Building ground models of Southern California

Steve Schlosser  
Michael Ryan  
Dave O’Hallaron  
Intel Research Pittsburgh

Julio López  
Ricardo Taborda  
Jacobo Bielak  
Carnegie Mellon University
Ground motion modeling 101

- Physical modeling
  - Mesh generation
  - Partition
  - Solve
  - Visualize
  - Analysis
Benefits of materialization

Query cost
- Materialized octrees are cheaper to query by 10-100x

Ease of use
- Eliminates days or weeks of dataset build time
- No need to integrate user-level code into SC apps
- Avoids quirks of existing model programs

Standardization
- Sharing among many scientists improves repeatability, validation
Observations
Boreholes, seismic events, etc.

Regions
LA basin, San Fernando, etc.

Interpolation

CVM Program: Community Velocity Model

Input: lat, long, depth tuples
Output: ground characteristics at that location
Two public implementations
SCEC (Fortran)
Harvard (C)

Basic building block
Goal:
Sample entire region at 10m resolution

\[ 6 \times 10^4 \times 3 \times 10^4 \times 1 \times 10^4 = 18 \times 10^{12} \text{ sample points!} \]

\[ \sim 1 \text{ PB of data uncompressed} \]

Approach:
Reduce early and reduce often

Image credit: Amit Chourasia, Visualization Services, SDSC
Map/Reduce implementation

Map: Sample entire region at target resolution

Image credit: Amit Chourasia, Visualization Services, SDSC
Map/Reduce implementation

Map: Sample entire region at target resolution

Reduce: Coalesce neighbors with similar characteristics

Image credit: Amit Chourasia, Visualization Services, SDSC
Map implementation

Map(String key, String value)
  // key: line #
  // value: x, y, z
  for each line:
    generate N samples
      starting at x, y, z
    emit <loc code, density>

Input tuples (line #, string)

Intermediate tuples (code, density)

Convert x, y, z coords to lat/lon/depth tuples for CVM input

Ground characteristic data: density, Vp, Vs

Convert x, y, z coords to intermediate locational codes for output

CVM

...
Intermediate key manipulation

Intermediate tuples (code, density)

Clear 3 low-order bits per octree level

Naturally gathers neighboring tuples together for Reduce

Manipulated tuples (code, density)
Reduce implementation

Intermediate tuples (code, density)

Reduce(String k, Iterator value)
// k: locational code
// value: sample data
vector samples = ();
foreach v in value
    samples.push(v);
if (tryCoalesce(samples))
    emit <coalesce(samples)>
else
    emit <samples>

Key = 000

Are they neighbors? Yes.

Are densities equal? No.

Run successive Reduces until data cannot be further coalesced

Output (code, density)
~1 day on our cluster

50 8-core blades
8GB memory
300GB disk
Several days on our cluster
“What if we did this in C?”

- Assign each region to a Maui/Torque task
- Generate samples in ascending loc code order
- Use stack-based coalescing algorithm
- Avoids all task/task communication

Harvard – a few hours
SCEC – ~1 day
Hadoop implementation

Map(String key, String value)
   // key: line #
   // value: x, y, z
   for each line:
       Fork CVM
           generate N samples
               starting at x, y, z
       Start stack coalescer thread
           if stack coalescer finished
               foreach (stack)
                   emit <loc code, density>

That’s it – no Reduce!
Conclusions

• Used Hadoop to build a ground model generator
• Hadoop implementation runs in O(days)
• Stack-based C and Hadoop versions run in several hours
• Cost of distributed group-by are not necessary for this app
• What is the lesson?

Map/Reduce
Map/Combine/Reduce
Map/Partition/Combine/Shuffle/Sort/Reduce
InputFormat/Map/ManipulateKey/Partition/Combine/Shuffle/Sort/Reduce/OutputFormat

Map/Reduce hides a lot of complexity