Industrial Seismic Imaging on a GPU Cluster

Scott Morton
Geophysical Technology
Hess Corporation
Seismic Imaging
Outline

• Industrial seismic data & imaging
  - Why use GPUs?

• Computational approach
  - Job decomposition & break-up
  - Hierarchical batch job submission & control
  - GPU programming

• Production codes
  - Performance
  - Multi-GPU issues & results

• Growth of a GPU cluster

• Summary of GPU usage
Seismic Imaging

Data

H\textsubscript{2}O

gas

oil
Seismic Imaging Data

Receiver

Time (ms)
Seismic Imaging Data
Seismic Imaging
An iterative process

1. Construct initial earth model
2. Perform imaging
3. Redundant images self-consistent?
   - Yes: Done
   - No: Update earth model

Computation scales as size of data and image/model
$t = T_S + T_R$

$\mathbf{I} (\mathbf{x}) = \sum_{\mathbf{s}, \mathbf{r}} D(\mathbf{s}, \mathbf{r}, t = T(\mathbf{s}, \mathbf{x}) + T(\mathbf{x}, \mathbf{r}))$
Seismic Imaging
Computational scale

- Imaging integral formulation:
  \[ O(\bar{x}) = \int \int d^2x_s \, d^2x_R \, A(\bar{x}, \bar{x}_s, \bar{x}_R) \, I(\bar{x}_s, \bar{x}_R, t = T(\bar{x}, \bar{x}_s) + T(\bar{x}, \bar{x}_R)) \]

- Computational complexity:
  - \( N_0 \sim 10^9 \) is the number of output image points
  - \( N_I \sim 10^8 \) is the number of input data traces
  - \( f \sim 30 \) is the number of cycles/point/trace
  - \( f \, N_0 \, N_I \sim 3 \times 10^{18} \) cycles \sim 30 \) cpu-years
Seismic Imaging

Theory

\[
\frac{1}{v(x)^2} \frac{\partial^2 P}{\partial t^2} = \nabla^2 P
\]

\[
I(x) = \sum_{\omega, \alpha} S^*_\alpha(x, \omega) R_\alpha(x, \omega)
\]

\[
\frac{\partial P}{\partial z} = \frac{\pm i \omega}{v(x)} \sqrt{1 + \frac{v^2(x)}{\omega^2} \left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right)} P
\]

\[
E = mc^2
\]

\[
\frac{\partial P}{\partial z} = \frac{\pm i \omega}{v(x)} \left( 1 + \sum_{l=1}^n \left[ \frac{\alpha_l S_x}{1 + \beta_l S_x} + \frac{\alpha_l S_y}{1 + \beta_l S_y} + \frac{\alpha_l S_{x+y}}{1 + \beta_l S_{x+y}} + \frac{\alpha_l S_{x-y}}{1 + \beta_l S_{x-y}} \right] \right) P
\]

\[
AP_{i-1,j}^k + (1 - 2A)P_{i,j}^k + AP_{i+1,j}^k = A^*P_{i-1,j}^k + \left( 1 - 2A^* \right)P_{i,j}^k + A^*P_{i+1,j}^k
\]
Seismic Imaging
Theory

\[ \frac{1}{v(x)^2} \frac{\partial^2 P}{\partial t^2} = \nabla^2 P \]

\[ I(x) = \sum_{\omega, \alpha} S_{\alpha}^*(x, \omega) R_{\alpha}(x, \omega) \]

\[ \frac{\partial P}{\partial z} = \frac{\pm i \omega}{v(x)} \sqrt{1 + \frac{v^2(x)}{\omega^2} \left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right)} P \]

\[ E = mc^2 \]

\[ \frac{\partial P}{\partial z} = \frac{\pm i \omega}{v(x)} \left( 1 + \sum_{l=1}^{n} \left[ \frac{\alpha_l S_x}{1 + \beta_l S_x} + \frac{\alpha_l S_y}{1 + \beta_l S_y} + \frac{\alpha_l S_{x+y}}{1 + \beta_l S_{x+y}} + \frac{\alpha_l S_{x-y}}{1 + \beta_l S_{x-y}} \right] \right) P \]

\[ A P_{i-1,j}^{k-1} + (1 - 2A) P_{i,j}^{k-1} + A P_{i+1,j}^{k-1} = \bar{A}^* P_{i-1,j}^k + (1 - 2\bar{A}^*) P_{i,j}^k + \bar{A}^* P_{i+1,j}^k \]
Seismic Imaging
Why GPUs?

- Price-to-performance ratio improvement
  - Want 10X to change platforms
    - Payback must more than cover effort & risk
    - Got 10X ten years ago in switching from supercomputers to PC clusters
Seismic Imaging
Why GPUs?

• Price-to-performance ratio improvement
  - Want 10X to change platforms
    • Payback must more than cover effort & risk
    • Got 10X ten years ago in switching from supercomputers to PC clusters
  - Several years ago there were indicators we can get 10X or more on GPUs
    • Peak performance
    • Benchmarks
    • Simple prototype kernels
Parallel algorithms

• Highly parallelizable
  - small kernel, inside large set of loops
  - computation dominates communication
    • $t_{\text{io}} \sim 1 \text{ day} \ll t_{\text{cpu}}$

• Example distributed algorithms
  - Distribute the output image across nodes
    • read data from tapes & pipe through all nodes
  - Replicated image volume on all nodes
    • read data from tapes & spread across nodes
    • at end-of-data, sum the image volumes
• Two levels of batch job handling
  - Top level is home-grown job handler
    • Submit full imaging jobs
      - Brazil, Gulf of Mexico, Indonesia, ...
    • Handles job steps
      - Distribute data
      - Pre-compute times of propagation
      - Break-up image region
      - Compute image pieces
      - Collect & sum image pieces
      - ...
  • Each step may contain many tasks
Two levels of batch job handling
- Use Sun Grid Engine (SGE) to handle arrays of tasks
  - Number of CPUs and GPUs per node are specified as SGE resources
  - Each task requires a specific set of resources
  - Linux OS handles CPU processes
  - NVIDIA driver handles GPU allocation
    - We use GPUs in “exclusive” mode, meaning only 1 process can use a given GPU
GPU Programming
Approach

- Design the computational algorithm just as you would with any parallel machine
  - Understand memory hierarchy
    • Size at each level
    • access characteristics
    • Bandwidth between levels
  - Stage data to computational engines efficiently
• Thread & block indices are analogs of loop indices
  - Use the CPU as a controller and data stager
    • Keep the CPU out of the loops as much as possible
    • Keep main data structures in GPU memory
    • Use as few GPUs per task as possible
    • Rarely efficient to loop over streaming data from CPU
  - Work for a thread is generally a single inner loop
GPU Programming
How to port a code?

• Create prototype GPU kernel
  - Include main computational characteristics
  - Test performance against CPU kernel
  - Iteratively refine prototype

• Port full kernels & compare with CPU results
  - Verify numerically
  - Compare performance

• Incorporate into production code & system
GPU Programming
Production codes

• 2 of our 3 production codes
  - Use 1 GPU per task
  - Have exactly 1 loop per thread

• All 3 production codes
  - Have at least 1 loop per thread
  - Use 1 CPU to control 1 GPU
  - Use only the GPUs on a single node

• For more algorithmic details, see my talk at
  - http://hpcoilgas.citris-uc.org/
Kirchhoff Imaging
Kernel optimization

0 – Initial Kernel
1 – Used Texture Memory
2 – Used Shared Memory
3 – Global Memory Coalescing
4 – Decreased Data Trace Shared Memory Use
5 – Optimized Use of Shared Memory
6 – Consolidated “if” Statements, Eliminated or Substituted Some Math Operations
7 – Removed an “if” and “for”
8 – Used Texture Memory for Data-Trace Fetch
Kirchhoff Imaging
Kernel performance

GPU-to-CPU Performance Ratio

Typical parameter range

<table>
<thead>
<tr>
<th>GPU Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
</tr>
</thead>
</table>

Image points per travel-time cell in x or y

GT200

G80

CUDA 2D Tex (nix=4)
CUDA Linear Tex (nix=4)
CUDA 2D Tex (niy=4)
CUDA Linear Tex (niy=4)
CUDA 2D Tex (nix=4) G2
CUDA Linear Tex (nix=4) G2
CUDA 2D Tex (niy=4) G2
CUDA Linear Tex (niy=4) G2
CUDA 2D Tex (nix=4) G2 PIN
CUDA Linear Tex (nix=4) G2 PIN
CUDA 2D Tex (niy=4) G2 PIN
CUDA Linear Tex (niy=4) G2 PIN
Kirchhoff Imaging
Production status

- GPU kernel incorporated into production code
  - Large kernel speed-ups results in task’s “CPU overhead” dominating GPU production runs

- Further optimizations
  - create GPU kernels for most “overhead” components
  - optimized left-over CPU code (which helps CPU version also)

<table>
<thead>
<tr>
<th>Time (hr)</th>
<th>Set-up</th>
<th>Kernel</th>
<th>Total</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original CPU code</td>
<td>5</td>
<td>20</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Main GPU kernel</td>
<td>5</td>
<td>0.5</td>
<td>5.5</td>
<td>5</td>
</tr>
<tr>
<td>Further optimizations</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>25</td>
</tr>
</tbody>
</table>
“Reverse-time” Imaging
Kernel performance

8th order, RTM

Grid size

Speedup

const, 16x16
shared, 16x16
const, 16x32
shared, 16x32

128x128x200 256x96x200 96x256x200 256x256x200 480x480x200 512x512x512 640x640x200 704x960x200 960x704x200 704x960x200 256x256x200 480x480x200 512x512x512 640x640x200 704x960x200 960x704x200 704x960x200
“Reverse-time” Imaging
Inter-GPU communication

• High frequency requires
  - Dense sampling
  - Large memory
  - Multiple GPUs
  - Halo exchange
  - Inter-GPU communication

• Device ↔ Host
  - Use pinned memory
  - PCIe bus predictably yields ~ 5 GB/s
  - ~ 10% of kernel time
  - Easily hidden
**“Reverse-time” Imaging**

**Inter-GPU communication**

- **CPU process ↔ process**
  - Currently using MPI
    - From legacy code
  - Performance variable
  - Comparable to kernel time
  - Solutions
    - OpenMP?
    - single controlling process?

- **Node ↔ node**
  - Currently Gigabit Ethernet
  - Moving to 10 Gigabit Ethernet
Multi-GPU Performance using MPI

![Graph showing performance over volume depth for 2-GPU, 4-GPU, and 8-GPU MPI configurations.](image-url)
Multi-GPU Performance
Using MPI+OpenMP

Performance, Gpoints/s

Z (total volume depth)
Hess GPU Cluster

• Got a single-card test system in spring ‘07
  - Used for prototyping Kirchhoff code
  - Benchmarked simple kernels
Hess GPU Cluster

• Got 32-node system in Dec ’07
  - Each node contains
    • Dual quad-core 1U server with 8 GB memory
    • Connected to 1U 4-GPU server with 1.5 GB each
  - Running test codes by end of Feb ‘08
  - Started production testing in May ‘08
  - 2 seismic imaging codes in production in summer ’08
Hess GPU Cluster

- 2nd generation system in Dec ’08
  - Upgraded to 4 GB, 2nd generation GPUs
  - Upgraded to 32 GBs on each CPU server
  - Added 82 more nodes to make 114 total
Hess GPU Cluster

• 2\textsuperscript{nd} generation system in Dec ’08
  - Upgraded to 4 GB, 2\textsuperscript{nd} generation GPUs
  - Upgraded to 32 GBs on each CPU server
  - Added 82 more nodes to make 114 total

• Upgraded Nov ’09
  - Retired 500 non-GPU nodes
  - Added 186 more GPU nodes
  - For a total of 1200 GPUs
    • Most of our compute power in GPUs

• Upgrading Nov ’10
  - Details being determined ....
Where on the S-Curve are we?

- "Wave-eqn" imaging
- Kirchhoff imaging
- "Reverse-time" imaging

GPU adoption vs. time
Acknowledgements

• Code co-authors/collaborators
  - Thomas Cullison (Hess & Colorado School of Mines)
  - Paulius Micikevicius (NVIDIA)
  - Igor Terentyev (Hess & Rice University)

• Hess GPU systems
  - Jeff Davis, Mac McCalla, Gary Whittle

• NVIDIA support & management
  - Ty Mckercher, Paul Holzhauer

• Hess management
  - Jacques Leveille, Vic Forsyth, Jim Sherman