OpenACC accelerated seismic imaging kernel: implementation and performance tuning

Shahzeb Siddiqui, Saber Feki

King Abdullah University of Science and Technology

SC13 – November 18th to 22nd 2013
Agenda

• Application: Seismic Imaging
• OpenACC Implementation of Seismic Imaging Kernel
• Performance Results
• OpenACC: Performance Tuning Approach
• Experiment: Profiling & Automatic Tuning
• Ongoing work
• Demo
Application: Seismic Imaging

- The boat has 10 cables with 556 receivers per cable, total number of receivers for 10 cables: 5,560
- Total number of shots: 82,557
- Total number of records ("traces") = shots x receivers = 82,557 x 5,560 = 459,016,920
- Each record length is 14.336 seconds with the 4.0ms sampling rate. Yield 3,584 samples per trace.
- Total data size = 459,016,920 x 3584 x 4 byte = ~ 6.6TB.
- Useful spectra are from 3.5Hz to about 45-50Hz, with depth up to 15km.
Computational Kernel

- Solve the acoustic wave equation (Isotropic case)

\[
\frac{1}{c^2} \frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}
\]

- Finite difference scheme, 2\textsuperscript{nd} order in time, 8\textsuperscript{th} order in space
Implementation of Seismic Imaging Kernel

```c
for (i=0; i<nt; i+=2) { // time loop
    for (x=4; x< p.ip-4; x++) {
        for (y=4; y< p.jp-4; y++) {
            for (z=4; z< p.kp-4; z++) {
                lap = coef[0]*V(x,y,z)
                    + coef[1]*(V(x+1,y,z) + V(x-1,y,z))
                    + coef[1]*(V(x,y+1,z) + V(x,y-1,z))
                    + coef[1]*(V(x,y,z+1) + V(x,y,z-1))
                    + coef[2]*(V(x+2,y,z) + V(x-2,y,z))
                    + coef[2]*(V(x,y+2,z) + V(x,y-2,z))
                    + coef[2]*(V(x,y,z+2) + V(x,y,z-2))
                    + coef[3]*(V(x+3,y,z) + V(x-3,y,z))
                    + coef[3]*(V(x,y+3,z) + V(x,y-3,z))
                    + coef[3]*(V(x,y,z+3) + V(x,y,z-3))
                    + coef[4]*(V(x+4,y,z) + V(x-4,y,z))
                    + coef[4]*(V(x,y+4,z) + V(x,y-4,z))
                    + coef[4]*(V(x,y,z+4) + V(x,y,z-4));

                U(x,y,z) = 2.*V(x,y,z) - U(x,y,z) + ROC2(x,y,z)*lap;

                if( (x==ixs) && (y==iys) && (z==izs) ){
                    U(ixs,iys,izs) = U(ixs,iys,izs) + source[i];
                }
            }
        }
    }
}
```
OpenACC Implementation of Seismic Imaging Kernel

```c
#pragma acc data copyin(U2[0:p.domain_size], U3[0:p.domain_size], source[0:nt], coef[0:five]) copy(U1[0:p.domain_size])
for(i=0; i<nt; i+=2) { // time loop
    #pragma acc parallel loop num_gangs(4) vector_length(32)
    for(x=4; x<p.ip-4; x++) {
        #pragma acc loop
        for(y=4; y<p.jp-4; y++) {
            #pragma acc loop
            for(z=4; z<p.kp-4; z++) {
                lap=coef[0]*V(x,y,z)
                +coef[1]*(V(x+1,y,z) + V(x-1,y,z))
                +coef[1]*(V(x  ,y+1,z) + V(x  ,y-1,z))
                +coef[1]*(V(x  ,y  ,z+1) + V(x  ,y  ,z-1))
                +coef[2]*(V(x+2,y,z) + V(x-2,y,z))
                +coef[2]*(V(x  ,y+2,z) + V(x  ,y-2,z))
                +coef[2]*(V(x  ,y  ,z+2) + V(x  ,y  ,z-2))
                +coef[3]*(V(x+3,y,z) + V(x-3,y,z))
                +coef[3]*(V(x  ,y+3,z) + V(x  ,y-3,z))
                +coef[3]*(V(x  ,y  ,z+3) + V(x  ,y  ,z-3))
                +coef[4]*(V(x+4,y,z) + V(x-4,y,z))
                +coef[4]*(V(x  ,y+4,z) + V(x  ,y-4,z))
                +coef[4]*(V(x  ,y  ,z+4) + V(x  ,y  ,z-4));

                U(x,y,z) = 2.0*V(x,y,z) - U(x,y,z) + ROC2(x,y,z)*lap;
                if((x==ixs) && (y==iys) && (z==izs)) {
                    U(ixs,iys,izs) = U(ixs,iys,izs) + source[i];
                }
            }
        }
    }
}
```
Performance Results: CPU – GPU Speed up

Speed Up gain CPU vs GPU and CPU vs GPU tuned

- CPU/GPU: 10.40, 27.99, 34.24, 24.28
- CPU/GPU tuned: 12.14, 28.74, 35.47, 30.24

Problem Size: 128x128x128, 256x256x256, 512x512x512, 640x640x640
Experiment: Profiling Kernel with nvprof

```
==17615== Profiling application: ./build/gpu_rtm_kernel_parallel --ip 64 --jp 64 --kp 64

Accelerator Kernel Timing data
/home/siddiqs/Research/version1/rtm_kernels/src/driver_parallel.c
main
  226: region entered 50 times
    time(us): total=14,874 init=5 region=14,869
    kernels=12,779
    w/o init: total=14,869 max=583 min=286 avg=297
  226: kernel launched 50 times
      grid: [904] block: [160]
    time(us): total=12,779 max=279 min=253 avg=255
/home/siddiqs/Research/version1/rtm_kernels/src/driver_parallel.c
main
  196: region entered 50 times
    time(us): total=309,092 init=6 region=309,086
    kernels=306,632
    w/o init: total=309,086 max=6,621 min=6,154 avg=6,181
  196: kernel launched 50 times
    time(us): total=306,632 max=6,187 min=6,116 avg=6,132

==17615== Profiling result:
Time(%)   Time      Calls   Avg     Min      Max    Name
 96.07%   305.49ms   50    6.1099ms  6.0979ms  6.1461ms main_196_gpu
 3.72%    11.819ms   50    236.38us   235.58us  238.21us main_226_gpu
 0.16%    517.73us   5     103.55us   2.7840us  170.62us [CUDA memcpy HtoD]
 0.05%    159.90us   1     159.90us   159.90us  159.90us [CUDA memcpyDtoH]
```
OpenACC: Performance Tuning Approach

- Wrap loops with directive `#pragma acc parallel` or `#pragma acc kernels` depending on whether you choose parallel or kernel construct
- Specify gang & vector parameters for kernel region, test and profile code until a good gang vector tuple is found
- Applications with varying problem size affect gang vector tuple therefore an automatic technique is needed to quickly find the best parameters
- NVIDIA recommends vector size should be multiple of warp size (32) on Kepler to acquire good performance thus we will use values [32,64,96,…,1024] for testing. 1024 is the maximum thread size in a gang on Kepler
- Gang values will be tested on increments of 2 starting from 2 up till 1024
Automatic Performance Tuning Model

Input code annotated with OpenACC

#pragma acc kernels
#pragma acc loop independent
fors (x = 4; x < nx-4; x++) {
    #pragma acc loop independent
does (y = 4; y < ny-4; y++) {
        #pragma acc loop independent
does (z = 4; k < nz-4; z++) {
            U[x][y][z] = c1*V[x][y][z] + ..... 
        }
    }
}

Accelerator Specification

Automatic code generator

#pragma acc kernels
#pragma acc loop independent
does (x = 4; x < nx-4; x++) {
    #pragma acc loop independent
does (y = 4; y < ny-4; y++) {
        #pragma acc loop independent
does (z = 4; k < nz-4; z++) {
            U[x][y][z] = c1*V[x][y][z] + ..... 
        }
    }
}

Runtime evaluation and selection

Database
Experiment: Automatic Brute Force Search

- Problem Size: 64x64x64
- Gang Combination: 512
- Vector Combination: 32
- Total Combinations: 512 * 32 = 16384
- Time recorded in seconds
- Avoids manual tuning, simply sort by time to find the best (gang,vector) tuple
### Performance Tuning Results

Kernel performance improvement using the tuning methodology on different domain sizes

![Graph showing performance improvement](image)

<table>
<thead>
<tr>
<th>3D Domain Size</th>
<th>Grid/block sizes chosen by compiler</th>
<th>Tuned grid/block sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>128x128x128</td>
<td>grid: [2x30] block: [64x4]</td>
<td>grid: [30x120] block: [64x4]</td>
</tr>
<tr>
<td>256x256x256</td>
<td>grid: [4x62] block: [64x4]</td>
<td>grid: [248x6] block: [64x6]</td>
</tr>
<tr>
<td>512x512x512</td>
<td>grid: [8x126] block: [64x4]</td>
<td>grid: [504x63] block: [32x8]</td>
</tr>
<tr>
<td>640x640x640</td>
<td>grid: [10x158] block: [64x4]</td>
<td>grid: [10x316] block: [64x4x2]</td>
</tr>
<tr>
<td>128x128x640</td>
<td>grid: [10x30] block: [64x4]</td>
<td>grid: [10x64] block: [64x4]</td>
</tr>
<tr>
<td>128x640x128</td>
<td>grid: [2x158] block: [64x4]</td>
<td>grid: [4x256] block: [64x4]</td>
</tr>
<tr>
<td>640x128x128</td>
<td>grid: [2x30] block: [64x4]</td>
<td>grid: [2x316] block: [64x4x2]</td>
</tr>
<tr>
<td>640x640x128</td>
<td>grid: [2x158] block: [64x4]</td>
<td>grid: [2x316] block: [64x4x2]</td>
</tr>
<tr>
<td>640x128x640</td>
<td>grid: [10x30] block: [64x4]</td>
<td>grid: [10x316] block: [64x4x2]</td>
</tr>
<tr>
<td>128x640x640</td>
<td>grid: [10x158] block: [64x4]</td>
<td>grid: [10x256] block: [128x4]</td>
</tr>
</tbody>
</table>
Ongoing Work

Manual Tuning
- Understand performance behavior of gang, vector of varying problem sizes using kernel and parallel construct
- Do exhaustive search to find a good gang vector parameter and compare performance gain compared to compiler
- Study compiler tuning settings for different problem sizes

Search Algorithm
- Utilize brute force technique to find the best gang vector tuple for parallel construct
- Extend experiment to other problem size to acquire more data
- **Future Work:** Exploring other advanced search methods such as genetic algorithm for a more efficient search

Loader
- Implemented a loader program to read experiment results for all problem sets and store data in an efficient data structure

Machine Learning
- **Future Work:** Use the experiment results in a machine learning algorithm for predicting gang vector parameters of different problem size and compare the efficiency and the performance gains:
  - Machine Learning vs. Brute Force
  - Machine Learning vs. Compiler Setting
Questions ?