Auto-Generation and Auto-Tuning of 3D Stencil Codes on GPU Clusters

Yongpeng Zhang, Frank Mueller
North Carolina State University
CGO'2012
Outline

‣ Motivation
‣ DSL front-end and Benchmarks
‣ Framework
‣ Experimental Results
‣ Conclusion
Motivation

- Main-stream microprocessors more parallel
  - GPU becomes a popular accelerator nowadays
- Achieving near-optimal perf. on GPUs still difficult
  - Perf. affected by a multitude of architectural features -- tradeoffs
  - Architectural difference b/w generations of hw line
  - Hand-tuning for all optimization options? -- counter-productive
- Domain Specific Languages (HTML, MATLAB, SQL)
  - Balance portability, performance and programmability
  - Sacrifice language generality
  - Performance comparable to hand-coded code
DSL for 3D Stencil

- Portable source-to-source (DSL to CUDA) framework
  - Auto-generation and auto-tuning for different GPUs
  - Iterative 3D Jacobi stencil computation
- Based on prior research to accelerate 3D stencil on GPUs
  - Summarize optimization techniques
- Abstract stencil into domain-specific specifications
- Extract critical tuning parameters
- Close to hand-written code
Related Work

‣ Auto-tuning: portability and productivity
  ‣ Libraries: ATLAS [2], OSKI [23], FFTW [6]

‣ Auto-tuning on GPUs
  ‣ Sparse Matrix-vector multiplication [7]
  ‣ GEMM [12], 3D FFT [19]

‣ Stencil on GPUs
  ‣ Hand-coded ([17, 18, 20])
  ‣ Ease of programming (auto-generation) ([5, 11, 14, 22])
  ‣ Tuning one parameter ([13, 16])

‣ Our work offers both performance and programmability
  ‣ Code auto-generation and auto-tuning
Stencil DSL

\[
out([i][j][k]) = \sum_{m} w_{m} \times in[i \pm I_{m}][j \pm J_{m}][k \pm K_{m}]
+ \sum_{l} w_{l}[i][j][k] \times in[i \pm I_{l}][j \pm J_{l}][k \pm K_{l}]
+ \sum_{n} w_{n} \times in_{n}
\]  

A sequence of such equations

- Single output array \((out[i][j][k])\)
- Single input array \((in[i + l][j + l][k + K])\)
- Multiple input parameter array \((w[i][j][k])\)
- Multiple temporary variable \((out)\)
Benchmarks (7/13/19/27 point)

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-point order-1</td>
<td>$\text{tmp} = (u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k} + u_{i,j,k+1} + u_{i,j,k-1}) \times \beta$; $u_{1,i,j,k} = \text{tmp} + \alpha \times u_{i,j,k}$;</td>
</tr>
<tr>
<td>13-point order-2</td>
<td>$\text{tmp} = \text{coef1} \times (u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k} + u_{i,j,k+1} + u_{i,j,k-1})$; $\text{tmp} = \text{coef2} \times (u_{i+2,j,k} + u_{i-2,j,k} + u_{i,j+2,k} + u_{i,j-2,k} + u_{i,j,k+2} + u_{i,j,k-2})$; $u_{1,i,j,k} = \text{tmp} + \text{coef0} \times u_{i,j,k}$;</td>
</tr>
<tr>
<td>19-point order-1 (himen)</td>
<td>$s0 = \text{wrk}1_{i,j,k} + a0d_{i,j,k} \times p_{i,j,k+1} + a1d_{i,j,k} \times p_{i,j,k+1}$; $s0 = b0d_{i,j,k} \times (p_{i,j+k+1} - p_{i,j-1},k+1 - p_{i,j+1},k-1 + p_{i,j-1},k-1) + a2d_{i,j,k} \times p_{i+1,j+k}$; $s0 = b1d_{i,j,k} \times (p_{i+1,j+k+1} - p_{i-1,j},k+1 - p_{i+1,j-1},k + p_{i-1,j-1},k);$ $s0 = b2d_{i,j,k} \times (p_{i+1,j+k+1} - p_{i-1,j},k+1 - p_{i+1,j-1},k+1 + p_{i-1,j-1},k);$ $s0 = c1d_{i,j,k} \times p_{i,j-1},k + c2d_{i,j,k} \times p_{i-1,j,k};$ $ss = (s0 \times a3d_{i,j,k} - p_{i,j,k}) \times bnd_{i,j,k};$ $\text{wrk}2_{i,j,k} = p_{i,j,k} + \omega \times ss$;</td>
</tr>
<tr>
<td>27-point order-1</td>
<td>$b_{i,j,k} = \text{param0} \times a_{i,j,k}$; $+ \text{param1} \times (a_{i-1,j,k} + a_{i+1,j,k} + a_{i,j-1,k} + a_{i,j+1,k} + a_{i,j,k-1} + a_{i,j,k+1})$; $+ \text{param2} \times (a_{i-1,j-1,k} + a_{i-1,j+1,k} + a_{i+1,j-1,k} + a_{i+1,j+1,k} + a_{i-1,j,k-1} + a_{i+1,j,k-1} + a_{i,j-1,k+1} + a_{i,j+1,k+1} + a_{i,j,k-1} + a_{i,j,k+1})$; $+ \text{param3} \times (a_{i-1,j-1,k-1} + a_{i-1,j-1,k+1} + a_{i-1,j+1,k-1} + a_{i-1,j+1,k+1} + a_{i-1,j,k-1} + a_{i-1,j,k+1} + a_{i+1,j-1,k-1} + a_{i+1,j-1,k+1} + a_{i+1,j+1,k-1} + a_{i+1,j+1,k+1})$;</td>
</tr>
</tbody>
</table>
- Parse specification file
- Extract stencil features
  - Halo region (Im, Jm, Km)
  - input/output array name, input scaler name
  - Determine corner or non-corner stencil
  - data type (float or double)
Decomposing Stencil Space

- Tuning parameters
  - BlockSize.x (16, 32, 64)
  - BlockSize.y (2, 3..16)
  - BlockDim.x (16, 32, 64, and divisible by BlockSize.x)
  - BlockDim.y (2...16 and divisible by BlockSize.y)
  - Input array texture mapping (boolean)
Decomposing Stencil Space

Tuning parameters

- BlockSize.x (16, 32, 64)
- BlockSize.y (2,3..16)
- BlockDim.x (16, 32, 64, and divisible by BlockSize.x)
- BlockDim.y (2...16 and divisible by BlockSize.y)
- Input array texture mapping (boolean)
Decomposing Stencil Space

- Tuning parameters
  - BlockSize.x (16, 32, 64)
  - BlockSize.y (2, 3..16)
  - BlockDim.x (16, 32, 64, and divisible by BlockSize.x)
  - BlockDim.y (2...16 and divisible by BlockSize.y)
  - Input array texture mapping (boolean)
Based on two templates

- With or w/o corner access
- Different usage of shared memory
  - No corner: Load just one plane to shared memory
  - Corner: Load \((1+2*\text{Km})\) planes to shared memory
- Experience learned from hand-written stencil code

Gen unrolling code given block sizes and block dims
Using shared memory

- More data points than threads
- One plane at a time
- Load Internal region first (1)
- Code generator to load margins (2)
  - Use constant memory for offset
  - No branches
  - No redundant memory load
__const__ __device__ int haloThreadMappingX[4][6] =
{{1,2,3,4,5,6}, {1,2,3,4,5,6}, {0,0,0,0,7,7}, {7,7,0,0,0,0}};

__const__ __device__ int haloThreadMappingY[4][6] =
{{0,0,0,0,0,0}, {5,5,5,5,5,5}, {1,2,3,4,1,2}, {3,4,0,0,0,0}};

__global__ stencil(...) {
    ...
    halo_offsetx = haloThreadMappingX[threadIdx.x][threadIdx.y];
    halo_offsety = haloThreadMappingY[threadIdx.x][threadIdx.y];
    for (k = HALO_MARGIN_Z; k < zSize + HALO_MARGIN_Z; k++) {
        ....
        // load internal
        shMem[threadIdx.y][threadIdx.x] = input[myindexY][myindexX];
        // load margins, no branches
        shMem[halo_offsetx][halo_offsety] = input[myindexY+halo_offsety-HALO_MARGIN_Y][myindexX+halo_offsetx-HALO_MARGIN_X];
        ...
    }
    }
}
Tuning Engine

- Brute force search all space
- Instantiate each tuning setting and profile
- Offline

```cpp
opt = max;
for all block sizes
    for all applicable block dims
        for useTexture = true/false
        {
            timing = profile execution;
            opt = timing < opt? timing : opt;
        }
```
Going to Multi-Node

- Overlapping comp. and comm.
- MPI asynchronous send/recv
- CUDA asynchronous memcpy
- Separate kernels
  - For margin layer
  - for non-continuous to continuous memcpy
## Experiment Setup

<table>
<thead>
<tr>
<th>Model</th>
<th>SM Count</th>
<th>Core Count</th>
<th>L1 Cache</th>
<th>Bandwidth (GB/s)</th>
<th>Register</th>
<th>Shared Memory</th>
<th>SP Gflops</th>
<th>DP Gflops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geforce GTX 280</td>
<td>30</td>
<td>240</td>
<td>N</td>
<td>141.7</td>
<td>16KB</td>
<td>16KB</td>
<td>933</td>
<td>78</td>
</tr>
<tr>
<td>Tesla C1060</td>
<td>30</td>
<td>240</td>
<td>N</td>
<td>102.4</td>
<td>16KB</td>
<td>16KB</td>
<td>933</td>
<td>78</td>
</tr>
<tr>
<td>Tesla C2050</td>
<td>14</td>
<td>448</td>
<td>Y</td>
<td>144</td>
<td>32KB</td>
<td>16 or 48 KB</td>
<td>1288</td>
<td>515</td>
</tr>
<tr>
<td>Geforce GTX 480</td>
<td>15</td>
<td>480</td>
<td>Y</td>
<td>177.4</td>
<td>32KB</td>
<td>16 or 48 KB</td>
<td>1345</td>
<td>168</td>
</tr>
</tbody>
</table>

### Single GPU

<table>
<thead>
<tr>
<th>Cluster</th>
<th># nodes</th>
<th>Network Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tesla C2050</td>
<td>36</td>
<td>32Gbps</td>
</tr>
<tr>
<td>Geforce GTX 480</td>
<td>48</td>
<td>32Gbps</td>
</tr>
</tbody>
</table>

### GPU Clusters
Single Node Auto-tuning

<table>
<thead>
<tr>
<th>Model</th>
<th>BlockSize.x</th>
<th>BlockSize.y</th>
<th>BlockDim.x</th>
<th>BlockDim.y</th>
<th>Texture</th>
<th>SP Gflops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geforce GTX 280</td>
<td>64/32/64/16</td>
<td>8/8/3/6</td>
<td>32/32/64/16</td>
<td>8/2/3/2</td>
<td>Y/Y/N/N</td>
<td>76.0/117.0/57.6/94.2</td>
</tr>
<tr>
<td>Tesla C1060</td>
<td>64/64/64/32</td>
<td>8/6/6/8</td>
<td>32/64/64/32</td>
<td>8/2/3/2</td>
<td>Y/N/Y/N</td>
<td>57.5/91.8/44.8/95.5</td>
</tr>
<tr>
<td>Tesla C2050</td>
<td>64/64/64/64</td>
<td>8/6/3/4</td>
<td>32/64/32/32</td>
<td>8/3/3/4</td>
<td>Y/Y/N/Y</td>
<td>87.3/133.8/64.6/157.6</td>
</tr>
<tr>
<td>Geforce GTX 480</td>
<td>64/64/64/64</td>
<td>3/3/3/8</td>
<td>32/32/32/32</td>
<td>3/3/3/4</td>
<td>Y/Y/N/Y</td>
<td>108.2/167.8/77.4/203.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>BlockSize.x</th>
<th>BlockSize.y</th>
<th>BlockDim.x</th>
<th>BlockDim.y</th>
<th>Texture</th>
<th>DP Gflops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geforce GTX 280</td>
<td>16/16/16/16</td>
<td>16/16/16/16</td>
<td>16/16/16/16</td>
<td>4/8/3/3</td>
<td>N/N/Y/N</td>
<td>32.5/35.4/24.0/29.0</td>
</tr>
<tr>
<td>Tesla C1060</td>
<td>32/16/32/16</td>
<td>6/16/4/6</td>
<td>32/16/32/16</td>
<td>2/8/2/3</td>
<td>N/n/Y/N</td>
<td>28.8/35.3/22.8/29.3</td>
</tr>
<tr>
<td>Tesla C2050</td>
<td>64/32/64/32</td>
<td>8/6/3/6</td>
<td>32/32/64/32</td>
<td>4/2/3/2</td>
<td>Y/Y/N/Y</td>
<td>45.9/66.8/31.8/97.7</td>
</tr>
<tr>
<td>Geforce GTX 480</td>
<td>64/32/64/32</td>
<td>6/6/3/4</td>
<td>32/32/64/16</td>
<td>3/2/3/4</td>
<td>Y/Y/N/Y</td>
<td>55.2/77.2/38.7/86.0</td>
</tr>
</tbody>
</table>

7/13/19/27 stencil

No general pattern
# Single Node Auto-tuning

<table>
<thead>
<tr>
<th>Model</th>
<th>BlockSize.x</th>
<th>BlockSize.y</th>
<th>BlockDim.x</th>
<th>BlockDim.y</th>
<th>Texture</th>
<th>SP Gflops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geforce GTX 280</td>
<td>64/32/64/16</td>
<td>8/8/3/6</td>
<td>32/32/64/16</td>
<td>8/2/3/2</td>
<td>Y/Y/N/N</td>
<td>76.0/117.0/57.6/94.2</td>
</tr>
<tr>
<td>Tesla C1060</td>
<td>64/64/64/32</td>
<td>8/6/6/8</td>
<td>32/64/64/32</td>
<td>8/2/3/2</td>
<td>Y/N/Y/N</td>
<td>57.5/91.8/44.8/95.5</td>
</tr>
<tr>
<td><strong>Tesla C2050</strong></td>
<td>64/64/64/64</td>
<td>8/6/3/4</td>
<td>32/64/32/32</td>
<td>8/3/3/4</td>
<td>Y/Y/N/Y</td>
<td>87.3/133.8/64.6/157.6</td>
</tr>
<tr>
<td>Geforce GTX 480</td>
<td>64/64/64/64</td>
<td>3/3/3/8</td>
<td>32/32/32/32</td>
<td>3/3/3/4</td>
<td>Y/Y/N/Y</td>
<td>108.2/167.8/77.4/203.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>BlockSize.x</th>
<th>BlockSize.y</th>
<th>BlockDim.x</th>
<th>BlockDim.y</th>
<th>Texture</th>
<th>DP Gflops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geforce GTX 280</td>
<td>16/16/16/16</td>
<td>16/16/16/16</td>
<td>16/16/16/16</td>
<td>4/8/3/3</td>
<td>N/N/Y/N</td>
<td>32.5/35.4/24.0/29.0</td>
</tr>
<tr>
<td>Tesla C1060</td>
<td>32/16/32/16</td>
<td>6/16/4/6</td>
<td>32/16/32/16</td>
<td>2/8/2/3</td>
<td>N/n/Y/N</td>
<td>28.8/35.3/22.8/29.3</td>
</tr>
<tr>
<td>Tesla C2050</td>
<td>64/32/64/32</td>
<td>8/6/3/6</td>
<td>32/32/64/32</td>
<td>4/2/3/2</td>
<td>Y/Y/N/Y</td>
<td>45.9/66.8/31.8/97.7</td>
</tr>
<tr>
<td>Geforce GTX 480</td>
<td>64/32/64/32</td>
<td>6/6/3/4</td>
<td>32/32/64/16</td>
<td>3/2/3/4</td>
<td>Y/Y/N/Y</td>
<td>55.2/77.2/38.7/86.0</td>
</tr>
</tbody>
</table>

7/13/19/27 stencil

*No general pattern*
## Single Node Auto-tuning

<table>
<thead>
<tr>
<th>Model</th>
<th>BlockSize.x</th>
<th>BlockSize.y</th>
<th>BlockDim.x</th>
<th>BlockDim.y</th>
<th>Texture</th>
<th>SP Gflops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geforce GTX 280</td>
<td>64/32/64/16</td>
<td>8/8/3/6</td>
<td>32/32/64/16</td>
<td>8/2/3/2</td>
<td>Y/Y/N/N</td>
<td>76.0/117.0/57.6/94.2</td>
</tr>
<tr>
<td>Tesla C1060</td>
<td>64/64/64/32</td>
<td>8/6/6/8</td>
<td>32/64/64/32</td>
<td>8/2/3/2</td>
<td>Y/N/Y/N</td>
<td>57.5/91.8/44.8/95.5</td>
</tr>
<tr>
<td><strong>Tesla C2050</strong></td>
<td><strong>64/64/64/64</strong></td>
<td><strong>8/6/3/4</strong></td>
<td><strong>32/64/32/32</strong></td>
<td><strong>8/3/3/4</strong></td>
<td><strong>Y/Y/N/Y</strong></td>
<td><strong>87.3/133.8/64.6/157.6</strong></td>
</tr>
<tr>
<td>Geforce GTX 480</td>
<td><strong>64/64/64/64</strong></td>
<td><strong>3/3/3/8</strong></td>
<td><strong>32/32/32/32</strong></td>
<td><strong>3/3/3/4</strong></td>
<td><strong>Y/Y/N/Y</strong></td>
<td><strong>108.2/167.8/77.4/203.7</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>BlockSize.x</th>
<th>BlockSize.y</th>
<th>BlockDim.x</th>
<th>BlockDim.y</th>
<th>Texture</th>
<th>DP Gflops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geforce GTX 280</td>
<td>16/16/16/16</td>
<td>16/16/16/16</td>
<td>16/16/16/16</td>
<td>4/8/3/3</td>
<td>N/N/Y/N</td>
<td>32.5/35.4/24.0/29.0</td>
</tr>
<tr>
<td>Tesla C1060</td>
<td>32/16/32/16</td>
<td>6/16/4/6</td>
<td>32/16/32/16</td>
<td>2/8/2/3</td>
<td>N/n/Y/N</td>
<td>28.8/35.3/22.8/29.3</td>
</tr>
<tr>
<td>Tesla C2050</td>
<td><strong>64/32/64/32</strong></td>
<td><strong>8/6/3/6</strong></td>
<td><strong>32/32/64/32</strong></td>
<td><strong>4/2/3/2</strong></td>
<td><strong>Y/Y/N/Y</strong></td>
<td><strong>45.9/66.8/31.8/97.7</strong></td>
</tr>
<tr>
<td>Geforce GTX 480</td>
<td><strong>64/32/64/32</strong></td>
<td><strong>6/6/3/4</strong></td>
<td><strong>32/32/64/16</strong></td>
<td><strong>3/2/3/4</strong></td>
<td><strong>Y/Y/N/Y</strong></td>
<td><strong>55.2/77.2/38.7/86.0</strong></td>
</tr>
</tbody>
</table>

**7/13/19/27 stencil**

**No general pattern**
- **Unrolling is necessary**
  - **BlockDim.x/y almost always differ from BlockSize.x/y**
  - **Except for 19-point DP Stencil for Fermi GPUs (BlockSize.y = 3, too small to unroll)**
Search for Block Sizes and Block Dims

Find Optimal Block Size at (64,8)

Find Optimal Block Dim, fixing block size at (64,8)

- GTX 280 7-point stencil SP
- Block size first, block dim second
- Local optima exist ((32,8) and (64,4) on right)
Weak Scaling in Multi-Node

- 13-point stencil is network bandwidth-bounded
- 7-point stencil comm-bounded in GTX 480 cluster
  - Because of higher Gflops on GTX 480
Comparison with Prior Work

‣ Our Gflops is **32.5** Gflops for 7-point DP in GTX280

‣ Datta* (SC’08) **36** Gflops (Same BlockSize and BlockDim)

‣ Philips* et al. (IPDPS’10, 19-point) **50** Gflops (44.8/47.6(normalized), similar BlockSize and BlockDim)

‣ Nguyen* et al. (SC’10) (similar to SC’08)

‣ Kamil** et al. (IPDPS’10) (14 Gflops)

‣ Unat** et al. (ICS’11) (22 Gflops on C1060, ours: **28** Gflops)

‣ Christen** et al (IPDPS’11) and Maruyama** et al. (SC’11)
  • Only SP results were reported, both inferior to ours.

* Hand written    ** Auto Generation
## Comparison with Prior Work

<table>
<thead>
<tr>
<th>Prior work</th>
<th>Benchmark</th>
<th>Theirs</th>
<th>Ours</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datta* (SC’08)</td>
<td>7-point DP on GTX 280</td>
<td>36 GFlops</td>
<td>32.5 GFlops</td>
<td>Same BlockSizes and BlockDims</td>
</tr>
<tr>
<td>Philips* et al. (IPDPS’10)</td>
<td>19-point SP on C1060</td>
<td>50 GFlop</td>
<td>44.8/47.6 (normalized)</td>
<td>Similar BlockSizes and BlockDims</td>
</tr>
<tr>
<td>Nguyen* et al. (SC’10)</td>
<td>7-point DP on GTX 285</td>
<td>36 GFlops</td>
<td>32.5 GFlops</td>
<td></td>
</tr>
<tr>
<td>Kamil** et al. (IPDPS’10)</td>
<td>7-point DP on GTX 280</td>
<td>14 GFlops</td>
<td>32.5 GFlops</td>
<td></td>
</tr>
<tr>
<td>Unat** et al. (ICS’11)</td>
<td>7-point DP on C1060</td>
<td>22 GFlops</td>
<td>32.5 GFlops</td>
<td></td>
</tr>
<tr>
<td>Christen** et. al (IPDPS’11)</td>
<td>7-point SP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maruyama** (SC’11)</td>
<td>7-point SP</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Hand written  ** Auto Generation
Conclusion

‣ GPU programmability and performance not mutually exclusive

‣ DSL front-end reduces programming effort

‣ Auto-tuning helps achieving near-optimal performance
Thank you!