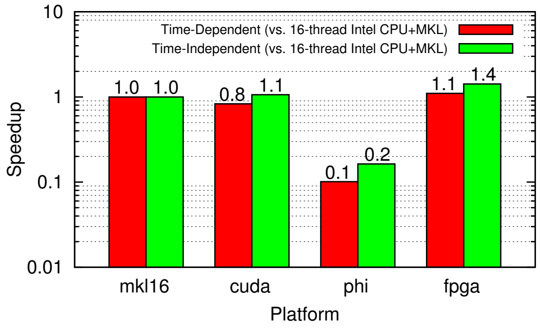


Fig. 8: Overall System-Level Speedups (Double Precision, 512×512×512 problem)

TABLE III: Platform Configurations across accelerators

Platform	OS	Compiler+Libraries
Xilinx FPGA	CentOS 6.5	MaxCompiler v2014.1.1 Xilinx ISE 13.3
NVIDIA GPU	Ubuntu 12.04	CUDA v6, cuFFT
Xeon Phi	CentOS 6.0	`ifort` v13.1.3 `composer_xe` v2013.5.192

TABLE IV: FPGA Resource Utilization Breakdown

Function	LUTs	FFs	DSPs	RAMs
Elastic	55K (19%)	80K (14%)	395 (20%)	0 (0%)
Surface	46K (16%)	69K (12%)	350 (18%)	6 (1%)
Cerruti	107K (37%)	157K (27%)	0.9K (50%)	342 (33%)
Total	213K (72%)	311K (53%)	1.7K (87%)	389 (37%)
Single-Precision, PCIe				
Total	81K (28%)	106K (18%)	339 (17%)	191 (18%)
Single-Precision, DRAM				
Total	121K (40%)	164K (27%)	334 (17%)	410 (38%)
Total (2-lane)[1]	186K (62%)	258K (43%)	556 (27%)	737 (69%)

[1] Here 2-lane design refers to two unrolled copies of the fused kernel operating in parallel

We clearly separate out the impact of I/O limits on performance in Figure 9. Fusing multiple kernels into a single kernel operates no worse than any single kernel as it is able to run in a streaming fashion on data sourced from the DRAM (3× 32b per dimension × 120 MHz = 11.5 GB/s). This means all inter-kernel I/O gets captures within the FPGA fabric allowing fully pipelined operation. While this is still only ≈30% of the DRAM bandwidth, a double-precision design occupies ≈87% of FPGA DSP resources preventing logic replication.

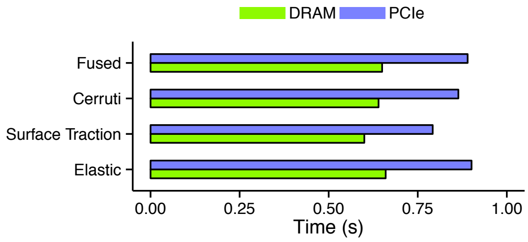


Fig. 9: Comparing Time-Dependent Simulation (PCIe) vs. Time-Independent Simulation (DRAM) Runtime on the FPGA accelerator

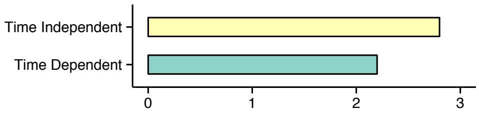


Fig. 10: FPGA Speedups (single-precision) vs. 16-thread Intel CPU+MKL (single-precision)

In comparison with the double-precision design, the single-precision design can be unrolled twice to have two concurrent datapaths that can operate on two sets of inputs streams from the DRAM. This results in a close to 2× improvements in runtime and an associated limited impact on accuracy of $1e^{-1}$ (l2-norm of $\mathbf{u}(\mathbf{x})$) which is within accepted tolerance for geophysics application requirements. We tabulate overall speedups in Figure 10. Both the GPU and Xeon Phi implementations are oddly marginally *slower* when using single-precision arithmetic due to misalignment in memory access (certain functions are faster, but slow overall).

For geophysical simulation scenarios, we consider various problem sizes from 128^3–512^3. As we can see, the GPU-based design is able to start closing the gap with CPU-based design and marginally beating it at the largest 512^3 problem size. The FPGA design always operates close to its I/O potential (whether using PCIe or DRAM links) and scales smoothly delivering the fastest design at all sizes. The 1024^3 time-dependent simulations (not shown), that enable more accurate scenarios to be processed, are only possible on the FPGA accelerator (24 GB DRAM) and CPUs.

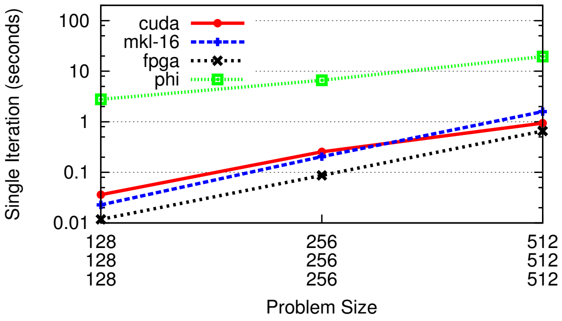


Fig. 11: Impact of Problem Size on Performance

In Table V, we show the power-performance tradeoffs across the different accelerators, and report the energy consumed by the different systems. As we can see, FPGA-based system delivers marginally better energy usage of 131 J over the 166 J multi-core CPU implementation. In contrast, the energy efficiency of GPU and Xeon Phi is 3× and 20× worse, respectively.

TABLE V: System Power-Performance Results

	CPU	FPGA	GPU	Xeon Phi
(tool)		`maxtop`	`nvidia-smi`	`micsmc`
Power (W)	128	143.8	214	263
Time (s)	1.3–1.6	0.92–1.4	1.7–1.9	8.5–15
Energy (J)	166–204	131–201	363–406	2235–3945
Ratio	1–1.2	1	2–2.7	17–19

A. Discussion

When inspecting peak capabilities of the different accelerators in Table II, the higher arithmetic throughput and faster DRAM interfaces on the GPU and the Xeon Phi may mislead a prospective developer into spending development and porting effort for those platforms. While they are easier to program (high-level CUDA or simple OpenMP pragmas), the architectures suffer I/O bottlenecks and other parallelization limits due to mismatch with the parallel application. FPGA-based designs are often harder, require careful design with a dataflow methodology, but can be configured to operate close to the I/O bandwidth capacity and logic potential of the fabric. The Green's function computation is fundamentally performance-limited by IO bandwidth. Under these circumstances, the accelerator with streamlined processing offers the best opportunity for acceleration. We present an FPGA design that occupies 80–90% of the FPGA while processing data in streamlined fashion limited only by the ability to access data. A board-level redesign of the FPGA accelerator card that enables retention of DRAM state offers one way to overcome the performance constraints imposed by PCIe IO bandwidth. While this is possible on the NVIDIA GPUs and Xeon Phi accelerators, the inter-kernel communication dependencies limit performance. The Maxeler FPGA accelerator even supports streamlined IO over 10G Ethernet that can be yet another opportunity to overcome PCIe IO limits. The Maxeler MPC C500 system can be configured to split the Green's function across the four FPGA cards and use MaxRing IO for streamlined operation. While this approach does not entirely eliminate PCIe streaming requirement by itself, we can use the extra logic capacity made available to program one of the cards to generate the 3D data structures entirely on the FPGA.

VI. Conclusions

We accelerate 3D Green's Function computation for geophysical applications using FPGAs by as much as 1.1–1.4× (double-precision) and 2.2–2.8× (single-precision) when compared to the best multi-core, GPU and Xeon Phi implementations for problem sizes as large as 512^3. We are able to deliver these speedups by exploiting spatial parallelism in the arithmetic expressions, loop restructuring to avoid off-chip data accesses and loop tiling optimizations. When limited by communication-bound problems, the FPGA dataflow methodology is capable of delivering balanced solutions that are better capable of using this limited resource more effectively than competing accelerators such as GPUs, and Xeon Phi.

References

[1] 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*, 182(2):568–582, 2010.

[2] C. He, M. Lu, and C. Sun. Accelerating seismic migration using FPGA-based coprocessor platform. *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, pages 207–216, 2004.

[3] B. Humphries and M. C. Herbordt. 3D FFT for FPGAs. In *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, pages 1–2, 2013.

[4] S. S. Masuti, S. Barbot, and N. Kapre. Relax-Miracle: GPU Parallelization of Semi-Analytic Fourier-Domain solvers for Earthquake Modeling. In *High Performance Computing (HiPC), 2014 21st International Conference on*, Dec 2014.

[5] O. Pell and O. Mencer. Surviving the end of frequency scaling with reconfigurable dataflow computing. *SIGARCH Computer Architecture News*, 39(4), Dec. 2011.

[6] F. Reid and I. Bethune. Evaluating CP2K on Exascale Hardware: Intel Xeon Phi. *PRACE whitepaper, PRACE-3IP*, 2014.

[7] 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*, 14(3):48–59, 2012.