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

Nicholas Moore and Miriam Leeser

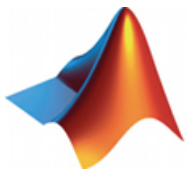
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA

Laurie Smith King

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

Northeastern

Supported by



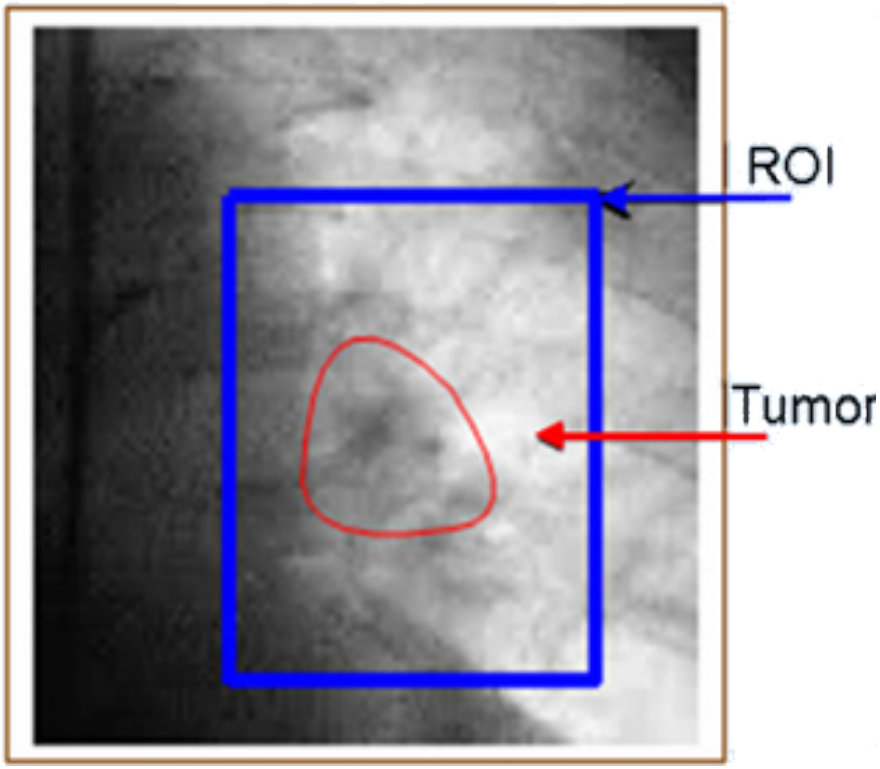
MathWorks®

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

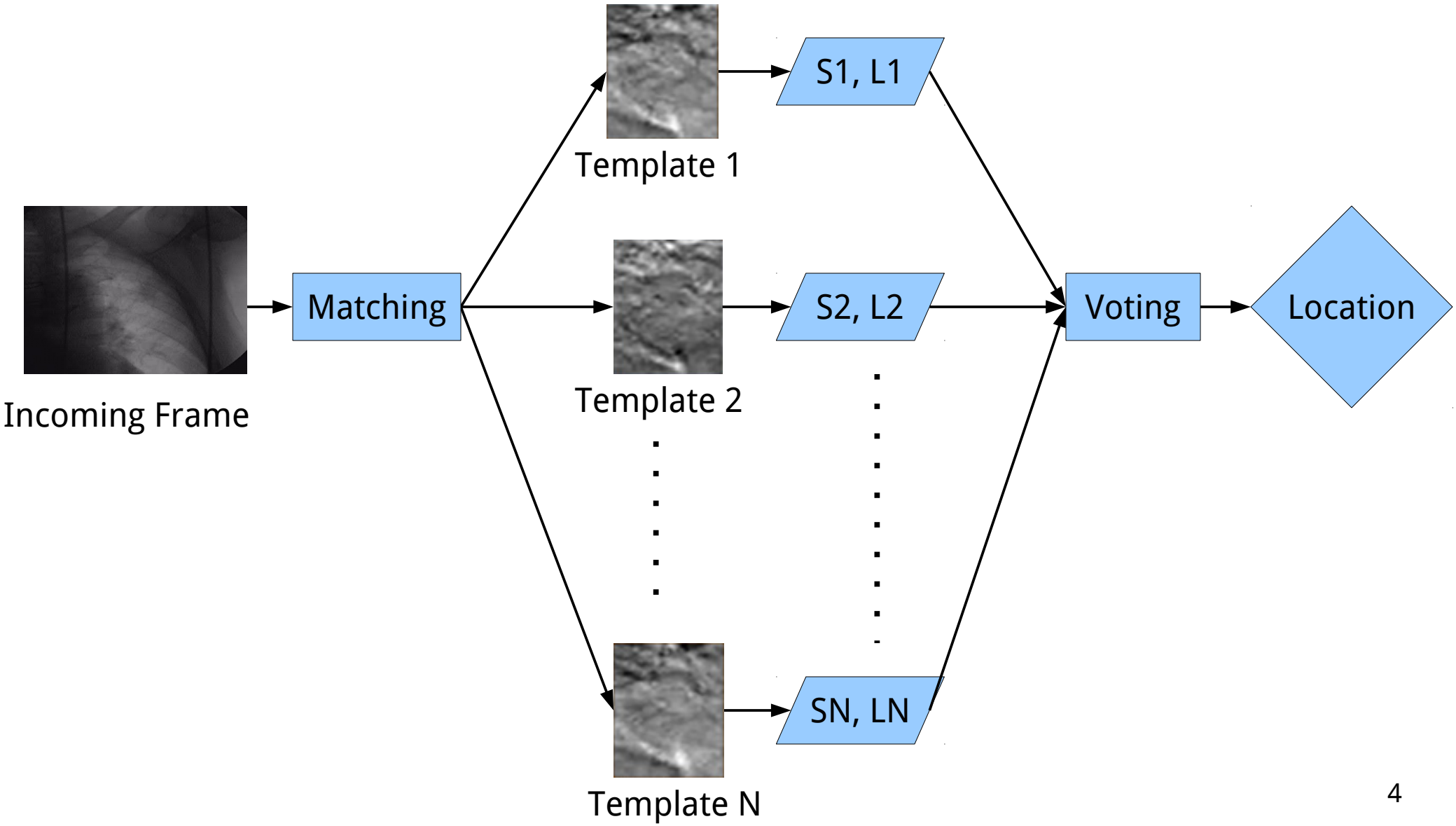
Template Matching (1)

- 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



Y. Cui, J. G. Dy, G. C. Sharp, B. Alexander, and S. B. Jiang, "Multiple Template Based Fluoroscopic Tracking of Lung Tumor Mass without Implanted Fiducial Markers," Physics in Medicine and Biology, Vol. 52, pp. 6229-6242, 2007.

Template Matching (2)



corr2()

$$corr2(A,B)=\frac{\sum_M\sum_N(A_{MN}-\bar{A})(B_{MN}-\bar{B})}{\sqrt{(\sum_M\sum_N(A_{MN}-\bar{A})^2)(\sum_M\sum_N(B_{MN}-\bar{B})^2)}}$$

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

Computation Reduction

$$corr2(A,B)=\frac{\sum_M\sum_N (A_{MN}-\bar{A})(B_{MN}-\bar{B})}{\sqrt{(\sum_M\sum_N (A_{MN}-\bar{A})^2)(\sum_M\sum_N (B_{MN}-\bar{B})^2)}}$$

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

Computation Reduction

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

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

Computation Reduction

$$corr2(A,B)=\frac{\sum_M\sum_N A^C_{MN}\left(B_{MN}-\bar{B}\right)}{\sqrt{A^D\sum_M\sum_N\left(B_{MN}-\bar{B}\right)^2}}$$

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

Reference Data Sets

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

- 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

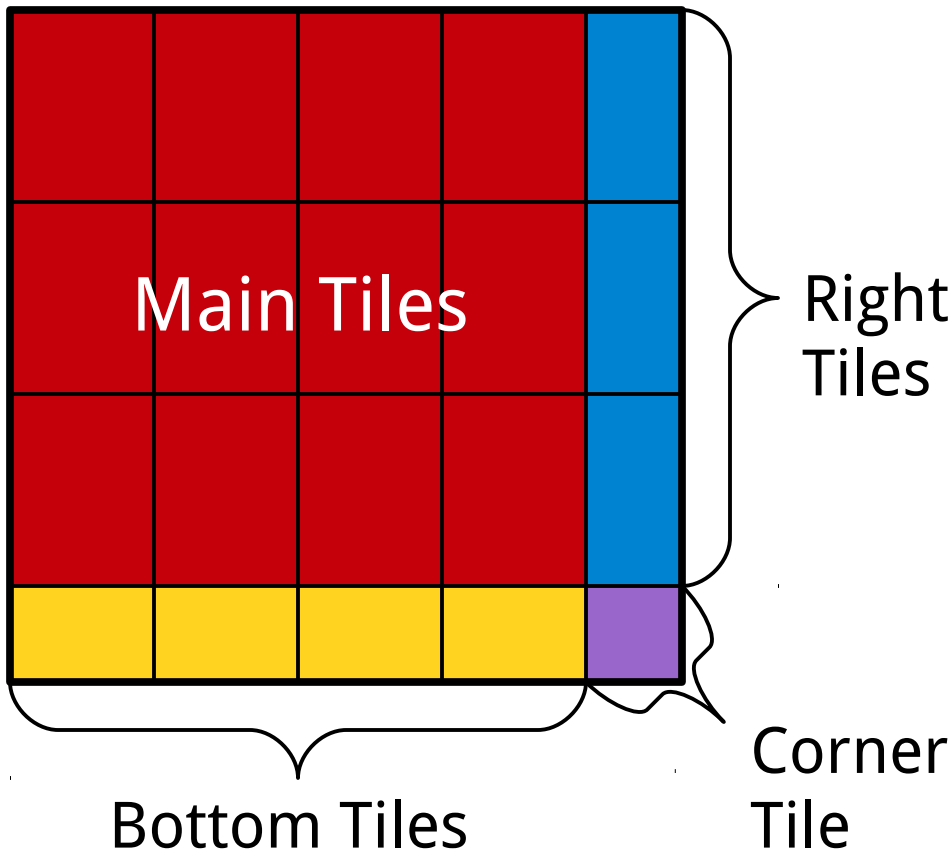
CUDA Implementation

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

- Multiple pass implementation
  - Average, denominator, and numerator similar
- Outer loops are all addition

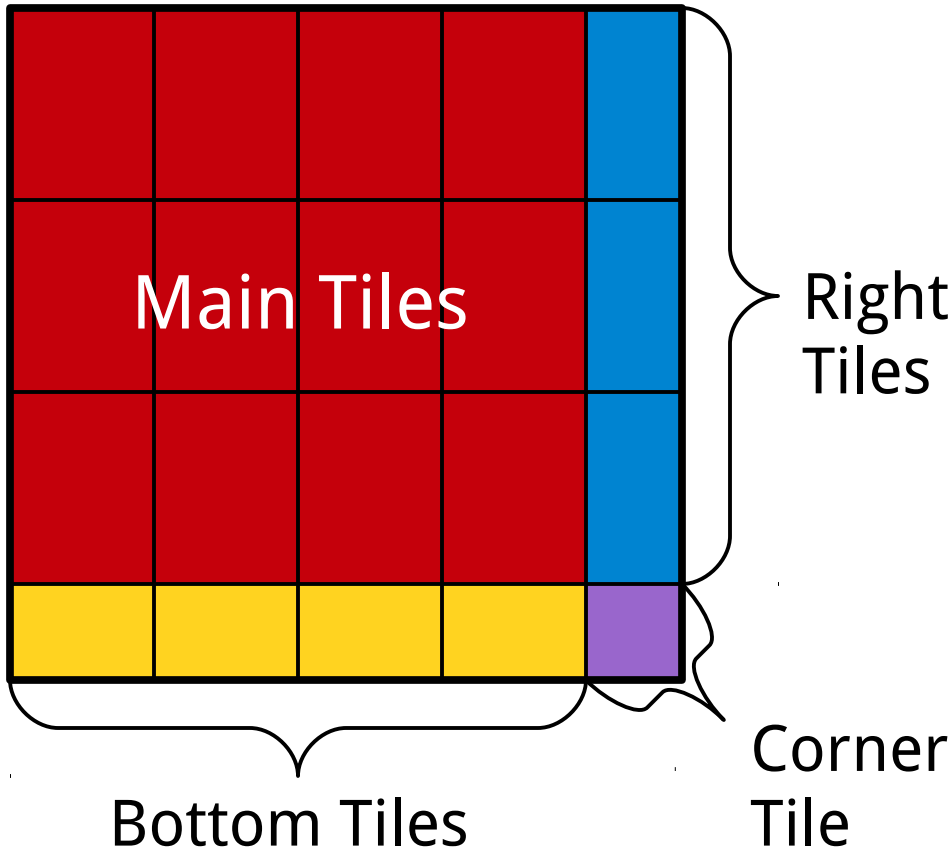
Tiled Template (1)

- 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

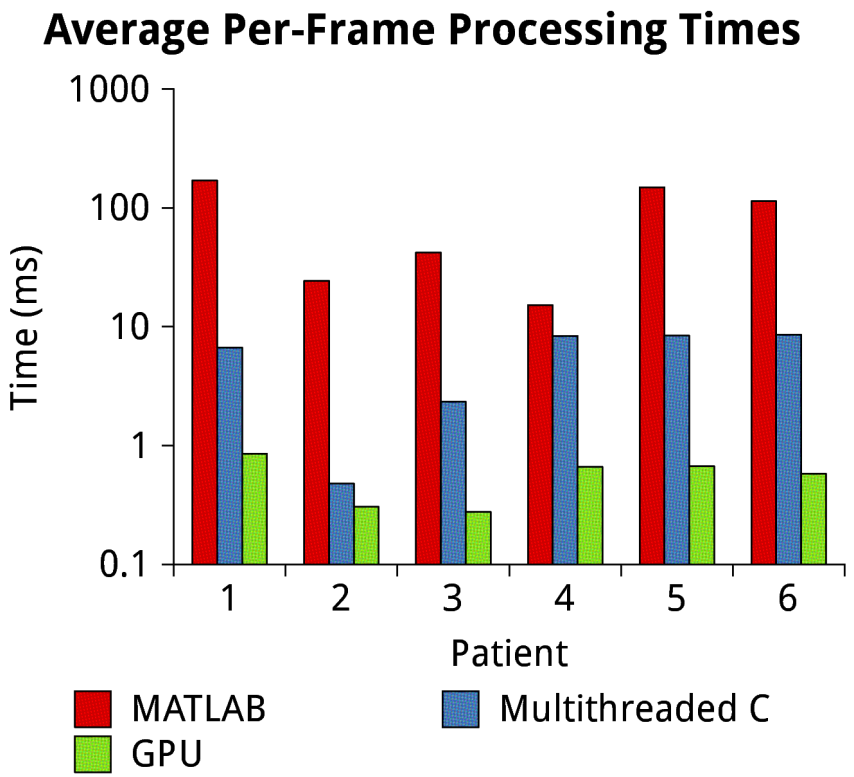


Experimental Setup

- Benchmarked tile sizes from 4×4 to 16×16
- 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:

Patient	1	2	3	4	5	6
Template Size	53×54	23×21	76×45	156×116	86×78	141×107
Best Tile Size	16×2	4×4	8×8	16×10	8×8	16×10
Total Speedup	7.80	1.57	8.48	12.67	12.50	14.78

Tile Size Selection (1)

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

Patient	1	2	3	4	5	6
Template Size	53×54	23×21	76×45	156×116	86×78	141×107
Best Tile Size	16×2	4×4	8×8	16×10	8×8	16×10
Total Speedup	7.80	1.57	8.48	12.67	12.50	14.78

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

Patient	1	2	3	4	5	6
Template Size	53×54	23×21	76×45	156×116	86×78	141×107
Best Tile Size	16×2	4×4	8×8	16×10	8×8	16×10
Total Speedup	7.80	1.57	8.48	12.67	12.50	14.78

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

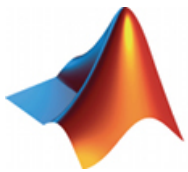
Thank You

Nicholas Moore: nmoore@coe.neu.edu

Miriam Leeser: mel@coe.neu.edu

Northeastern

Supported by



MathWorks®

Performance Breakdown