

# Program Optimization Space Pruning for a Multithreaded GPU

Shane Ryoo, Christopher I. Rodrigues,  
Sam S. Stone, Sara S. Baghsorkhi, Sain-Zee Ueng,  
John A. Stratton, and Wen-mei W. Hwu

Center for Reliable and High-Performance Computing  
University of Illinois at Urbana-Champaign



# The GeForce 8800 GTX GPU

- 389 peak GFLOPS
- Various local memories
- 768 MB DRAM with 86.4 GB/s bandwidth
- 8GB/s PCIe
- More general programming model
- Are programs getting maximum performance from this platform?

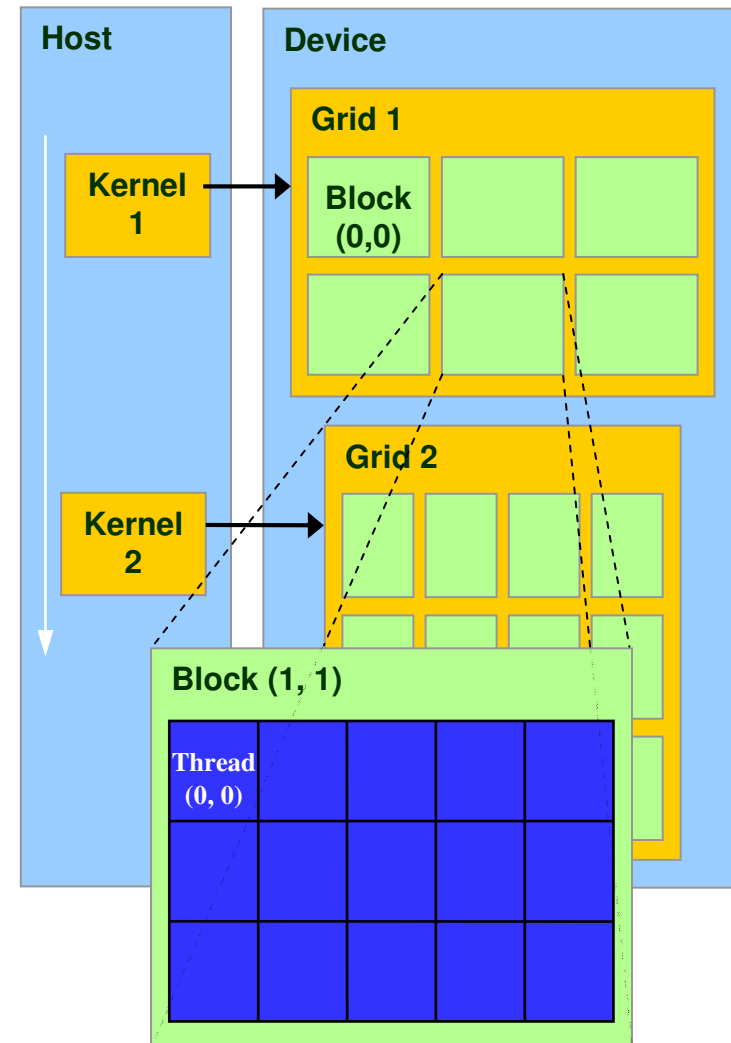


# Agenda

- CUDA Threading Model and Architecture
- Optimization
- Pruning via Pareto-Optimality
- Experiments
- Conclusion

# Threading Model

- Single-Program, Multiple-Data (SPMD)
- No global synchronization
- Allows random DRAM access
- Explicit memory spaces
- Indexed hierarchy of structures
  - 1 kernel = 1 **grid**
  - **Thread blocks**
  - **Threads**



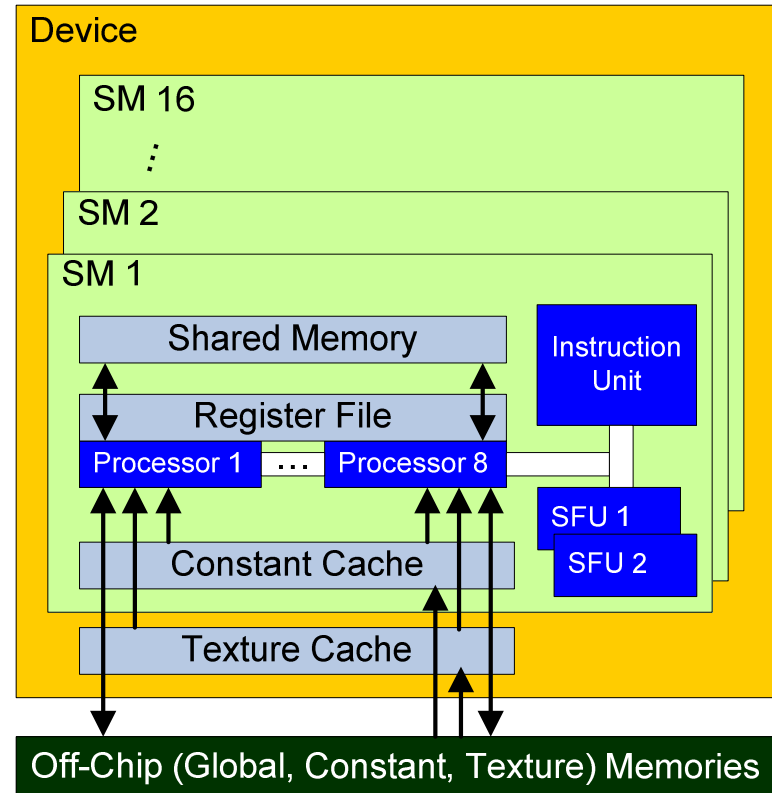
# Architecture

Consists of 16 **streaming multiprocessors (SMs)**

- 8 **streaming processors (SPs)**
- 2 **special function units (SFUs)**
- 8k registers
- Several memories and caches
- Up to 768 thread contexts with resources divided among them

An integral number of thread blocks is assigned to each SM

Instructions issue in bundles of 32 called **warps**



# Optimization

- First order principles [PPoPP '08]
  - Enough threads for TLP
  - Little or no inter-thread communication
  - Good use of on-chip memory to reduce bandwidth pressure
  - Thread grouping to optimize SIMD & memory banking
- Second order – balancing instruction stream efficiency and hardware utilization

# Examples of Optimizations

- Instruction stream efficiency
  - Loop unrolling
  - TLP granularity changes
  - Traditional optimization
- Hardware utilization
  - Prefetching
  - ILP-extraction transformations
  - Scheduling for MLP
  - Increasing TLP

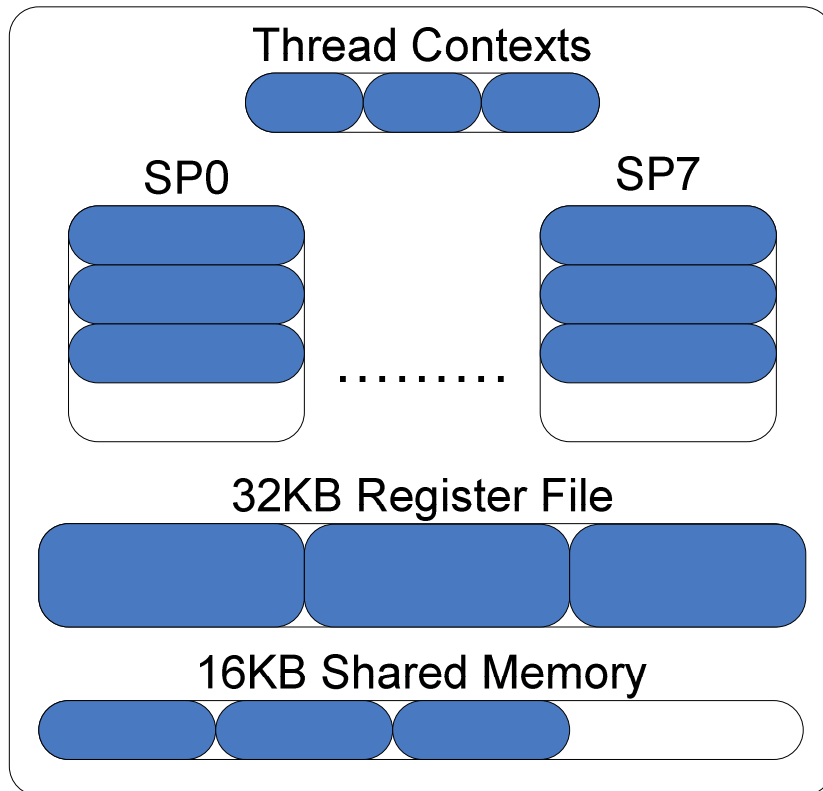
# Discontinuous Optimization Space

- Begin with a version with 3 thread blocks per SM
  - 256 threads per thread block, 10 registers per thread
  - $3 * 256 * 10 = 7680$  total registers
- Perform an optimization that uses an additional register per thread
  - $3 * 256 * 11 = 8448 > 8k!$
  - Performance per thread has increased
  - But this version has only 2 thread blocks per SM
- Is the optimized version better or worse?

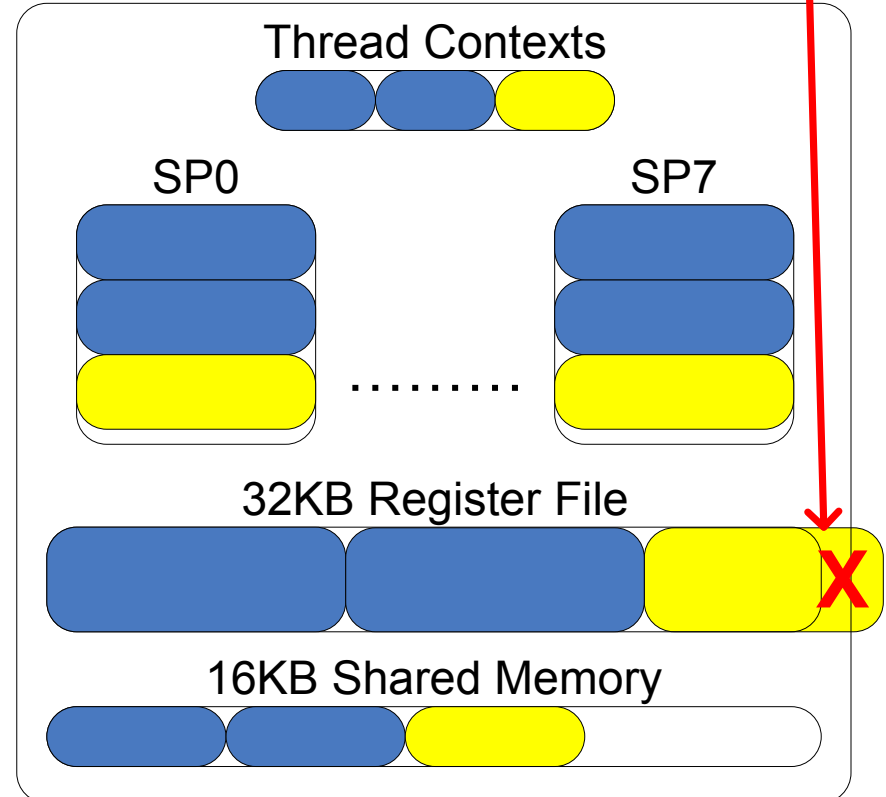
# Resource Allocation Example

TB0 TB1 TB2

Insufficient  
registers to allocate  
3 blocks



(a) Pre-“optimization”



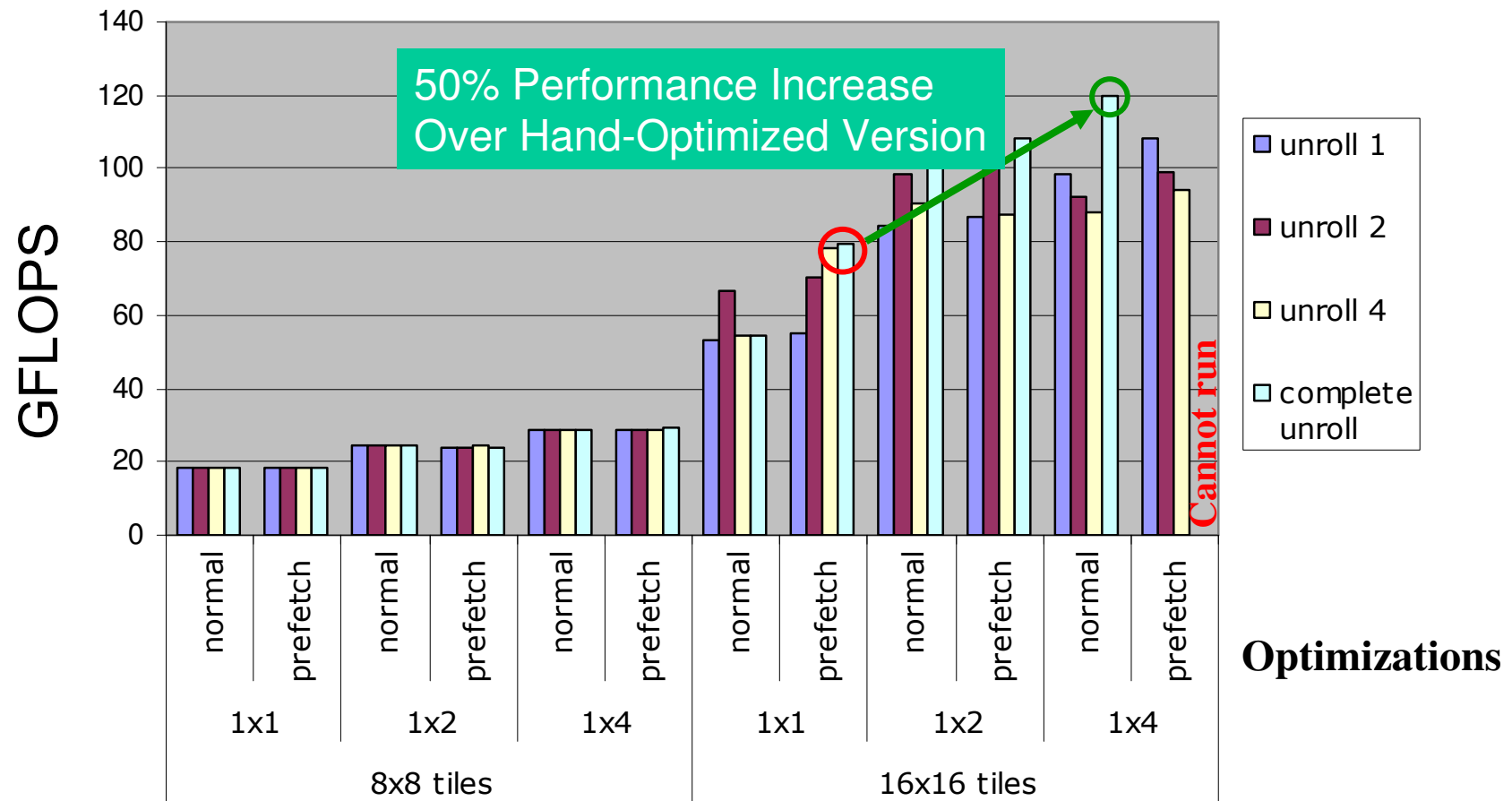
(b) Post-“optimization”

Increase in per-thread performance, but fewer threads:  
Lower overall performance in this case

# How Close Are We to Best Performance?

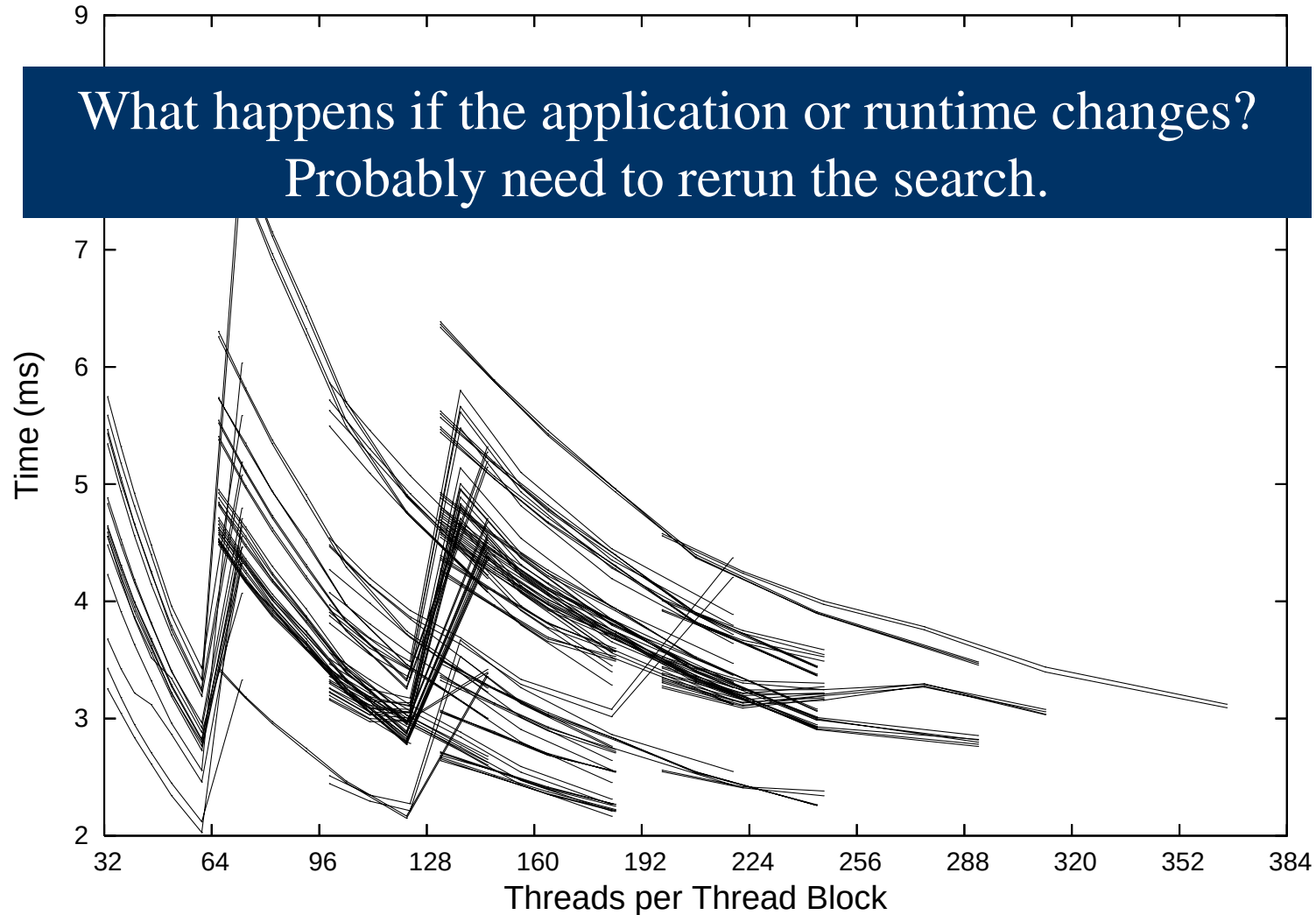
- Investigated applications with many optimizations
- Exhaustive optimization space search
  - Applied many different, controllable optimizations
  - Parameterized code by hand
- Hand-optimized code is deficient
  - Generally >15% from the best configuration
  - Trapped at local maxima
  - Often non-intuitive mix of optimizations

# Matrix Multiplication Space



# SAD Thread Block Size

What happens if the application or runtime changes?  
Probably need to rerun the search.



# Solution: Evaluate All Likely Configurations

- Begin with a complete, parameterized optimization space per kernel
  - Tile size and shape
  - Thread block size, work per thread block
  - Other optimizations, such as manual register spilling
- Compile all configurations to find resource usage
- Generate metric values for each configuration
- Prune space using a Pareto-optimal curve

# Metrics for the GeForce 8800

How efficient is our instruction stream?

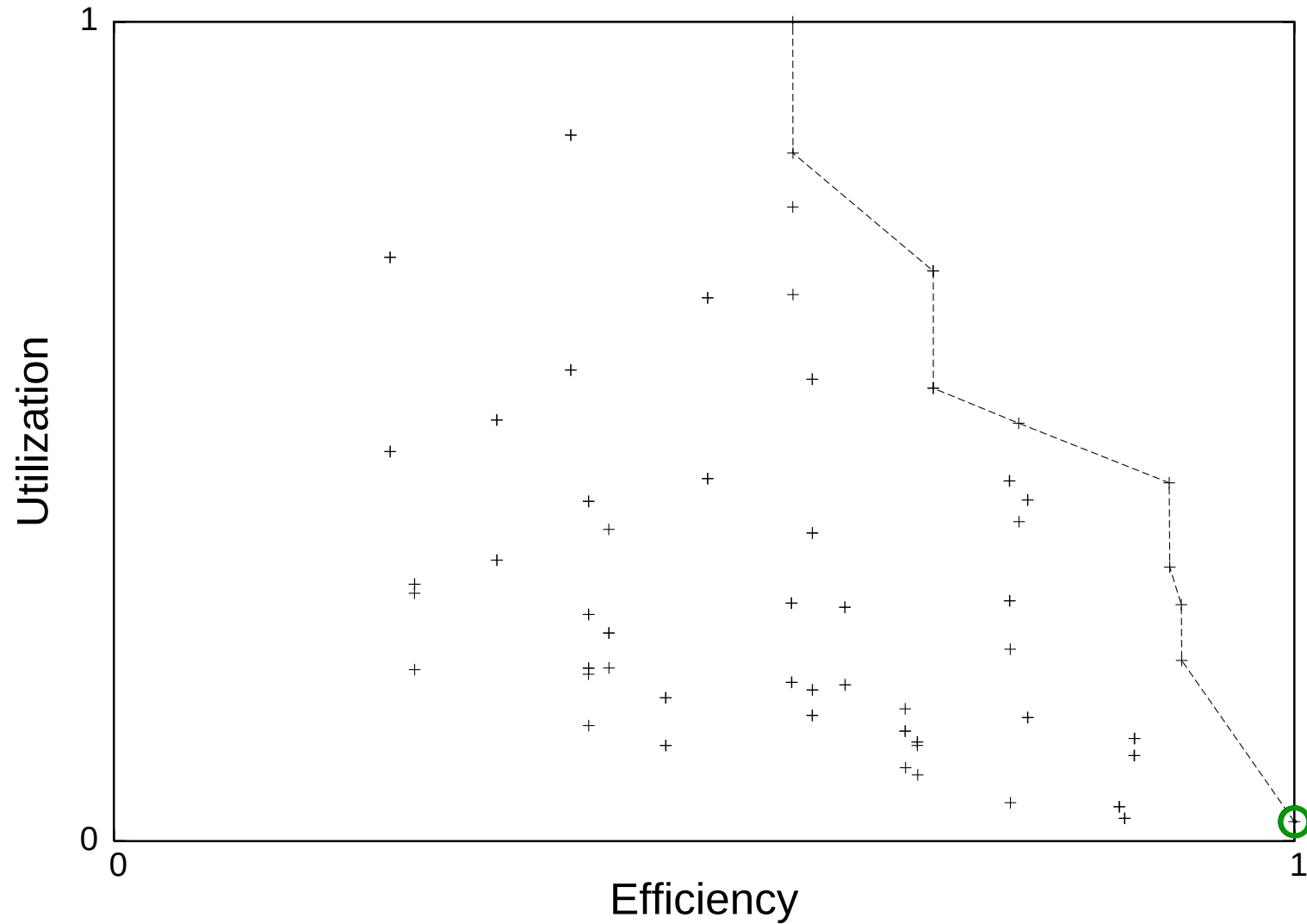
$$Efficiency = \frac{1}{Instr * Threads}$$

Can we keep the processors utilized?

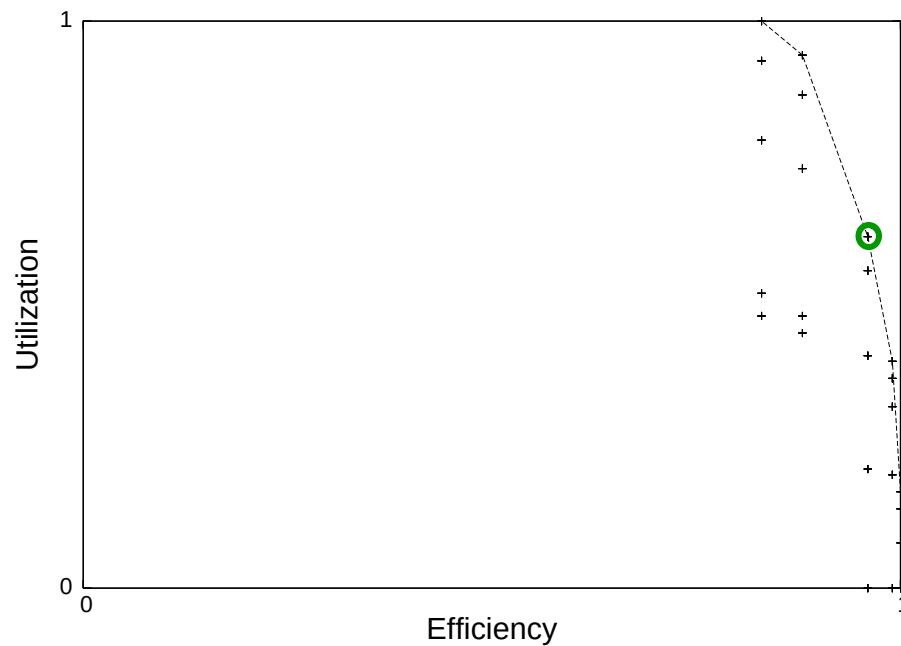
$$Utilization = \frac{Instr}{Regions} \left[ \frac{W_{TB} - 1}{2} + (B_{SM} - 1)(W_{TB}) \right]$$

These capture first-order effects of applications.

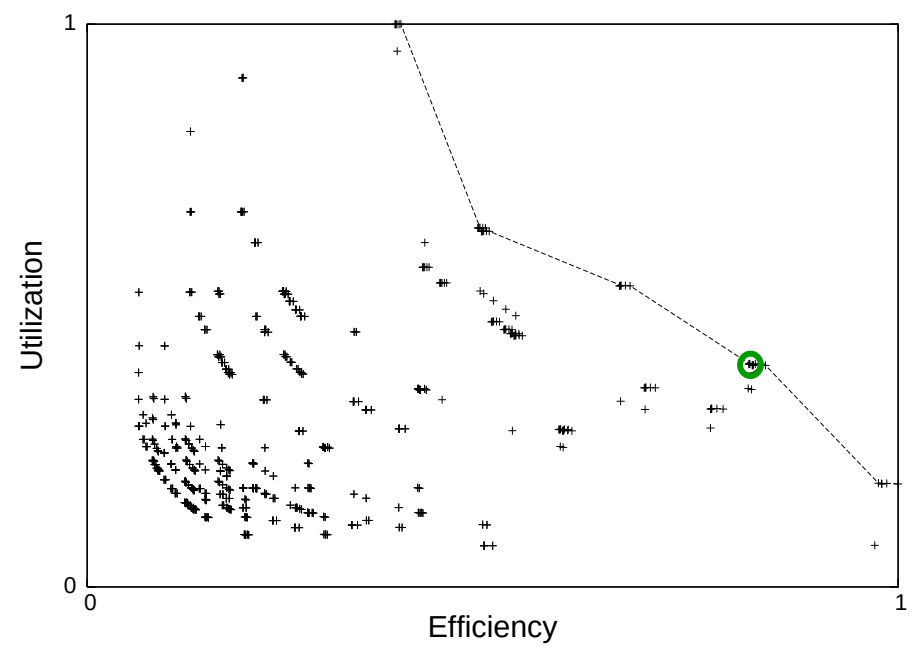
# Matrix Multiplication Pareto Plot



# MRI-FHD and SAD Plots



**MRI-FHD**



**SAD**

# Space Reduction

| Kernel  | Total Configs | Total Eval | Selected Configs | Space Reduc. | Time Reduc. | Selected Perf. |
|---------|---------------|------------|------------------|--------------|-------------|----------------|
| MM      | 93            | 363.3 s    | 11               | 88%          | 87%         | 100%           |
| CP      | 38            | 159.5 s    | 10               | 74%          | 73%         | 100%           |
| SAD     | 908           | 7.677 s    | 16               | 98%          | 98%         | 100%           |
| MRI-FHD | 175           | 771.9 s    | 30               | 77%          | 73%         | 100%           |

## Future Work

- Adaptation to more complex applications and behavior
- Developing a single cost function for performance
- Pruning prior to compilation
- Targeting of different parallelism granularities
- Visualization tools for performance tuning

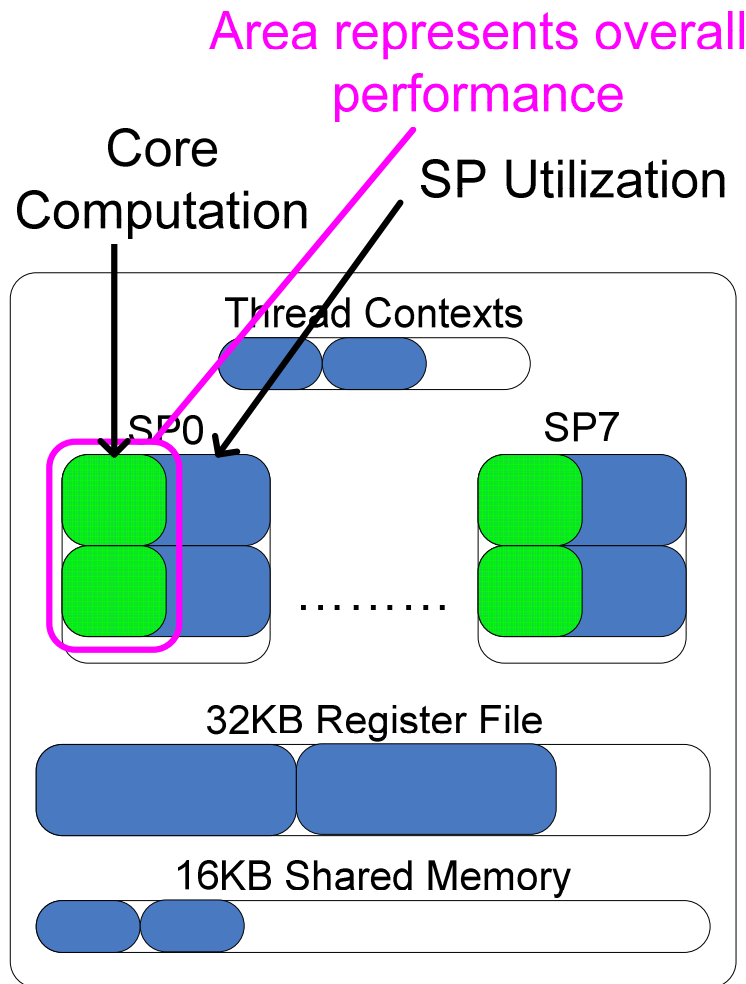
# Summary and Conclusion

- Performance tuning involves tradeoffs between instruction efficiency and HW utilization
- Manual, iterative optimization usually leaves performance on the table
- Reoptimization needed for new hardware, application, or runtime
- Pareto-based pruning removes up to 98% of the optimization space, still finds best configuration
- Ryoo Ph.D. dissertation contains further work:  
<http://www.crhc.uiuc.edu/IMPACT>.

# Questions?

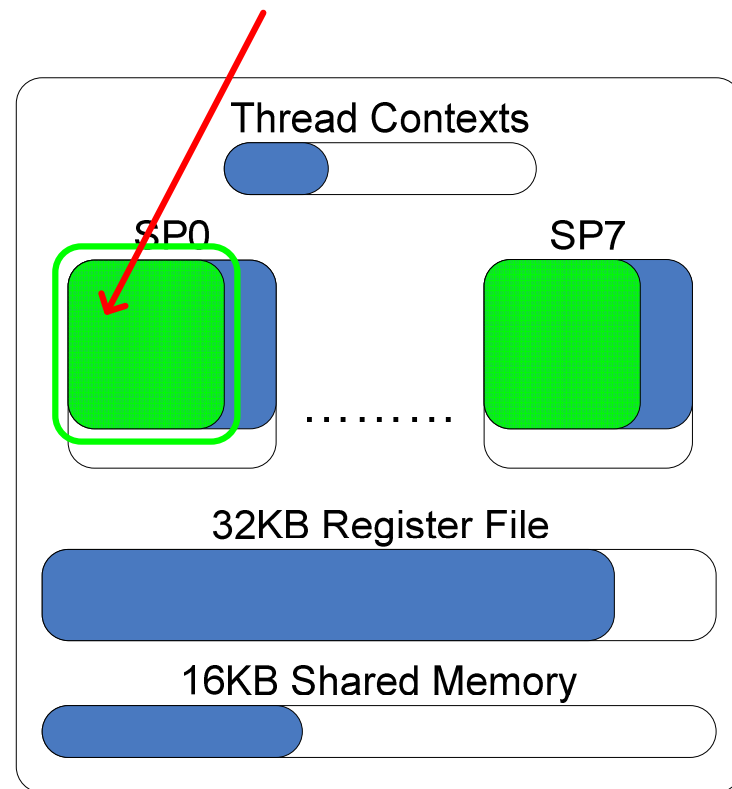
# Backup Slides

# Matrix Multiplication Resource Usage



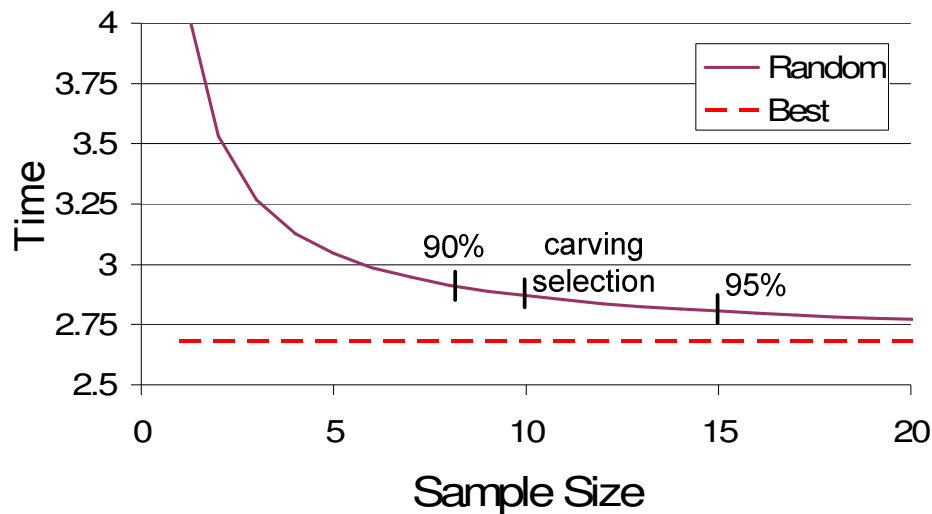
(a) Hand-Optimized Configuration

Better code efficiency with equivalent SP utilization and fewer threads

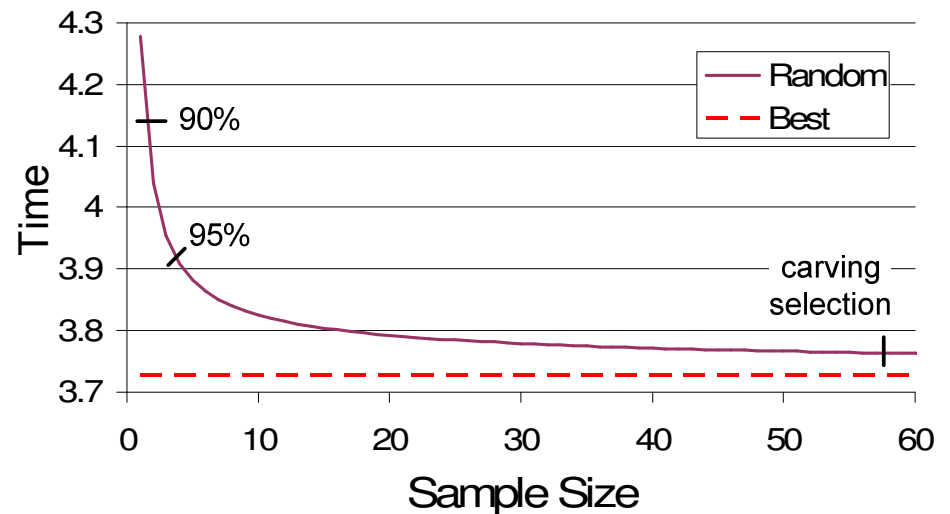


(b) Best Configuration from Exploration

# Evaluation: Random Sampling



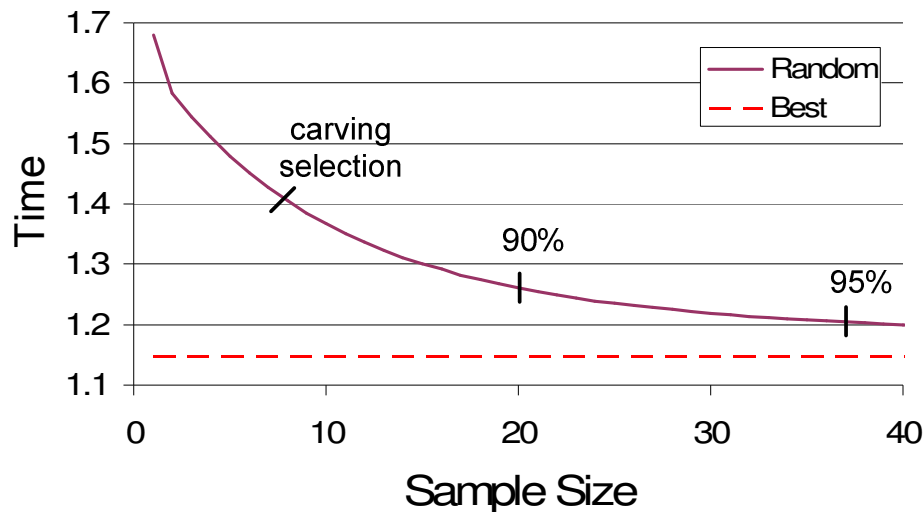
**CP**



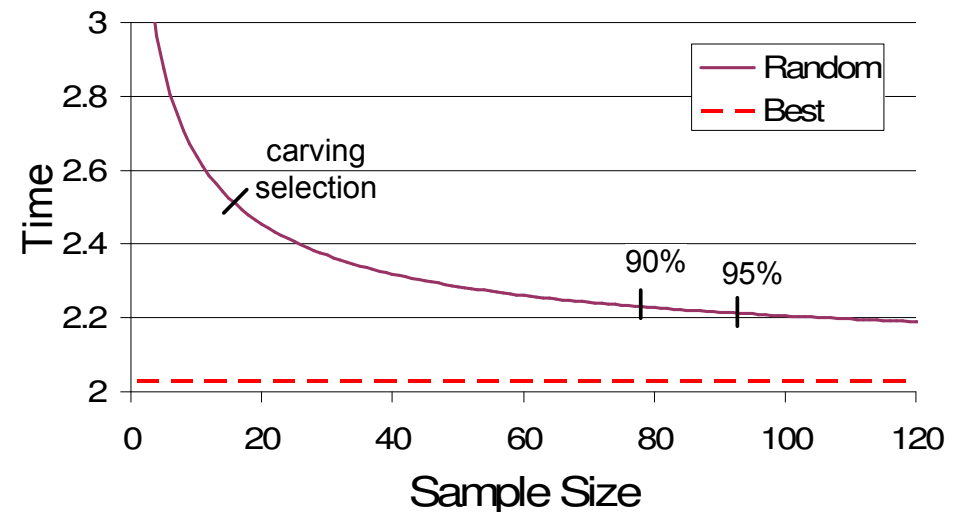
**MRI-FHD**

Random sampling is effective when there are a large number of high-performance configurations

# Random Sampling on Multi-Tiled Algorithms



**MM (16x16)**



**SAD**

- These kernels have multiple dimensions of tiling
- Best configuration requires hitting the right combination of optimizations