Volumetric processing and visualization on heterogeneous architecture

Wei Li
Siemens Corporation
Corporate Research & Technology
Princeton, NJ

In collaboration with Wei Hong & Gianluca Paladini
Volumetric computation

Areas
- Medical imaging, scientific visualization, simulation, movies, games, …

Volumetric computation
- Large amount of samples
- Rectilinear grids
- Similar computations on each element

Bricking
- Box-shaped
- Uniform-sized
Challenges

Bricked volumetric computation
- Order of bricks
- Workload distribution
- Brick memory management

Compiler
- Instruction scheduling
- Register allocation

Key ideas
- Virtual instruction with brick operands
- Compiler optimizations
Outline

- Instruction
- Related work
- Virtual instruction with brick operands
  - Overview
  - Instruction scheduling
  - Brick memory management
  - Implementation through OpenCL
- Example applications
  - Brick-based volume rendering
  - Flow simulation based on LBM
- Conclusion
Related work

- Register allocation
  - Graph coloring [Chaitin 82], Linear scan [Poletto & Sakar 99]

- Scheduling
  - Convergent scheduling [Lee et al. 02]
  - Heterogeneous Earliest-Finish-Time (HEFT) [Topcuoglu et al. 02]

- Combined
  - Parallel interference graph [Pinter 93]
  - Combined register allocation & instruction scheduling [Motwani 95]
  - Cooperative scheduling + linear scan [Win et al. 05]
  - Cache sensitive scheduling [Hardnet et al. 01]
Related work (cont’)

- Bricked-based volume rendering
  - E.g.: [Boada & Scopigno 01], [Magallon et al 01], [Bruckner 04], [Hadwiger et al 06], [Crassin et al 09]

- OpenCL & heterogeneous computing framework
  - OpenCL 1.1 spec. [Khronos group. 11]
  - StarPU [Augonnet et al. 09]
  - OpenCL with many GPUs [Barak et al. 10]
  - Maestro [Spafford et al. 10]
  - MAGMA [Agullo et al. 10]
Outline

- Instruction
- Related work
- **Virtual instruction with brick operands**
  - Overview
  - Instruction scheduling
  - Brick memory management
  - Implementation through OpenCL
- Example applications
  - Brick-based volume rendering
  - Flow simulation based on LBM
- Conclusion
Virtual instruction with brick operands

- Computation involving one or more bricks => virtual instruction
- A brick => an operand
- Instruction + dependencies = DAG
System overview
Virtual instructions: example
Strategies

- Cache as many bricks as possible
- Don’t delete brick until short of memory
- Minimize brick transfer among devices
  - Goal: each brick is transferred at most once during each iteration
- Cache & memory sensitive scheduling
Outline

- Instruction
- Related work
- Virtual instruction with brick operands
  - System overview
  - Instruction scheduling
  - Brick memory management
  - Implementation through OpenCL
- Case studies
  - Bricked-based volume rendering
  - Flow simulation based on LBM
- Conclusion & future work
Instruction Scheduling (cont’d)

- Priority function

\[
priority(n_i) = \alpha \times rank_u(n_i) + \beta \times cache(n_i)
\]

\[
rank_u(n_i) = \overline{w}_i + \max_{n_j \in \text{succ}(n_i)} \left( c_{i,j} + \text{rank}_u(n_j) \right)
\]

\[
cache(n_i) = C - \min_{d} \sum_{n_j \in \text{prepare}(n_i)} w_{j,d}
\]

\[
\text{prepare}(n_i) = \{ n_j | n_j \in \text{conditionals} \land n_j \in \text{produce}(op) \land op \notin \text{cached} \land (op \in \text{operands}(n_i) \lor op \in \text{operands}(\text{prepare}(n_i))) \}
\]
Instruction Scheduling (cont’d 2)
Outline

- Instruction
- Related work
- Virtual instruction with brick operands
  - Overview
  - Instruction scheduling
  - Brick memory management
  - Implementation through OpenCL
- Case studies
  - Bricked-based volume rendering
  - Flow simulation based on LBM
- Conclusion & future work
Spill buffer selection

- Linear scan
  - Spill the live interval that has the largest end point
Memory management instruction

- Instructions
  - setupBuffer
  - deleteBuffer
  - spillBuffer
  - refillBuffer
  - copyBuffer

- Operand access flags
  - read_only
  - Write_only
  - read_and_write
  - setup
  - delete
  - spill
  - refill
  - host_to_device
  - device_to_host
Time stamps

- Each operand has multiple instances
  - One for each device, One for the spilled buffer on each host system
- A time stamp for every instance
  - Time stamp set to -1 if an instance does not exist
- Time stamps are tracked and checked to avoid unnecessary buffer copying

Relation between operand access and time stamp action

<table>
<thead>
<tr>
<th>access</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_only</td>
<td>none</td>
</tr>
<tr>
<td>read_and_write</td>
<td>$t_d ++$</td>
</tr>
<tr>
<td>write_only</td>
<td>$t_d ++$</td>
</tr>
<tr>
<td>setup</td>
<td>$t_d \leftarrow -1$</td>
</tr>
<tr>
<td>delete</td>
<td>none</td>
</tr>
<tr>
<td>spill and device_to_host</td>
<td>$t_h \leftarrow t_d$</td>
</tr>
<tr>
<td>refill and host_to_device</td>
<td>$t_d \leftarrow t_h$</td>
</tr>
</tbody>
</table>
### Time stamps

- Each operand has multiple instances
  - One for each device, one for the spilled buffer on each host system
- A time stamp for every instance
  - Time stamp set to -1 if an instance does not exist
- Time stamps are tracked and checked to avoid unnecessary buffer copying

**Changing buffer manipulation instructions depending on time stamps**

<table>
<thead>
<tr>
<th>instruction</th>
<th>condition</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spillBuffer</code></td>
<td>$t_d = t_h$</td>
<td>replaced by <code>deleteBuffer</code></td>
</tr>
<tr>
<td><code>refillBuffer</code></td>
<td>$t_h = -1$</td>
<td>replaced by <code>setupBuffer</code></td>
</tr>
<tr>
<td><code>copyBuffer</code></td>
<td>$t_d = t_h$</td>
<td>skipped</td>
</tr>
</tbody>
</table>
Outline

- Instruction
- Related work
- Virtual instruction with brick operands
  - Overview
  - Instruction scheduling
  - Brick memory management
  - Implementation through OpenCL
- Case studies
  - Bricked-based volume rendering
  - Flow simulation based on LBM
- Conclusion & future work
Implementation through OpenCL

- device => OpenCL context
- queue => OpenCL command queue
- One queue for each independent processor on a device
  - Example: Fermi GPU, 1 queue for all computing cores, 1 queue for each data-transfer engine.
- A virtual instruction => one or more OpenCL kernels/tasks
Implementation through OpenCL (cont’)

- We enqueue OpenCL tasks non-blockingly
- Use event callback for expensive non-OpenCL computation
- Synchronize across queues through events
- Disable out-of-order execution
- No synchronization necessary for instructions on the same queue
Instructions on Different Queues

- Traditional compilers
  - Advance the load instructions
- Different queues for computing and transferring
  - A task starts as early as possible
  - Computing and transferring can overlap

- Virtual instructions
  - No need to move load instructions forward
  - Need to submit instructions in order
Instructions on Different Queues (cont’)

0: load density [0]
1: load density [1]
2: load density [2]
3: load density [3]
4: load density [4]
5: render [0]
6: render [1]
7: render [2]
8: render [3]
9: render [4]

0: load density [0]
1: render [0]
2: load density [1]
3: render [1]
4: load density [2]
5: render [2]
6: load density [3]
7: render [3]
8: load density [4]
9: render [4]
Instructions on Different Queues (cont’)

(a) loadBrick

(b) renderBrick

idle
Outline

- Instruction
- Related work
- Virtual instruction with brick operands
  - Overview
  - Instruction scheduling
  - Brick memory management
  - Implementation through OpenCL
- Demo applications
  - Bricked-based volume rendering
  - Flow simulation based on LBM
- Conclusion & future work
Bricked-based volume rendering: Stereo or multiple views
Bricked-based volume rendering: Stereo or multiple-views
Bricked-based volume rendering
Bricked-based volume rendering with shadowing & scattering

- illumination volume [Ropinski et al.10]
- Light propagation from a voxel to its neighbors
- Bricked volume
  - Propagation inside brick
  - Propagation inter brick
Allow light propagation instruction access neighbor illumination bricks

`propagateLight brick(0, 0, 0), brick(0, 1, 0), brick(1, 1, 0), brick(1, 0, 0), ...`

Need current brick and neighbor bricks live in memory
Propagation buffer

- Face vector
- Propagate out-of brick
- Propagate into brick
Propagation buffer (cont’)

(a) Instruction $i$

(b) Instruction $j$
Advantage of propagation buffer

`propagateLight` brick(0, 0, 0), propagation buffer

- Reduces size of working set
- 2D slices vs. 3D bricks
- Can reside on fast memory
Flow simulation based on LBM

- Lattice Boltzmann Method
  - Lattice-based flow simulation
  - Data stored on rectilinear grid
  - Computation performed for each grid point
  - Propagation to direct neighbors at each step
  - Replicate 1 slice on every face from neighbors
Boundary exchange of bricked LBM

- Exchange one slice at each step on every face of a brick
Boundary exchange of bricked LBM (cont')
Boundary exchange of bricked LBM (cont’)

- Exchange $n$ slices in exchange step after simulating $n$ steps
- Gain: $(n-1)O$
  - $O$: per-step overhead (such as spill/refill brick)
- Loss: $6(n-1)k$ extra elements
  - $k$: # of bricks
Integrated Flow Simulation & Visualization (con’t)

- Simulation on GPU
- Visualization on CPU
- copy velocity field using deviceToHost
Integrated Flow Simulation & Visualization
Other applications in progress or in plan

- Reconstruction
- Segmentation
- Registration
- More volume rendering
  - Multi-volume fusion, deformable volume, compressed volume, dynamically generated volume
- More flow visualization
  - Dye advection, LIC
Summary

- Model a volumetric computation by a DAG of virtual instructions with brick operands
- Priority function promoting instructions with cached operands
- Case studies using virtual instructions with brick operands
  - Propagation buffer
  - Boundary exchange strategy
Questions?
Extra slices
Advantages of bricking

- Process volumes too large to fit into physical memory
- Reduce intermediate memory and storage
- Skip region with no contribution
- Decrease delay in streaming
- Distribute workload to multiple devices
Types of virtual instructions

- Computation
- Boundary exchange
- Memory manipulation
- Unconditional
- Conditional
Light propagation across bricks

- Propagation order of bricks: same as the visibility order viewed from the light source

- Point light

- Directional light
Instruction Scheduling (HEFT method)

1. Schedule the ready instruction with the highest priority
   - Ready instruction: no predecessor
   - Choose a device with earliest-finish-time
   - Remove the instruction from DAG

3. Repeat until no more instruction
Implementation through OpenCL (cont’)

- Synchronization through events:
  - Find predecessors on other queues. Add their completion events into event waiting list
  - Set the event waiting list to the first OpenCL task
  - If any successor on another queue, the last OpenCL task outputs a completion event.
Bricked-based volume rendering

- Unshaded or gradient on-the-fly
Bricked-based volume rendering (cont’)

- Using gradient brick
Virtual instruction: steps

- Update unconditional instruction dependencies
- Schedule unconditional instructions
- Insert conditional memory manipulation instructions
- Execute
Software Architecture and Programming Interface

- Key functions in a C++ lib
  - Classes
    - Brick operands, grid of bricks, DAG, virtual instruction
  - Implementation
    - Scheduler, memory manipulation instructions, dependency & synchronization management,
  - Utilities to construct DAG
- To build an application
  - Application specific virtual instructions
  - Create a DAG
  - Instantiate a scheduler
  - Send the DAG to the scheduler