Adaptable Two-Dimension Sliding Windows on NVIDIA GPUs with Runtime Compilation

Nicholas Moore and Miriam Leeser
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA

Laurie Smith King
Dept. of Mathematics and Computer Science
College of the Holy Cross
Worcester, MA

Supported by
Northeastern
MathWorks
Motivation

- GPUs offer significant performance potential
- GPU development is difficult
  - Complicated target with changes over time
- Leads to problem-specific non-reusable code
  - Affects library developers and users
- Goal: more adaptable kernel implementations
  - Case study: template matching application
  - Technique: problem-specific kernel compilation
Template Matching (1)

- Real-world tumor tracking application
  - Ying Cui, Jennifer Dy, Gregory Sharp, Brian Alexander, and Steve Jiang
- Visual tracking of tumor
  - Focused radiotherapy
  - Tumor moves during breathing

Template Matching (2)

Incoming Frame

Matching

Template 1

S1, L1

Template 2

S2, L2

Voting

SN, LN

Template N

Location
corr2()

corr2(A, B) = \frac{\sum_{M} \sum_{N} (A_{MN} - \bar{A})(B_{MN} - \bar{B})}{\sqrt{\left(\sum_{M} \sum_{N} (A_{MN} - \bar{A})^2\right) \left(\sum_{M} \sum_{N} (B_{MN} - \bar{B})^2\right)}}

- Sliding window template matching
- Pearson's correlation for similarity score
- Floating-point data
  - Templates and frames pre-processed
Computation Reduction

\[
\text{corr2}(A, B) = \frac{\sum \sum \left( A_{MN} - \bar{A} \right) \left( B_{MN} - \bar{B} \right)}{\sqrt{\left( \sum \sum \left( A_{MN} - \bar{A} \right)^2 \right) \left( \sum \sum \left( B_{MN} - \bar{B} \right)^2 \right)}}
\]

- Template data (A)
  - Not expected to be separable
  - Fixed for given template
Computation Reduction

\[
\text{corr2}(A, B) = \frac{\sum_M \sum_N A_{MN}^C (B_{MN} - \bar{B})}{\sqrt{A^D \sum_M \sum_N (B_{MN} - \bar{B})^2}}
\]

- Template data (A)
  - Not expected to be separable
  - Fixed for given template
Computation Reduction

\[ \text{corr2}(A, B) = \frac{\sum_M \sum_N A_{MN} C (B_{MN} - \bar{B})}{\sqrt{A^D \sum_M \sum_N (B_{MN} - \bar{B})^2}} \]

- ROI data (B)
  - Dependent on window location and frame
  - Subtraction complicates frequency domain
## Reference Data Sets

<table>
<thead>
<tr>
<th>Patient</th>
<th>Templates</th>
<th>Template Size (pixels)</th>
<th>Shift ±V/±H (pixels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>53×54</td>
<td>18/9</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>23×21</td>
<td>11/5</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>76×45</td>
<td>9/4</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>156×116</td>
<td>9/3</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>86×78</td>
<td>11/6</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>141×107</td>
<td>9/2</td>
</tr>
</tbody>
</table>

- Large templates
  - Significant variation in dimensions
- Small search with single ROI per frame
- Different part of the problem space
Convolution Implementations

- Kong et al. (GPGPU 2010)
  - Template stored in shared memory
  - Only 7×7 kernels presented
- NVIDIA Performance Primitives
  - Only supports uint8
- Accelereyes Jacket
  - Last documented version supports arbitrary kernels up to 5×5, square kernels to 10×10
- OpenCV
  - Supports single precision floating point
  - Non-separable templates stored in constant memory.
CUDA Mapping Complications

- Common correlation case:
  - Small template
  - Large image with many window locations

- Template matching application:
  - Templates too large to use shared or constant memory
  - Few sources of parallelism
    - Few templates (10 to 14)
    - Relatively small ROI (95 to 703 positions)
    - Single ROI per frame
  - Problem parameters vary between patients
CUDA Mapping Solution

- Tiling of the template
  - Reduces local working set size
  - More independent parallelism
- Problem-specific kernel compilation
  - Adaptability without performance impact
CUDA Implementation

\[ \text{corr2}(A, B) = \frac{\sum_M \sum_N A_{MN}^C (B_{MN} - \bar{B})}{\sqrt{A^D \sum_M \sum_N (B_{MN} - \bar{B})^2}} \]

- Multiple pass implementation
  - Average, denominator, and numerator similar
- Outer loops are all addition
Tiled Template (1)

- Tile and process sub-templates separately
  - More parallelism
  - Reduces working set size to fit in shared memory
- Tiles mapped across CUDA grid
  - Scales to arbitrary template sizes
Tiled Template (2)

- Efficient tile size may not match problem
- Corr2() complicates padding
- Varying template size per block
Experimental Setup

- Benchmarked tile sizes from $4 \times 4$ to $16 \times 16$
- Compared against
  - MATLAB and pthreads-based C application
  - Both used constant template optimization
- Benchmarking
  - Intel Xeon W3580 (4 Nehalem cores @ 3.33 GHz, 6MB L2)
  - NVIDIA GeForce GTX 480 (Fermi) with CUDA 3.2
  - 64-bit Linux (GCC 4.4.3) and MATLAB R2010a
Performance

- Good performance across patients
- Steady-state streaming
  - Includes data transfer

**GPU vs CPU:**

<table>
<thead>
<tr>
<th>Patient</th>
<th>Template Size</th>
<th>Best Tile Size</th>
<th>Total Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>53×54</td>
<td>16×2</td>
<td>7.80</td>
</tr>
<tr>
<td>2</td>
<td>23×21</td>
<td>4×4</td>
<td>1.57</td>
</tr>
<tr>
<td>3</td>
<td>76×45</td>
<td>8×8</td>
<td>8.48</td>
</tr>
<tr>
<td>4</td>
<td>156×116</td>
<td>16×10</td>
<td>12.67</td>
</tr>
<tr>
<td>5</td>
<td>86×78</td>
<td>8×8</td>
<td>12.50</td>
</tr>
<tr>
<td>6</td>
<td>141×107</td>
<td>16×10</td>
<td>14.78</td>
</tr>
</tbody>
</table>

**Average Per-Frame Processing Times**

- MATLAB
- GPU
- Multithreaded C

---

*Note:* The table and graph provide a comparison of processing times and speedups for different patients, illustrating the performance benefits of GPU and multithreaded C implementations over MATLAB.
Tile Size Selection (1)

- Trade-off between efficiency and parallelism
  - Limited execution hardware
- Patient 2
  - Small tiles for more parallelism

<table>
<thead>
<tr>
<th>Patient</th>
<th>Template Size</th>
<th>Best Tile Size</th>
<th>Total Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>53×54</td>
<td>16×2</td>
<td>7.80</td>
</tr>
<tr>
<td>2</td>
<td>23×21</td>
<td>4×4</td>
<td>1.57</td>
</tr>
<tr>
<td>3</td>
<td>76×45</td>
<td>8×8</td>
<td>8.48</td>
</tr>
<tr>
<td>4</td>
<td>156×116</td>
<td>16×10</td>
<td>12.67</td>
</tr>
<tr>
<td>5</td>
<td>86×78</td>
<td>8×8</td>
<td>12.50</td>
</tr>
<tr>
<td>6</td>
<td>141×107</td>
<td>16×10</td>
<td>14.78</td>
</tr>
</tbody>
</table>
Tile Size Selection (2)

- Trade-off between efficiency and parallelism
  - Limited execution hardware
- Patient 4
  - 4×4 tiles results in no edge cases
  - Larger 16×10 tiles generates enough parallelism
    - 16×6, 12×16, and 12×6 edge tiles

<table>
<thead>
<tr>
<th>Patient</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template Size</td>
<td>53×54</td>
<td>23×21</td>
<td>76×45</td>
<td>156×116</td>
<td>86×78</td>
<td>141×107</td>
</tr>
<tr>
<td>Best Tile Size</td>
<td>16×2</td>
<td>4×4</td>
<td>8×8</td>
<td>16×10</td>
<td>8×8</td>
<td>16×10</td>
</tr>
<tr>
<td>Total Speedup</td>
<td>7.80</td>
<td>1.57</td>
<td>8.48</td>
<td>12.67</td>
<td>12.50</td>
<td>14.78</td>
</tr>
</tbody>
</table>
CUDA Adaptability

- Adaptability may affect performance
  - Compile-time optimizations not-possible
    - Loop unrolling
    - Strength reduction (esp. % or /)
  - Increased resource usage
- Mitigate issues with problem-specific kernel compilation
Problem-Specific Kernel Compilation (PSKC)

- No C-level source compilation in CUDA API
  - Productivity and portability vs. PTX
- Framework for runtime compilation
  - Part of larger set of GPU host-code abstractions
  - Automates compilation and loading of modules
- nvcc called at runtime
  - Kernels written in terms of unspecified compile-time constants
  - -D flag used to set parameters
- Overhead acceptable: one time setup, then streaming
PSKC: Current Benefits

- Loop unrolling for all tile regions
  - Instantiation of separate computation loops with C++ templates
- Strength reduction
  - Bit-wise offset calculations
  - Instance & implementation parameter values inlined
- Register usage reduction
Conclusions

- Tiled implementation allows for processing of large templates
  - Better usage of fast memories
  - Better performance through better parallelism
- Problem-specific kernel compilation supports adaptability at runtime
  - Loop unrolling, strength reduction, efficient register usage
- Future work: ability to adapt to both problem and hardware
  - Problem and implementation parameterization
    - Applications: particle image velocimetry
    - Different GPUs
  - PSKC: quantify benefits and explore limits
Thank You

Nicholas Moore: nmoore@coe.neu.edu
Miriam Leeser: mel@coe.neu.edu
Performance Breakdown

GPU Runtime Breakdown

- **Time (ms)**

- **Patient**
  - Patient 1
  - Patient 2
  - Patient 3
  - Patient 4
  - Patient 5
  - Patient 6

- **Legend**
  - Numerator Tiles
  - Denominator Tiles
  - Average Tiles
  - Denominator Reduction
  - Numerator Reduction
  - Average Reduction
  - Final Fraction
  - Host to Device Transfer
  - Device to Host Transfer