GRAMPS Beyond Rendering

Jeremy Sugerman
11 December 2009
PPL Retreat
The PPL Vision: GRAMPS

Applications
- Scientific Engineering
- Virtual Worlds
- Personal Robotics
- Data informatics

Domain Specific Languages
- Rendering
- Physics (Liszt)
- Scripting
- Probabilistic
- Machine Learning (OptiML)

DSL Infrastructure
- Domain Embedding Language (Scala)

Common Parallel Runtime (Delite, Sequoia)
- Domain specific optimization
- Task & data parallelism
- Locality aware scheduling

Hardware Architecture
- OOO Cores
- SIMD Cores
- Threaded Cores

Programmable Hierarchies
- Scalable Coherence
- Isolation & Atomicity
- Pervasive Monitoring
Introduction

• Past: GRAMPS for building renderers

• This Talk: GRAMPS in two new domains: map-reduce and rigid body physics

• Brief mention of other GRAMPS projects
GRAMPS Review (1)

• Programming model / API / run-time for heterogeneous many-core machines

• Applications are:
  – Graphs of multiple stages (cycles allowed)
  – Connected via queues

• Interesting workloads are irregular
GRAMPS Review (2)

- Shaders: data-parallel, plus push
- Threads/Fixed-function: stateful / tasks

Example Rasterization Pipeline
• Queue set:
  – single logical queue, independent subqueues
• Synchronization **and** parallel consumption
• Binning, screen-space subdivision, etc.
Map-Reduce

• Popular parallel idiom:

Map:
Foreach(input) {
  Do something
  Emit(key, &val)
}

Reduce:
Foreach(key) {
  Process values
  EmitFinalResult()
}

• Used at both cluster and multi-core scale
• Analytics, indexing, machine learning, …
Map-Reduce: Combine

- Reduce often has high overhead:
  - Buffering of intermediate pairs (storage, stall)
  - Load imbalance across keys
  - Serialization within a key
- In practice, Reduce is often associative and commutative (and simple).
- **Combine** phase enables *incremental, parallel* reduction
Preparing GRAMPS for Map-Reduce
Queue Sets, Instanced Threads

• Make queue sets more dynamic
  – Create subqueues on demand
  – Sparsely indexed ‘keyed’ subqueues
  – ANY_SUBQUEUE flag for Reserve

Make-Grid(obj):
For (cells in o.bbox) {
  key = linearize(cell)
  PushKey(out, key, &o)
}

Collide(subqueue):
For (each o1, o2 pair)
  if (o1 overlaps o2)
    ...

Fan-in Shaders

• Use shaders for parallel partial reductions
  – Input: One packet, Output: One element
  – Can operate in-place or as a filter
  – Run-time coalesces mostly empty packets

**Histogram(pixels):**

```plaintext
For (i < pixels.numEl) {
    c = .3r + .6g + .1b
    PushKey(out, c/256, 1)
}
```

**Sum(packet):**

```plaintext
For (i < packet.numEl) {
    sum += packet.v[i]
    Packet.v[0] = sum
    packet.numEl = 1
}
```
Fan-in + In-place is a **builtin**

- **Alternatives:**
  - Regular shader accumulating with atomics
  - GPGPU multi-pass shader reduction
  - Manually replicated thread stages
  - Fan-in with same queue as input and output

- **Reality:** Messy, micro-managed, slow
  - Run-time should *hide* complexity, not export it
• Three Apps (based on Phoenix):
  – Histogram, Linear Regression, PCA
• Run-time Provides:
  – API, GRAMPS bindings, elems per packet
Map-Reduce App Results

<table>
<thead>
<tr>
<th>footprint</th>
<th>Occupancy (CPU-Like)</th>
<th>Footprint (Avg.)</th>
<th>Footprint (Peak)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Histogram-512 (combine)</td>
<td>97.2%</td>
<td>2300 KB</td>
<td>4700 KB</td>
</tr>
<tr>
<td>LR-32768 (combine)</td>
<td>96.2%</td>
<td>10 KB</td>
<td>20 KB</td>
</tr>
<tr>
<td>PCA-128</td>
<td>65.5%</td>
<td>100 KB</td>
<td>205 KB</td>
</tr>
<tr>
<td></td>
<td>97.0%</td>
<td>1 KB</td>
<td>1.5 KB</td>
</tr>
<tr>
<td></td>
<td>99.2%</td>
<td>.5 KB</td>
<td>1 KB</td>
</tr>
</tbody>
</table>
Reduce vs Combine: Histogram
Two Pass: PCA (GPU-Like)
Sphere Physics
1. Split Spheres into chunks of N
2. Emit(cell, sphereNum) for each sphere
3. Emit(s1, s2) for each intersection in cell
4. For each sphere, resolve and update
256 Spheres (CPU-Like)
Other People’s Work

• Improved sim: model ILP and caches
• Micropolygon rasterization, fixed functions

• x86 many-core:
Thank You

• Questions?
Backup Slides
Optimizations for Map-Reduce

- Aggressive shader instancing
- Per-subqueue push coalescing
- Per-core scoreboard
GRAMPS Map-Reduce Apps

Based on Phoenix map-reduce apps:

- **Histogram**: Quantize input image into 256 buckets
- **Linear Regression**: For a set of \((x,y)\) pairs, compute average \(x\), \(x^2\), \(y\), \(y^2\), and \(xy\)
- **PCA**: For a matrix \(M\), compute the mean of each row and the covariance of all pairs of rows
Histogram 512x512 (GPU)
Linear Regression 32768
PCA 128x128 (CPU)
Sphere Physics

A (simplified) proxy for rigid body physics:
Generate N spheres, initial velocity
while(true) {
    • Find all pairs of intersecting spheres
    • Compute $\Delta v$ to resolve collision (conserve energy, momentum)
    • Compute updated result velocity and position
}


Future Work

• Tuning:
  – Push, combine coalesce efficiency
  – Map-Reduce chunk sizes for split, reduce

• Extensions to enable more shader usage in Sphere Physics?

• Flesh out how/where to apply application enhancements, optimizations