Scaling 3D Elastic Applications with Multi-GPU Systems
What is your view of the world?

Goal: Discover oil/gas

- Limited time
- Limited resources
- Reduce area of study
What is your view of the world?

- Process 5X faster
- Perform more studies
- Same amount of time

Add GPUs
What is your view of the world?

- Add GPUs
  - Also scale linearly
  - Regional studies
  - Same amount of time
What is your view of the world?

- Reduce cycle time
- Improve earth model
- Understand reality

Add GPUs
Seismic 3D Elastic Forward Wave Modeling

- Fit-for-GPU
- Acceleration with directives
- Method for sizing multi-GPU systems
Seismic 3D Elastic Forward Wave Modeling

**Fit-for-GPU**

Acceleration with directives

Method for sizing multi-GPU systems
What are seismic modeling challenges?

Develop realistic earth models

- Improve success rate for finding oil/gas
- Reduce risk and save money
- Compare synthetic seismic data to field data
- Refine layer properties (velocity, density, ...)
- Optimize data acquisition strategies
- Maximize compute resources
Seismic imaging compute requirements

Source: exascale.org, TOTAL
Why 3D elastic modeling?

Earth is heterogeneous and anisotropic

- Important oil/gas reserves hidden under geologic discontinuities
- Complex geology scatters seismic waves
  - Salt formations, faults, dipping layers, density contrast
- Acoustic methods make simplifying assumptions
- Use compression and shear wave formulation
  - Accurately model wave dispersion at discontinuities
What are 3D elastic modeling challenges?

- Discrete methods used to solve 3D wave equation
- Artificial edges from numerical grids
- Handle boundary conditions to absorb waves
- Divide domain to fit system requirements
- Single seismic shot records span multiple systems
- Need to store more field properties than acoustic
Research Project

“Accelerating 3D Finite Difference (FD) wave propagation using GPUs” [1]

- Extend 3D FDTD prototype developed by NVIDIA [2]
- Convolutional Perfectly Matched Layer (CPML) boundary conditions
- Velocity-stress formulation
- Staggered grid, 4th order space stencil, isotropic medium
- Use GPU texture cache for absorbing layer
- Use MPI to overlap communication and computation
How do GPUs help accelerate imaging?

Natural domain decomposition

- Programming model grid of thread blocks
- Domain decomposition using 2D tiles
- Every grid point is managed by a lightweight thread
- Can benefit from shared memory and registers
- Can overlap communication with computation
Results

Multi-GPU cluster: Work/GPU constant

Multi-GPU cluster: Mesh size constant

Source: http://hal.inria.fr/docs/00/52/84/87/PDF/paper_David.pdf
Important 3D elastic modeling factors

- Computation patterns very similar to acoustic methods
- Elastic requires more property fields per grid point
  - Sub-domain dimensions per GPU may be smaller
- Stencil size
  - Smaller stencils require less memory accesses, however may not produce best image
- Grid implementation
  - Adjust mesh with depth
Seismic 3D Elastic Forward Wave Modeling

- **Fit-for-GPU**
- **Acceleration with directives**
- **Method for sizing multi-GPU systems**
Challenges

- **Productivity**
  - Focus more on Geoscience, less on computer science

- **Portability**
  - Unified approach that targets different accelerators

- **Safety**
  - Conservative approach that manages bookkeeping details

- **Performance**
  - Reuse common HPC patterns automatically
Solution: Accelerator directives

- Directives are added to source code
  - Manage loop parallelization
  - Manage data transfer between CPU and GPU memory

- Works with either C or FORTRAN
  - Can be combined with explicit CUDA C/FORTRAN usage

- Directives are formatted as comments
  - Do not interfere with serial execution

- Maintains portability of original code
Proof of Concept

- **Seismic CPML** [3]
  - Collection of 10 open source FORTRAN programs
  - Solve elastic wave equation
  - 3D FDTD, 4th order space, 2nd order time, isotropic medium

- **Portland Group (PGI) Accelerator Directives**
  - Mathew Colgrove, PGI [4]
Methodology

- Identify Critical Regions
- GPU Device Assignment
- Add Compute Regions
- Add Data Regions
- Optimize Data Movement
- Optimize Loop Schedules

Use The Portland Group Accelerator Tools
Identify critical performance regions

- Use PGI compiler as tool to identify opportunities
  - Complements profiler (PGPROF, or NVIDIA Visual Profiler)
  - Use built-in timing capability (compile option: -ta=nvidia,time)
  - Warn about data dependencies
  - Inform about data movement situations
Compute Region

!$acc region

do k = kmin,kmax
    do j = NPOINTS_PML+1, NY-NPOINTS_PML
        do i = NPOINTS_PML+1, NX-NPOINTS_PML

            total_energy_kinetic = total_energy_kinetic + 0.5d0 * rho*( vx(i,j,k)**2 + vy(i,j,k)**2 + vz(i,j,k)**2)

            epsilon_xx = ((lambda + 2.d0*mu) * sigmaxx(i,j,k) - lambda * sigmayy(i,j,k) - lambda*sigmazz(i,j,k)) / (4.d0 * mu * (lambda + mu))

            epsilon_yy = ((lambda + 2.d0*mu) * sigmayy(i,j,k) - lambda * sigmaxx(i,j,k) - lambda*sigmazz(i,j,k)) / (4.d0 * mu * (lambda + mu))

            epsilon_zz = ((lambda + 2.d0*mu) * sigmazz(i,j,k) - lambda * sigmaxx(i,j,k) - lambda*sigmayy(i,j,k)) / (4.d0 * mu * (lambda + mu))

            epsilon_xy = sigmaxy(i,j,k) / (2.d0 * mu)

            epsilon_xz = sigmaxz(i,j,k) / (2.d0 * mu)

            epsilon_yz = sigmayz(i,j,k) / (2.d0 * mu)

            total_energy_potential = total_energy_potential + 0.5d0 * (epsilon_xx * sigmaxx(i,j,k) + epsilon_yy * sigmayy(i,j,k) + epsilon_yy * sigmazz(i,j,k) + 2.d0 * epsilon_xy * sigmaxy(i,j,k) + 2.d0*epsilon_xz * sigmaxz(i,j,k)+2.d0*epsilon_yz * sigmayz(i,j,k))

        enddo
    enddo
enddo

!$acc end region
Verbose compiler output

% pgfortran -Mmpi=mpich2 -fast -ta=nvidia -Minfo=accel seismic_CPML_3D_isotropic_MPI_ACC_1.F90 -o gpu1.out

seismic_cpml_3d_iso_mpi_openmp:
1107, Generating copyin(vz(11:91,11:631,kmin:kmax))
Generating copyin(vy(11:91,11:631,kmin:kmax))
Generating copyin(vx(11:91,11:631,kmin:kmax))
Generating copyin(sigmaxx(11:91,11:631,kmin:kmax))
Generating copyin(sigmayy(11:91,11:631,kmin:kmax))
Generating copyin(sigmazz(11:91,11:631,kmin:kmax))
Generating copyin(sigmaxy(11:91,11:631,kmin:kmax))
Generating copyin(sigmaxz(11:91,11:631,kmin:kmax))
Generating copyin(sigmayz(11:91,11:631,kmin:kmax))
Generating compute capability 1.3 binary
Generating compute capability 2.0 binary

1108, !$acc do parallel, vector(4) ! blockidx%y threadidx%z
1109, !$acc do parallel, vector(4) ! blockidx%x threadidx%y
1110, !$acc do vector(16) ! threadidx%x
Using register for ‘sigmayz’
Using register for ‘sigmaxz’
Using register for ‘sigmaxy’
Using register for ‘sigmazz’
Using register for ‘sigmay’
Using register for ‘sigmax’

CC 1.3 : 38 registers; 2176 shared, 24 constant, 0 local memory bytes; 25% occupancy
CC 2.0 : 43 registers; 2056 shared, 140 constant, 0 local memory bytes; 33% occupancy

1116, Sum reduction generated for total_energy_kinetic
1134, Sum reduction generated for total_energy_potential
!$acc data region
!$acc     copyin(a_x_half,b_x_half,k_x_half,
!$acc             a_y_half,b_y_half,k_y_half,
!$acc             a_z_half,b_z_half,k_z_half,
!$acc             ix_rec,iy_rec,
!$acc             a_x,a_y,a_z,b_x,b_y,b_z,k_x,k_y,k_z),
!$acc     copyout(sisvx,sisvy),
!$acc     local(memory_dvx_dx,memory_dvy_dx,memory_dvz_dx,
!$acc             memory_dvx_dy,memory_dvy_dy,memory_dvz_dy,
!$acc             memory_dvx_dz,memory_dvy_dz,memory_dvz_dz,
!$acc             memory_dsigmaxx_dx, memory_dsigmaxy_dy,
!$acc             memory_dsigmaxz_dz, memory_dsigmaxy_dx,
!$acc             memory_dsigmaxz_dy, memory_dsigmayz_dy,
!$acc             memory_dsigmayy_dy, memory_dsigmayz_dz,
!$acc             memory_dsigmazz_dz,
!$acc             vx,vy,vz,vx1,vy1,vz1,vx2,vy2,vz2,
!$acc             sigmazz1,sigmax1,sigmay1,
!$acc             sigmazz2,sigmax2,sigmay2)
!$acc     copyin(sigmazz,sigmaxz,sigmazy,sigmayy,sigmayz,sigmazz)
What about data movement?

- If data is not contiguous, then compiler has to copy data in successive segments multiple times

- Not only interested in how much data is being copied
  - Also need to know how often you copy data

- Used conditional compile (preprocessor macro `_ACCEL`) to sub-divide 3D data into 2D tiles
# Results: single system scaling

<table>
<thead>
<tr>
<th>Programming Iteration</th>
<th>MPI Processes</th>
<th>OMP Threads</th>
<th>GPUs</th>
<th>Time (sec)</th>
<th>Approx. Programming Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>948</td>
<td></td>
</tr>
<tr>
<td>Naïve approach</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3599</td>
<td>10</td>
</tr>
<tr>
<td>Data regions</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>610</td>
<td>60</td>
</tr>
<tr>
<td>2D tiling</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>194</td>
<td>120</td>
</tr>
<tr>
<td>Scheduling</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>167</td>
<td>120</td>
</tr>
</tbody>
</table>

Just over 5 hours effort to obtain 5.5X speedup

Source: Portland Group
4 Core Intel Core-i7 920 2.67Ghz
2 Tesla C2070 GPUs
MPICH-2
Multi-GPU cluster scaling with directives

<table>
<thead>
<tr>
<th>Model Size</th>
<th>MPI Procs</th>
<th>OMP Threads</th>
<th>GPUs</th>
<th>Time (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_OMP 101 x 641 x 1536</td>
<td>24</td>
<td>96</td>
<td>0</td>
<td>1324</td>
<td>-</td>
</tr>
<tr>
<td>MPI_ACC 101 x 641 x 1536</td>
<td>24</td>
<td>0</td>
<td>24</td>
<td>353</td>
<td>3.8X</td>
</tr>
<tr>
<td>MPI_OMP 101 x 641 x 3072</td>
<td>24</td>
<td>96</td>
<td>0</td>
<td>2081</td>
<td>-</td>
</tr>
<tr>
<td>MPI_ACC 101 x 641 x 3072</td>
<td>24</td>
<td>0</td>
<td>24</td>
<td>508</td>
<td>4.1X</td>
</tr>
</tbody>
</table>

Source: Portland Group
HP SL390 G7 Starter Kit: 8 Node Cluster
Each node contains 2 socket Westmere @3.06 GHz (96 cores total)
MPICH over GbE
24 Tesla M2070 GPUs
Are Directives appropriate for your code?

- Need loops with large loop counts
  - Parallel nested loops with large loop indices
  - Best if loops are rectangular

- Locality is important to enable use of GPU
  - Take advantage of shared memory on GPU
  - Exploit data parallelism (accessing contiguous data at same time)

- Lag time to adopt new CUDA features (Peer-to-Peer)
  - Compilers must be conservative, don’t support host pinned memory
  - New async directive on the horizon
Seismic 3D Elastic Forward Wave Modeling

Fit-for-GPU

Acceleration with directives

Method for sizing multi-GPU systems
Challenges

- Develop system sizing information
  - Consider power and space constraints
  - Recommend CPU / GPU ratios

- Need application-driven approach
  - Must be meaningful for seismic industry
  - Optimized for CPU as well as GPU

- Measure power consumption of system components
  - Separate GPU power from CPU power
Possible solution

- Use 3D elastic forward wave modeling test
  - Supports MPI and OpenMP so can obtain CPU baseline
  - GPU accelerated version available
  - Measure single system, multi-GPU, cluster scalability

- Power monitoring
  - HP Proliant Power Interface Controller (PPIC)
  - Per outlet power monitors
  - NVIDIA Management Library (NVML) power usage
How is performance related to power?

3D Elastic Forward Wave Modeling

CPU
(2 Westmere Sockets)
- 598 Mpoints/s
- 577 Watts
- Perf/W: 1.04

GPU
(Server + 2 M2090s)
- 1,877 Mpoints/s
- 927 Watts
- Perf/W: 2.03

GPU delivers 1.95X more performance for same power budget

Power measured by: HP Proliant Power Interface Control (PPIC) tool
CPU: Dual Westmere x5670 2.93 GHz
Result: Perf/W of multi-GPU systems

Performance Delivered for Same Power Budget

- M1060 (2)
- Westmere (2)
- M2090 (2)
- M2090 (6)
- M2090 (8)

Perf/W Efficiency
Estimate for next-generation systems

Measured & Estimated Performance and Power Consumption

- M1060 (2)
- Westmere (2)
- SandyBridge (2)
- M2090 (2)
- M2090 (4)
- M2090 (8)
- Next GPU (4)
Summary

- GPUs are used to develop realistic earth models
  - Elastic methods are an extension of acoustic methods

- Tuning approach used with native CUDA apply to directives
  - Optimize data movement

- Use Perf/Watt for systems sizing recommendations
  - Application-driven approach
Conclusions

- Favorable scaling results for 3D elastic modeling codes
  - Using CUDA and PGI acceleration directives

- Accelerator directives + tools
  - Improve productivity, offer portability, and deliver performance
  - Should be added to programmers toolbox

- Multi-GPU systems offer Performance/Watt advantage
  - Solve datacenter issues constrained by power and space
References

- [1] “Accelerating a 3D finite-difference wave propagation code using GPU graphics cards”
  - David Michea and Dimitri Komatitsch, Geophysical Journal International
  - http://hal.inria.fr/docs/00/52/84/87/PDF/paper_David.pdf

- [2] “3D Finite Difference Computation on GPUs using CUDA”
  - Paulius Micikevicius, NVIDIA DevTech

- [3] Seismic Convolutional Perfectly Matched Layer
  - Dimitri Komatitsch and Roland Martin
  - http://www.geodynamics.org/cig/software/seismic_cpml

- [4] “5x in 5 hours, accelerating Seismic CPML using PGI Directives”
  - Mat Colgrove, PGI
Acknowledgements

- Mathew Colgrove, Portland Group, USA
- Thomas Toy, Portland Group, USA
Questions?