The Scout Compiler

Jeff Inman
jti@lanl.gov

Pat McCormick, Jim Ahrens

Advanced Computing Lab (CCS-1)
Los Alamos National Laboratory
What is Scout?

• tool for accelerating visualization & analysis on GPUs
  – interactive development environment (GUI / display)
  – programming language (compiler / runtime)

Scout: A Hardware-Accelerated System for Quantitatively Driven Visualization and Analysis
McCormick, P; Inman, J; Ahrens, J; Hansen C; Roth G
Proceedings of IEEE Visualization 2004, 171-178

(LAUR-04-6226)
Compiler Goals

- exploit GPU architecture / coprocessor
  - expose GPU power
  - convenient to code and run
- data-parallel
  - matches hardware
  - expressive
  - easy to learn and use
- support for visualization and analysis
  - ray-tracing, volume-rendering, point-rendering
  - coloring, clipping, selecting, viewports
Platform Challenges

• GPU represents an unusual virtual machine
  – co-processor
    • install one “pass”, bind arguments, execute (and maybe read results)
    • data movement
  – strictly functional
    • read-only inputs / write-only outputs
    • gather-only (no scatter)
  – some “mismatches” with CPU
    • smear, FP exceptions, clamping, etc.
    • vertex-engines, fragment-engines
• GPU hardware “churn”
  – new architecture every ~6 months
    • OpenGL extensions, looping, control of memory, Cg, etc.
• many architecture details are proprietary
Language Challenges

- hide OpenGL
  - user writes application, not infrastructure
- hide runtime
  - control of GPU passes
  - data-movement
Simple Example

```c
// Compute entropy from <pressure> and <density>
compute with (shapeof(density)){
    float entropy = pressure / pow(density, 4/3.0);

    [etc ...]
}
```

- hand-coding for the GPU would require ~250 lines of code
- language modeled after C* from Thinking Machines Corp.
  - simple data-parallel variables and expressions
Performance Comparison
(Does not include texture download and readback times)

320 x 320 x 320

<table>
<thead>
<tr>
<th>platform</th>
<th>time</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0 GHz Intel Xeon EM64T (scalar)</td>
<td>3.3 sec</td>
<td></td>
</tr>
<tr>
<td>3.0 GHz Intel Xeon EM64T (vector)</td>
<td>0.42 sec</td>
<td>7.9x</td>
</tr>
<tr>
<td>NVIDIA Quadro 3400</td>
<td>0.12 sec</td>
<td>3.5x (27.5x)</td>
</tr>
</tbody>
</table>
Hue / Saturation / Value / Alpha

// show HUE with full SAT and VALUE
render quad with(shapeOf(pt)) {
    image = hsva(i1 * 360, 1, 1, 1);
}

// SAT and VALUE with constant HUE
render quad with(shapeOf(pt)) {
    image = hsva(240.0, i1, j1, 1);
}
Another Example

```plaintext
render quad with(shapeof(pt)) {
    where(land)
        image = 0; // color the continents black
    else
        image = hsva(240 - norm(pt)*240, 1, 1, 1);
}
```
Heat Transfer

// compute next time-step
compute with(shapeof(t0)) {
    where(mask != 0) { // avoid boundaries...
        float tx, ty;
        tx = (alpha/(dx*dx))
            * t0[i+1][j] - 2*t0 + t0[i-1][j]);
        ty = (alpha/(dy*dy))
            * t0[i][j+1] - 2*t0 + t0[i][j-1]);
        t1 = dt * (tx + ty) + t0;
    }
}

// display next time-step value
render quad with(shapeOf(t1)) {
    image = hsva(240 - 240*t1, 1, 1, 1);
}

// swap t0 and t1 for next time-step
t0 = swap(t1);

Scout implementation and figures courtesy of Nehal Desai (CCS-1).
Heat Transfer (cont)

512 x 512 grid
3,482 time steps

2.2 GHz Opteron: 93 seconds
NVIDIA GeForce 7800: 6.7 seconds

Scout implementation and figures courtesy of Nehal Desai (CCS-1).
Dendrite Growth

- Data Set size: 240 x 240
- CPU version is optimized, production-quality code
  - one node of QSC (4 Alpha EV6, 16 GB shared)
  - Adaptive Mesh Refinement (SAMRAI, developed at LLNL)
- GPU version written in Scout
  - NV Quadro FX 4500 GPU with 512 MB RAM
  - full resolution

Scout implementation and figures courtesy of Jamaludin Mohd-Yusof (CCS-2).
Dendrite Growth (AMR)

QSC Simulation results provided by Sharen Cummins (CCS-2) and Juan Ramirez (MST-8).

Scout implementation and figures courtesy of Jamaludin Mohd-Yusof (CCS-2).
Dendrite Growth (performance)

Scout implementation and figures courtesy of Jamaludin Mohd-Yusof (CCS-2).
Dendrite Growth (3D with GPU)

- ~ 1 week programmer effort (2D representation of 3D data)
- runs faster than 2D code!
- not supported on the CPU

Scout implementation and figures courtesy of Jamaludin Mohd-Yusof (CCS-2).
Compiler Overview

Flow Graph

- exposes task-level parallelism
- simplifies runtime
Targeting IR to Hardware

- basic-block boundaries have an additional constraint
  - incoming branch
  - outgoing branch
  - architectural-affinity change
- blocks are “colored” by HW they target
  - CPU-only (e.g. iterating over GPU passes, glReadPixels, etc)
  - GPU-only (everything else)
- flow-graph has information about data-movement
Flow-Graph Node

- basic-block, plus dependencies (avoid pass-through dependencies!)
- bind / release semantics
- distinguish readable / writable bindings

![Diagram showing the states of a flow-graph node: not-yet-runnable, runnable, and completed.](image-url)
Optimization

• GPU code optimized by vendor at load time
  – we can infer: dead-code removal, dual-issue / co-issue
• loops complicated by multiple architectures
  – per datum versus multiple passes
• global optimization complicated by hidden architecture
• what we do:
  – minor local optimizations
  – avoid passing values through blocks
    (slight variation on usual data-dependency algorithms)
  – vertex-engines compute offset array indices
Flow-Graph Illustration with Reduction

- efficient GPU implementation requires multiple passes
Flow Graph  
(simple reduction)

```c
#define DBG_NET_DYNAMIC  
declare pt("data/pt_tiny.smf");

float sum;
compute with(shapeOf(pt)) {
    sum = gplus(pt);
}
render quad with(shapeOf(pt)) {
    image = pt / sum;
}
```

newer compiler implements reductions (etc) with single objects
assign _t2

enable iteration

loop header

enable true branch

loop body

enable false branch

swap _t14, _t2

.glReadPixels
Acknowledgements

- Nehal Desai (CCS-1) - heat transfer
- Jamaludin Mohd-Yusof (CCS-2) - dendrite growth
- John Turner (CCS-2)
END
Dendrite Growth

- coupled PDEs for phase field and enthalpy variables
- dendrite has 4-way symmetry using homogenous Neumann (symmetry) boundary conditions on all sides

**phase field:**

\[
\left[A(n)\right]^2 \frac{1}{Le} \frac{\partial \phi}{\partial t} = \phi(1-\phi^2) - \lambda(1-\phi^2)\theta + \nabla \cdot \left(\left[A(n)\right]^2 \nabla \phi\right) \\
+ \frac{\partial}{\partial x} \left[ A(n)A'(n) \frac{\partial \phi}{\partial y} \right] + \frac{\partial}{\partial y} \left[ A(n)A'(n) \frac{\partial \phi}{\partial x} \right]
\]

**enthalpy:**

\[
\frac{\partial \theta}{\partial t} = (LeD)\nabla^2 \theta + \frac{1}{2} \frac{\partial \phi}{\partial t}
\]
Debugging with Scout

- Can be difficult to debug GPU programs
  
  ```
  where (isNaN(pt)) { ... }
  ```

- Visualization can be helpful
  - compute with replaced by render with
  - multiple viewports

- re-compilation probably necessary (to isolate stmts from shaders)
  - per-statement flow-graph?

**Status:**
- Work in progress...
- A lot of work will be required to make a decent debugging tool
Team Members / Collaborators

- LANL Scout Team
  - Pat McCormick (project leader)
  - Jim Ahrens
  - Jeff Inman jti@lanl.gov

- UC Davis
  - John Owens
  - Adam Moershell

- University of Utah
  - Chuck Hansen
  - Greg Roth