



## Accelerating Dense Linear Algebra on the GPU

**Sørensen, Hans Henrik Brandenborg**

*Publication date:*  
2011

*Document Version*  
Publisher's PDF, also known as Version of record

[Link back to DTU Orbit](#)

### Citation (APA):

Sørensen, H. H. B. (Invited author). (2011). Accelerating Dense Linear Algebra on the GPU. Sound/Visual production (digital) <http://cg.alexandra.dk/2011/11/24/accelerating-computations-a-research-conference-on-graphics-processing-units-visual-computing-and-beyond/>

---

### General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

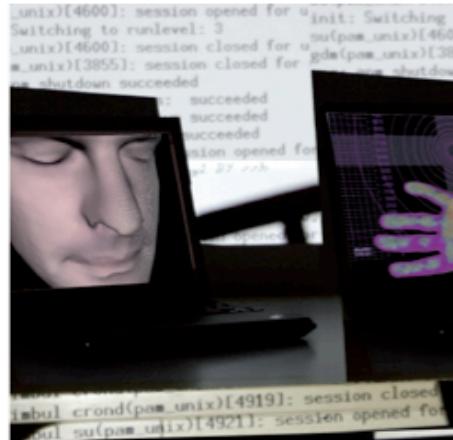
- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

# Accelerating Dense Linear Algebra on the GPU

Hans Henrik Brandenborg Sørensen, Postdoc  
Section for Scientific Computing  
DTU Informatics

# GPU-LABoratory



## Research and education in Graphics Processing Units in Denmark

Established in August 2008 and is a unique national competence center and hardware laboratory.

- Development of efficient algorithms
- High-performance scientific computing
- Performance profiling and prediction
- Software development
- Education

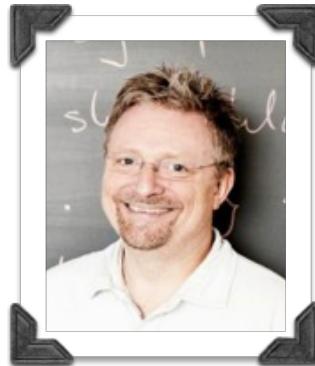
<http://gpulab.imm.dtu.dk>

With support 2010-2013 from national FTP grant  
“Desktop Computing on Consumer Graphics Cards”  
PI: Prof. Per Christian Hansen

# GPUlab

DTU Informatics

## Facebook



Prof.  
Per C. Hansen



Assoc. Prof.  
Jeppe R. Frisvad



Assoc. Prof.  
Bernd Dammann



Assoc. Prof.  
John B. Jørgensen



Postdoc  
Hans-Henrik B. Sørensen



PhD Student  
Stefan L. Glimberg



PhD Student  
Nicolai Fog Gade-Nielsen



Assoc. Prof.  
Allan P. Engsig-Karup

# Educational activities



## Topics:

- Parallel programming
- High-Performance Computing
- Algorithms and Numerical Methods for Solving Differential Equations

~200 participants with approx. 150 with research background since May 2010

## 2010

- Ph.D. School on “Scientific GPU Computing” with Hendrik Lensch (Germany), Robert Strzodka (Germany) and Timothy Lanfear (Nvidia)

## 2011

- One week course on “Introduction to parallel programming of GPUs using CUDA” in DTU course 02614 on “High-Performance Computing”
- Ph.D. School in “Scientific GPU Computing” with Tim Warburton (Rice University, USA)
- Ph.D. School in “Iterative Methods for Large Linear Systems” with Tim Kelley (North Carolina University)
- Workshop on “Easy, Effective, Efficient: GPU Programming in Python with PyOpenCL and PyCUDA” with Andreas Kloeckner (New York University, USA)

# Outline

1. Motivation
2. Identifying tuning parameters / kernel designs
3. Performance prediction
4. Auto-tuning results

# Motivation

# Motivation

- Running time of dense linear algebra depends on;
  - # flops \* time per flop
  - # bytes moved / bandwidth
  - # messages \* latency

# Motivation

- Running time of dense linear algebra depends on;

- # flops \* time per flop Compute bound
- # bytes moved / bandwidth
- # messages \* latency Memory bound

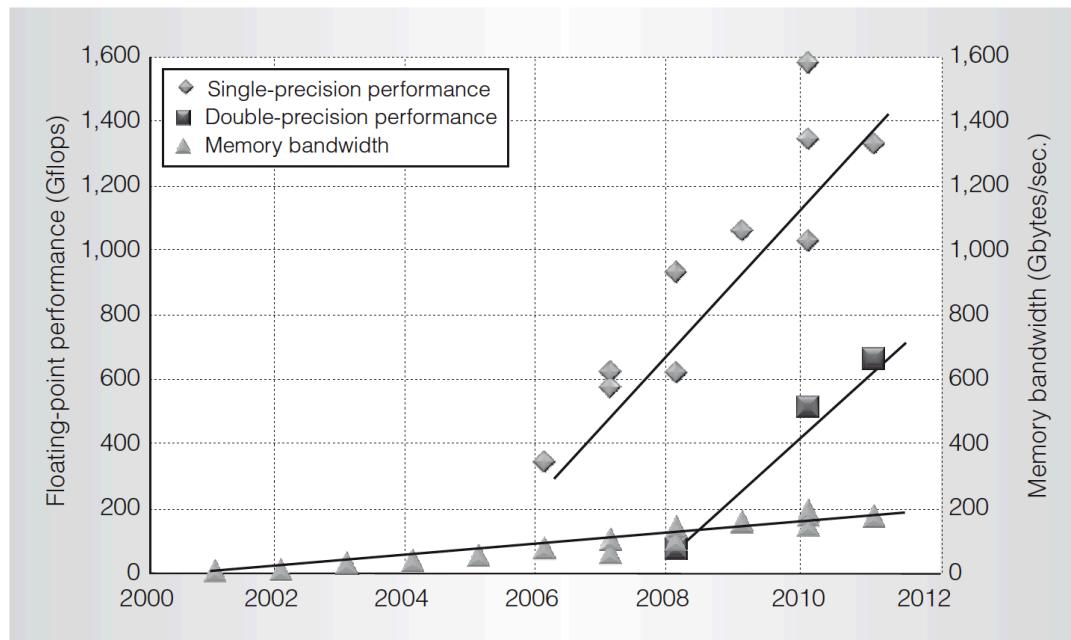
# Motivation

- Running time of dense linear algebra depends on;

- # flops \* time per flop
- # bytes moved / bandwidth
- # messages \* latency

Compute bound

Memory bound



Source: Keckler et al, "GPUs and the future of parallel computing", Nvidia

## WANTED

- **High memory-bandwidth**  
(good for memory-bound algorithms)
- **Massive multi-threading capability**  
(good latency-hiding)
- **Many-core processing**  
(good for massively parallel processing)
- **Programmable with standard tools**  
(good for productivity)
- **Mass-produced commodity hardware**  
(widely available and affordable)

# High-Performance Computing on GPUs



PRESENTED BY  
UNIVERSITY OF  
MANNHEIM



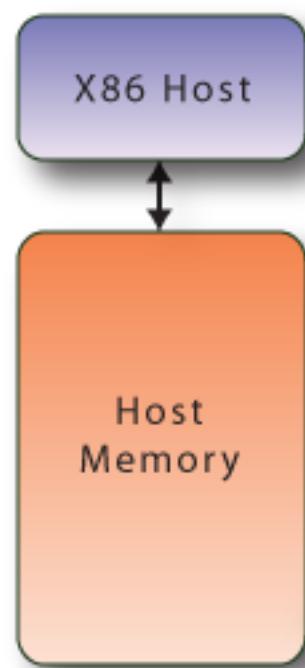
Lawrence Berkeley  
National Laboratory

FIND OUT MORE AT  
[www.top500.org](http://www.top500.org)

	NAME/MANUFACTURER/COMPUTER	SITE	COUNTRY	CORES	R <sub>max</sub> Pflop/s
1	K computer SPARC64 VIIIfx 2.0GHz, Tofu interconnect	RIKEN	Japan	705,024	10.5
2	Tianhe-1A 6-core Intel X5670 2.93 GHz + Nvidia M2050 GPU w/custom interconnect	NUDT/NSCC/Tianjin	China	186,368	2.57
3	Jaguar Cray XT-5 6-core AMD 2.6 GHz w/custom interconnect	DOE/OS/ORNL	USA	224,162	1.76
4	Nebulae Dawning TC3600 Blade Intel X5650 2.67 GHz + NVidia Tesla C2050 GPU w/Iband	NSCS	China	120,640	1.27
5	Tsubame 2.0 HP Proliant SL390s G7 nodes (Xeon X5670 2.93GHz), NVIDIA Tesla M2050 GPU w/Iband	TiTech	Japan	73,278	1.19

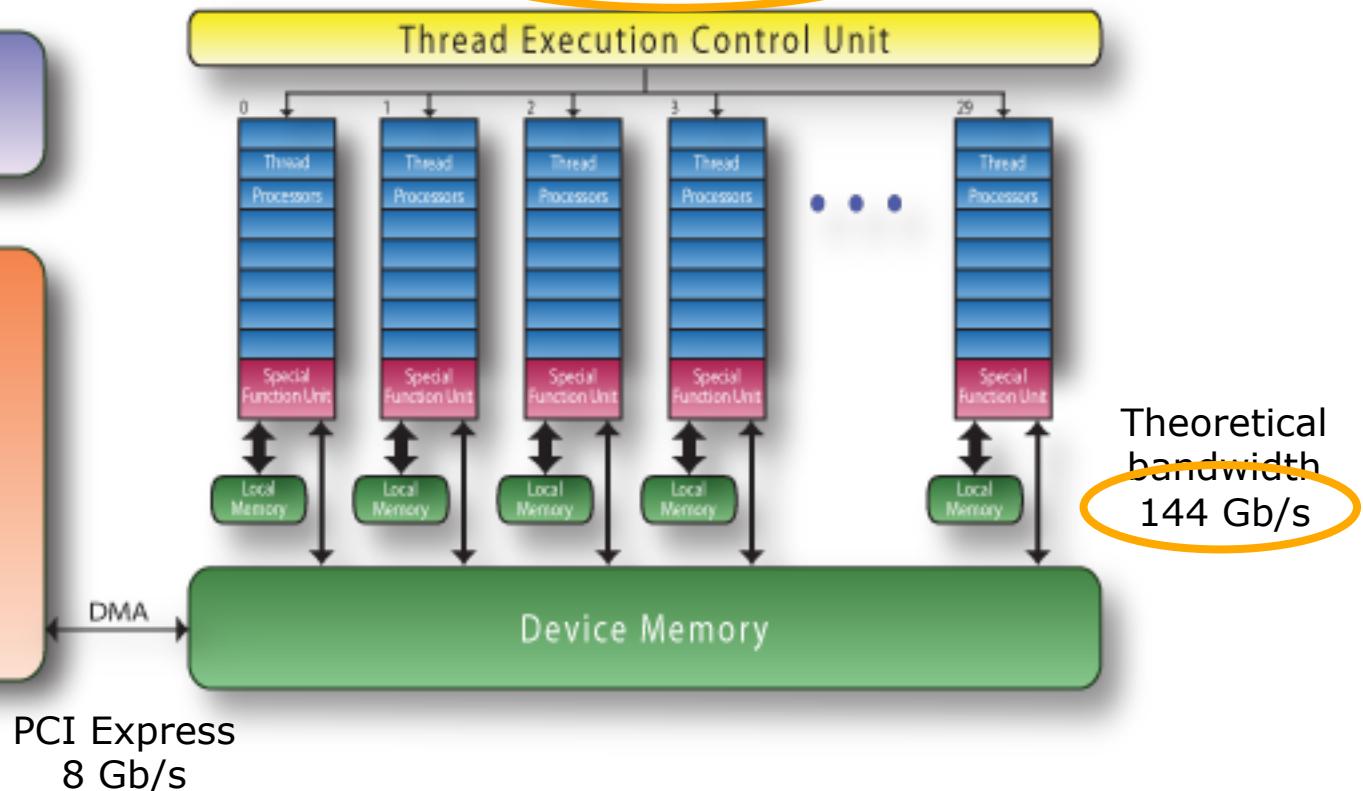
# GPU Lab hardware

**Host**  
Intel Xeon  
4 cores  
2.4 GHz  
38 Gflop/s (DP)



## Accelerator (GPU)

Nvidia C2050 "Fermi"  
448 cores  
1.15 GHz  
515 Gflop/s (DP)



Source: PGI Insider: "The PGI Accelerator Programming Model on NVIDIA GPUs. Part 1"

# Dense Linear Algebra on GPUs

- BLAS (Basic Linear Algebra Subroutines)
  - Level 1 BLAS: (xAXPY, xDOT, xNRM2, etc.)
    - Vectors of length N ( $4 \times N$  bytes)
    - $2 \times 4 \times N$  bytes :  $O(N)$  flops
    - Memory bound

# Dense Linear Algebra on GPUs

- BLAS (Basic Linear Algebra Subroutines)
  - Level 1 BLAS: (xAXPY, xDOT, xNRM2, etc.)
    - Vectors of length N (4xN bytes)
    - $2 \times 4 \times N$  bytes :  $O(N)$  flops
    - **Memory bound**
  - Level 2 BLAS: (xGEMV, xSYMV, xTRSV, etc.)
    - Matrix of size NxN (4xNxN bytes) + Vectors
    - $4 \times N^2 + 2 \times 4 \times N$  bytes :  $O(N^2)$  flops
    - **Memory bound**

# Dense Linear Algebra on GPUs

- BLAS (Basic Linear Algebra Subroutines)
  - Level 1 BLAS: (xAXPY, xDOT, xNRM2, etc.)
    - Vectors of length N (4xN bytes)
    - $2 \times 4 \times N$  bytes :  $O(N)$  flops
    - **Memory bound**
  - Level 2 BLAS: (xGEMV, xSYMV, xTRSV, etc.)
    - Matrix of size NxN (4xNxN bytes) + Vectors
    - $4 \times N^2 + 2 \times 4 \times N$  bytes :  $O(N^2)$  flops
    - **Memory bound**
  - Level 3 BLAS: (xGEMM, xSYMM, xTRSM, etc.)
    - 1 or 2 matrices of size NxN (4xNxN bytes)
    - $(2 \times) 4 \times N^2$  bytes :  $O(N^3)$  flops
    - **Compute bound** for large N

# Dense Linear Algebra on GPUs

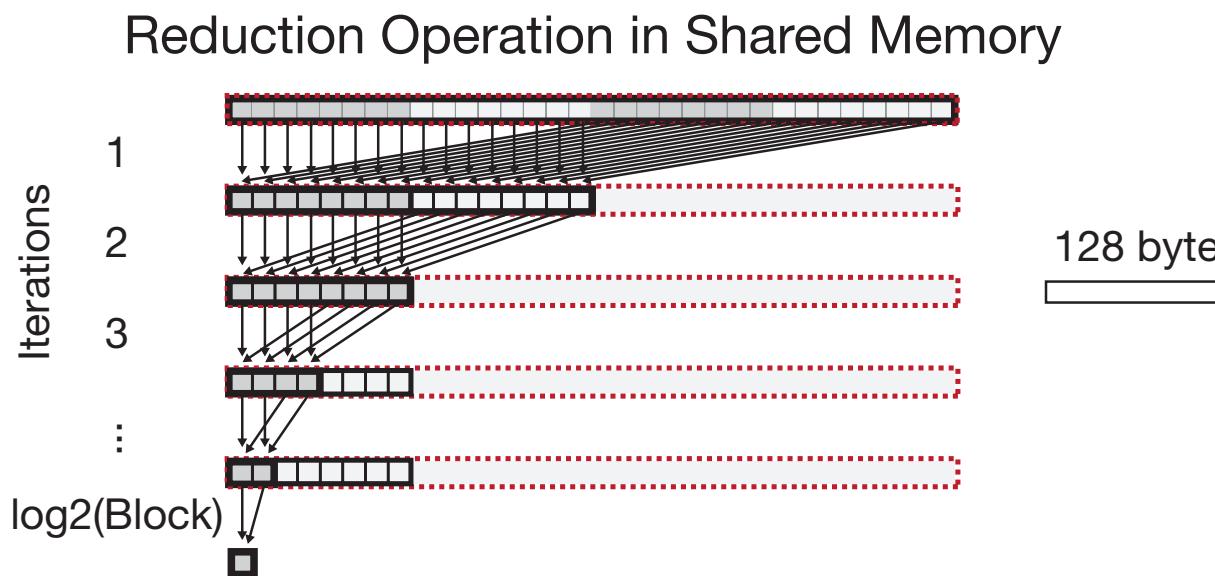
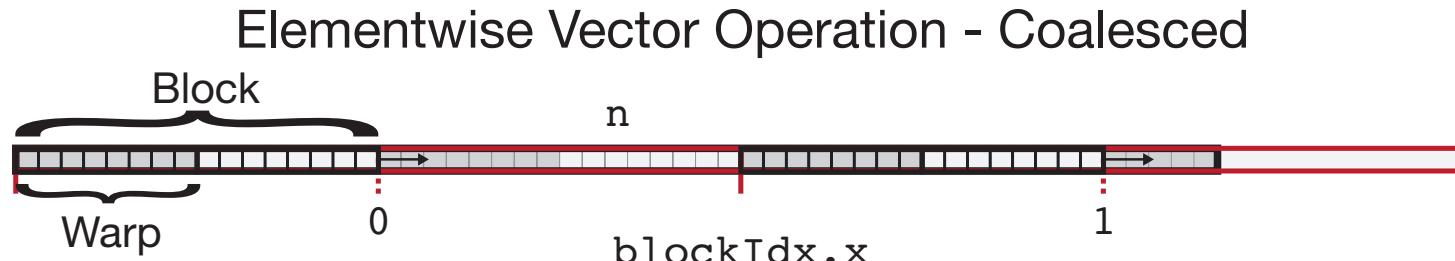
Tesla C2050 (Fermi): 448 CUDA cores @ 1.15GHz; theoretical SP peak=1.03 TFlop/s

- Level 1 BLAS: (xAXPY, xDOT, xNRM2, etc.)
  - Vectors of length N (4xN bytes)
  - $2 \times 4 \times N$  bytes :  $O(N)$  flops
  - Memory bound
    - Performance < 144 GB/s  $\sim 72$  GFlops/s
    - Little attention from the GPU community!
- Level 2 BLAS:
  - Matrix of size NxN (4xNxN bytes) + Vectors
  - $4 \times N^2 + 2 \times 4 \times N$  bytes :  $O(N^2)$  flops
  - Memory bound
- Level 3 BLAS: (xGEMM, xSYMM, xTRSM, etc.)
  - 1 core
    - Performance > 600 GFlops/s.
  - (2) cores
    - Very much attention!
  - Compute bound for large N



# **Identifying tuning parameters / kernel designs**

# Level 1 BLAS: Vector Operations



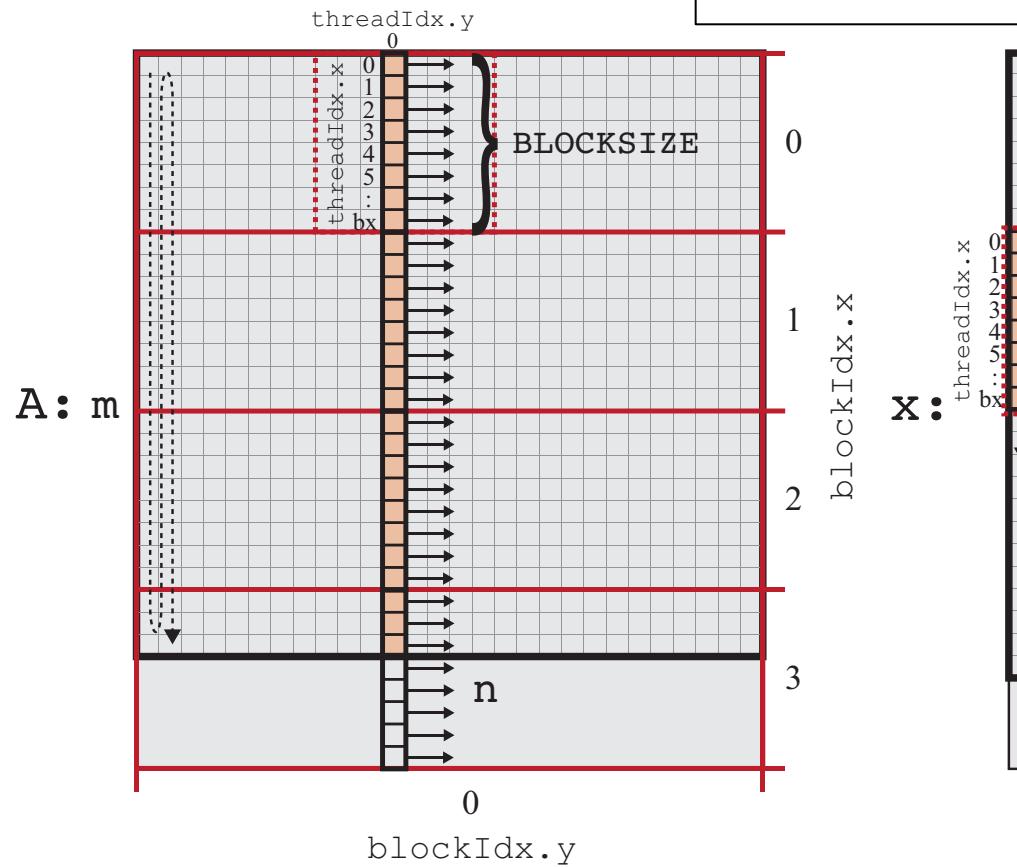
Tuning parameters: BLOCKSIZE, WORKSIZE\_n, UNROLL\_LEVEL

# Level 2 BLAS: Matrix-Vector Mult.

## MatVec: Typical $Ax=y$ algorithm

- Compute one dot product per thread
- Tuning parameter: BLOCKSIZE

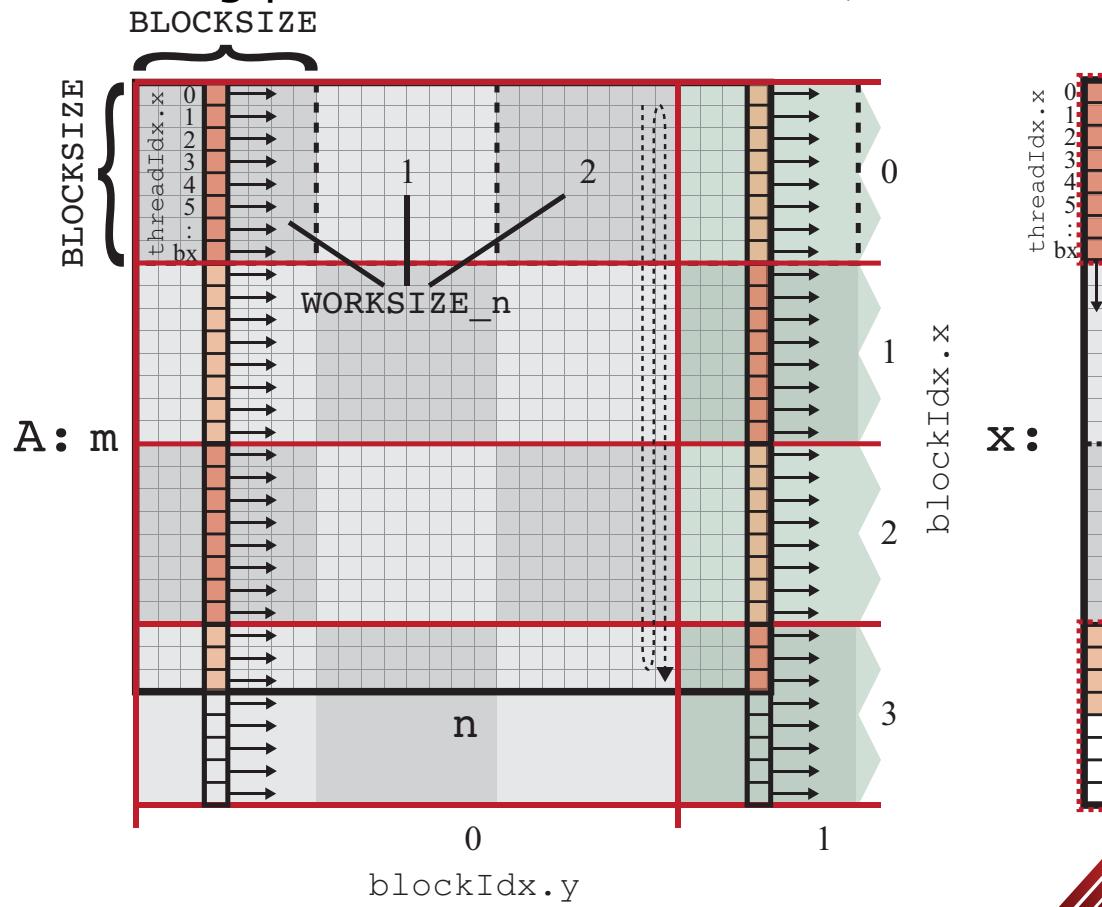
```
for i = 1:Rows
    for j = 1:Columns
        y(i) += A(i,j) * x(j);
    end
end
```



# Level 2 BLAS: Matrix-Vector Mult.

## Several threads per row

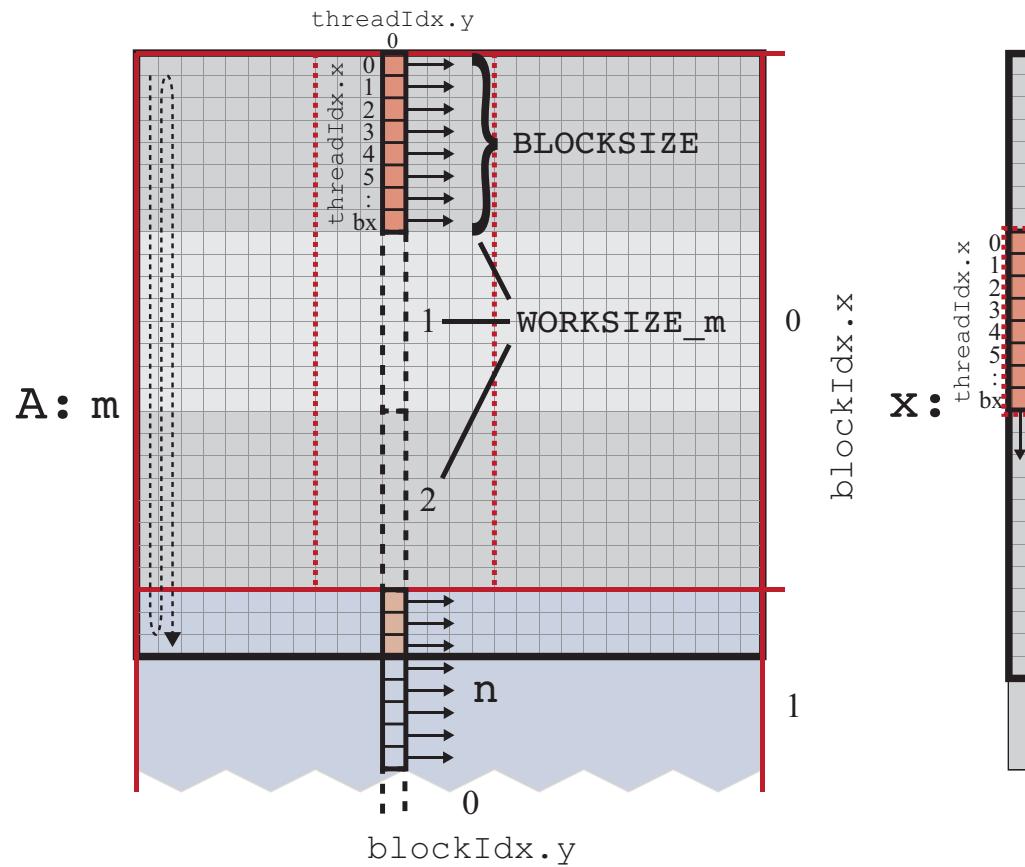
- For wide matrices; we avoid "few threads that do all the work".
- This requires tuning parameters: `BLOCKSIZE`, `WORKSIZE_n`.



# Level 2 BLAS: Matrix-Vector Mult.

## Several rows per thread

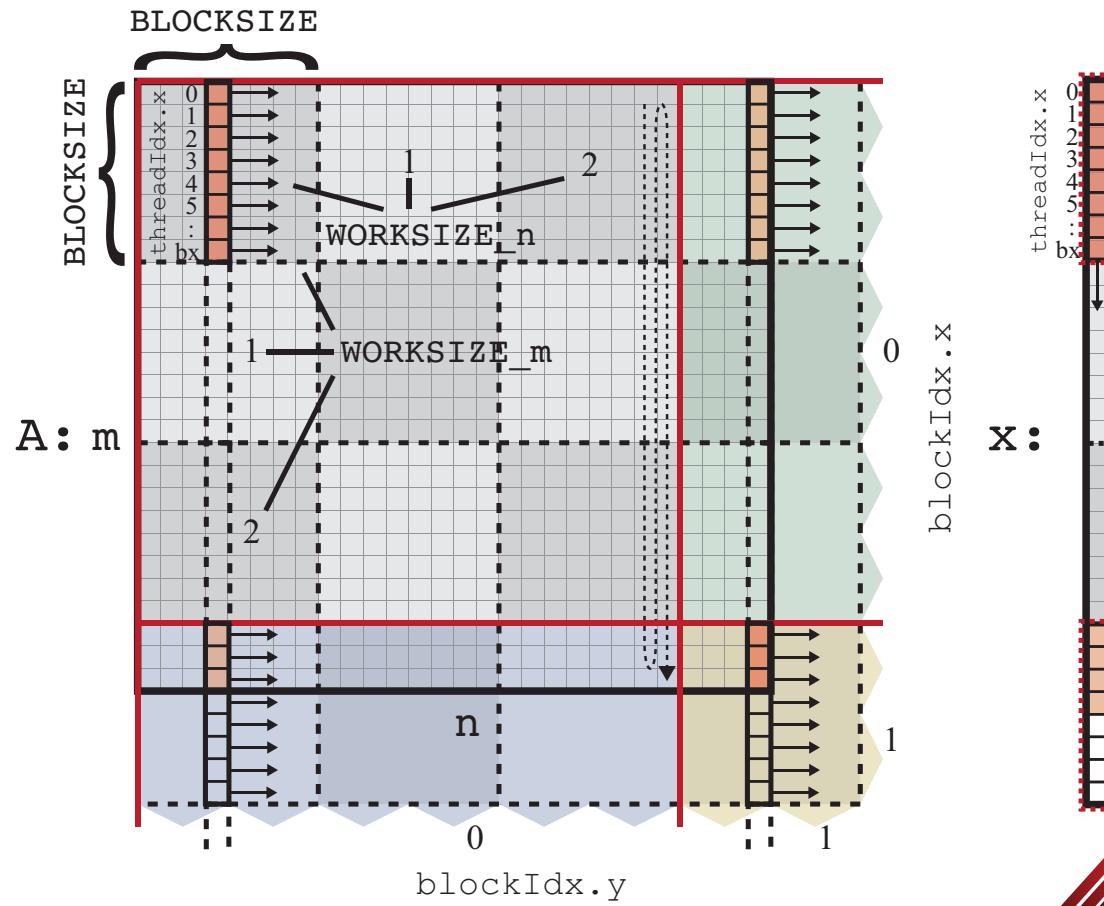
- For tall matrices; we avoid “many threads that do little work each”.
- This requires a new parameter: `WORKSIZE_m`.



# Level 2 BLAS: Matrix-Vector Mult.

## Generic kernel

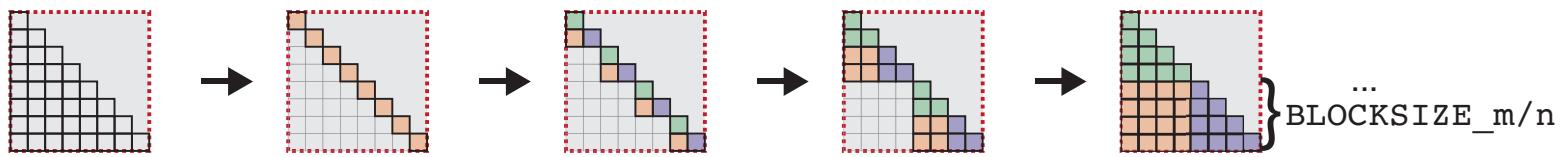
- This requires parameters: `BLOCKSIZE`, `WORKSIZE_m`, `WORKSIZE_n`.



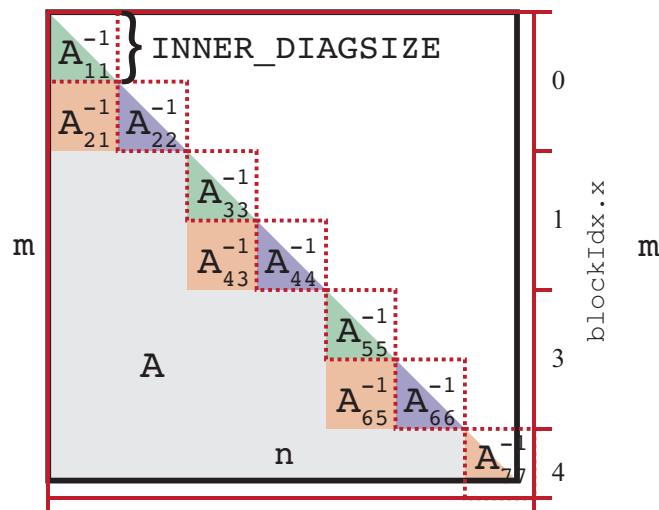
# Level 2 BLAS: Triangular Solve

$$A_{21}^{-1} = A_{22}^{-1} \times A_{21} \times A_{11}^{-1}$$

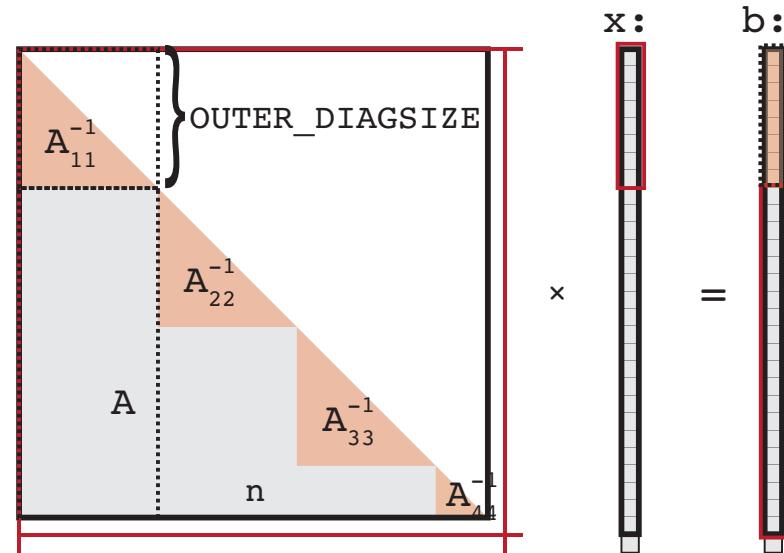
A. Triangular Matrix Inverse of Diagonal Blocks in Shared Memory



B. Expanding Triangular Matrix Inverses



C. Blocked Forward Substitution

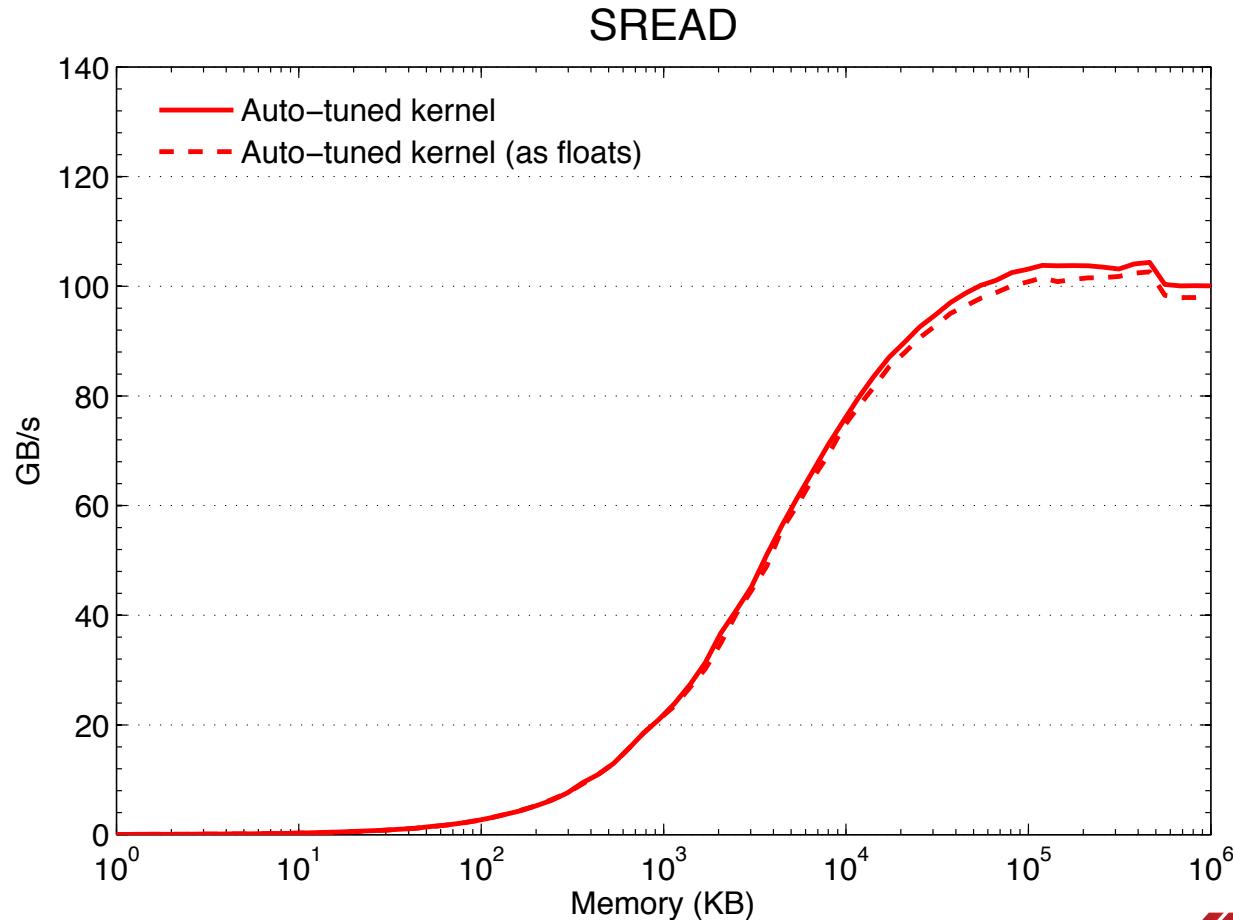


Tuning parameters: BLOCKSIZE\_m/n, INNER + OUTER\_DIAGSIZE

# Performance Prediction

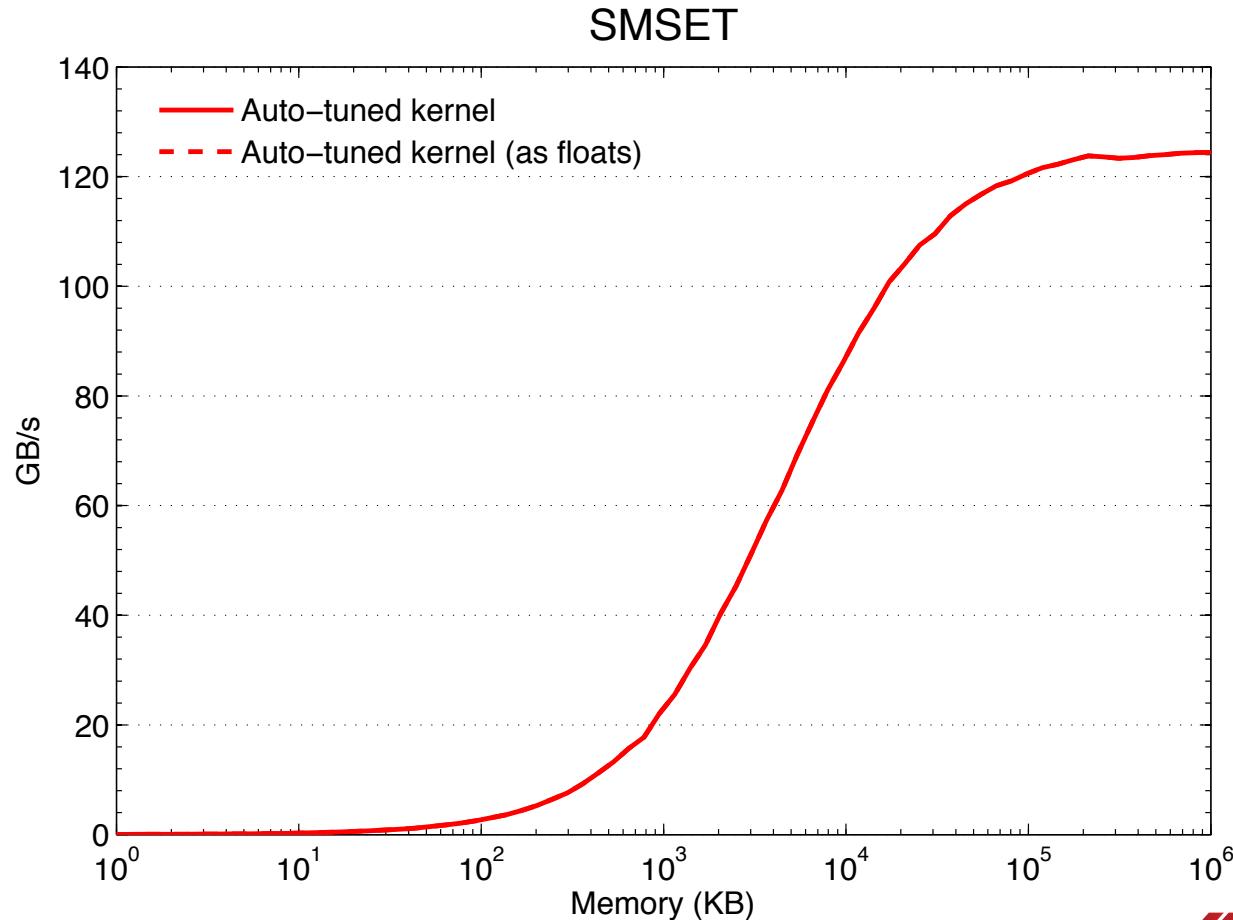
# Performance Prediction

Tesla C2050: Theoretical bandwidth 144 GB/s  
What is the effective bandwidth?



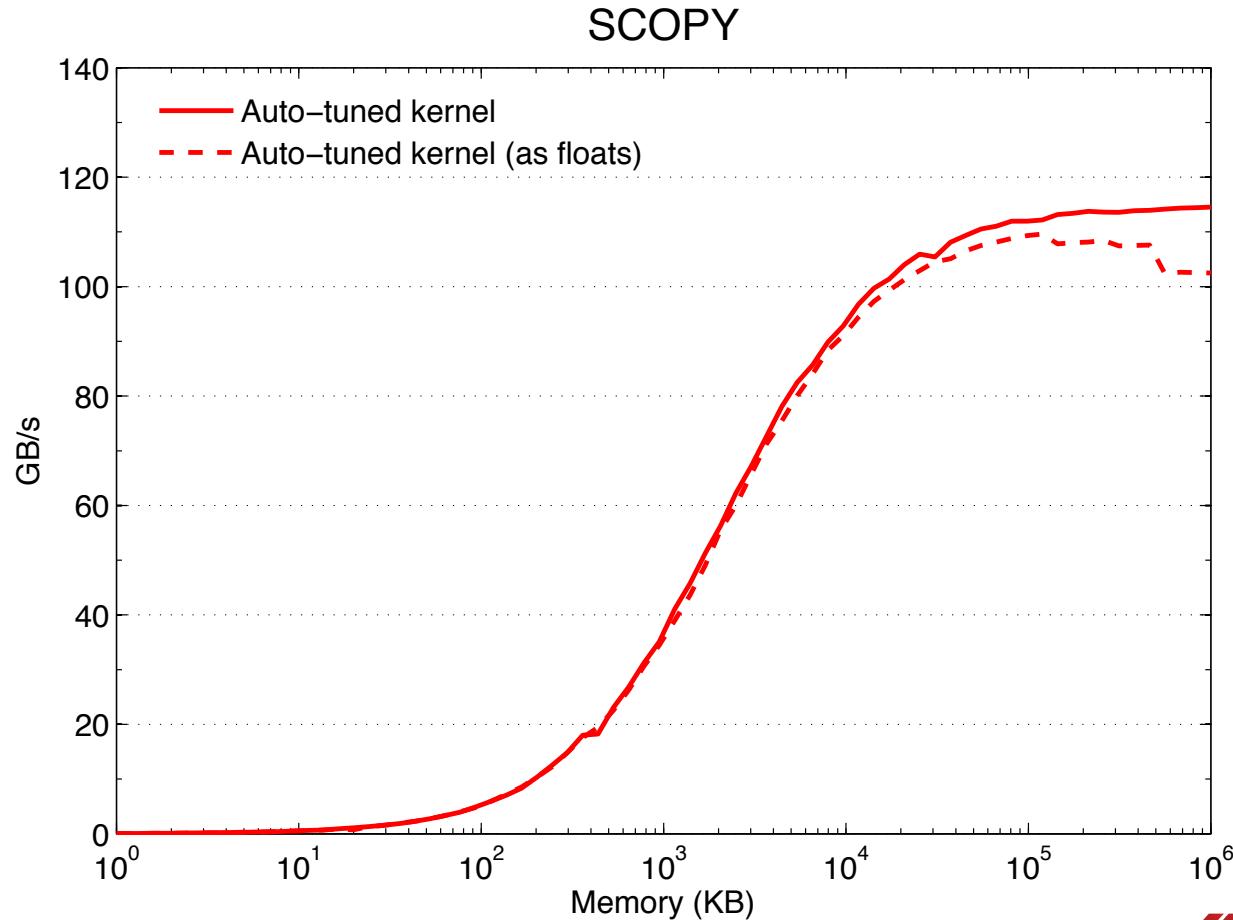
# Performance Prediction

Tesla C2050: Theoretical bandwidth 144 GB/s  
What is the effective bandwidth?



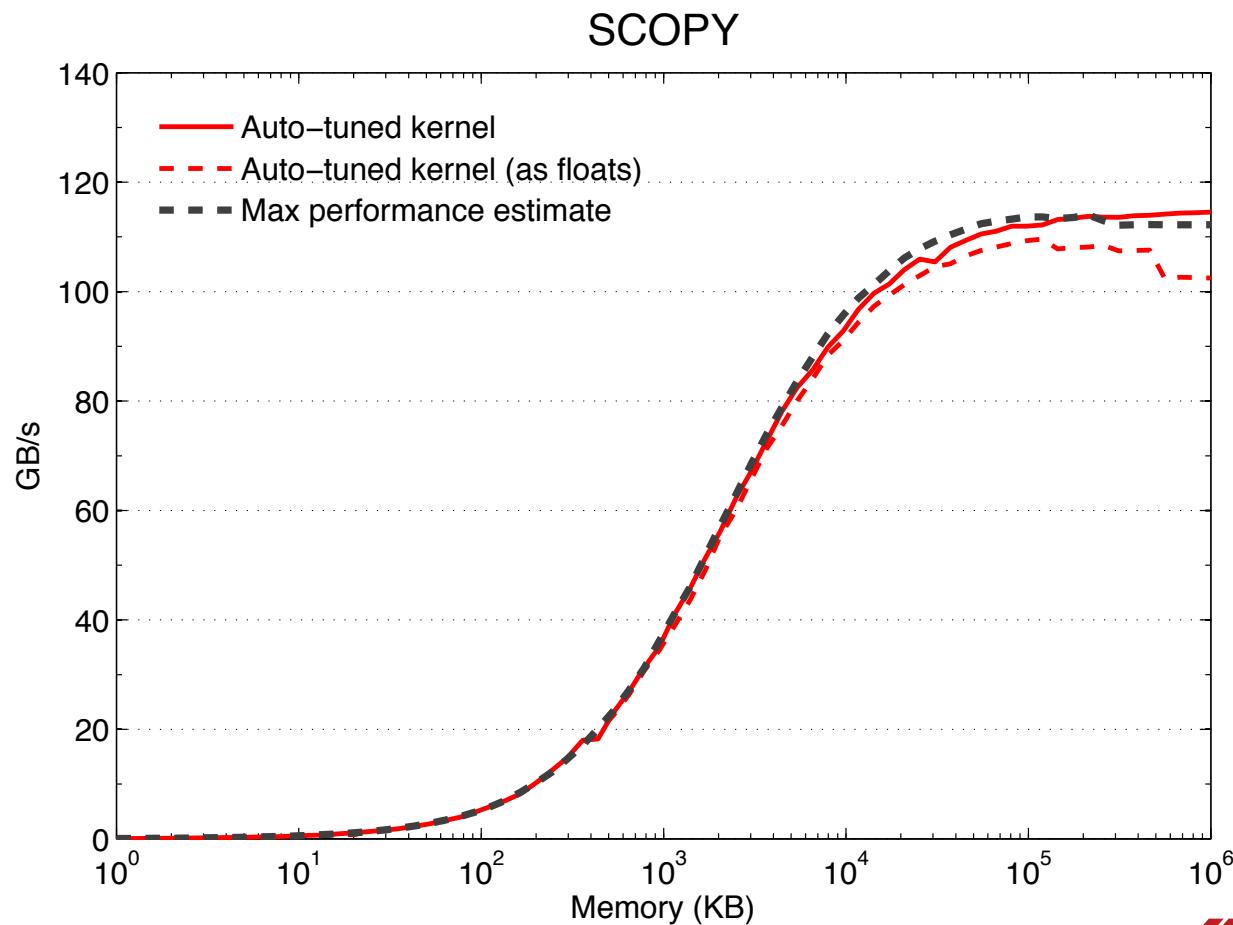
# Performance Prediction

Tesla C2050: Theoretical bandwidth 144 GB/s  
What is the effective bandwidth?



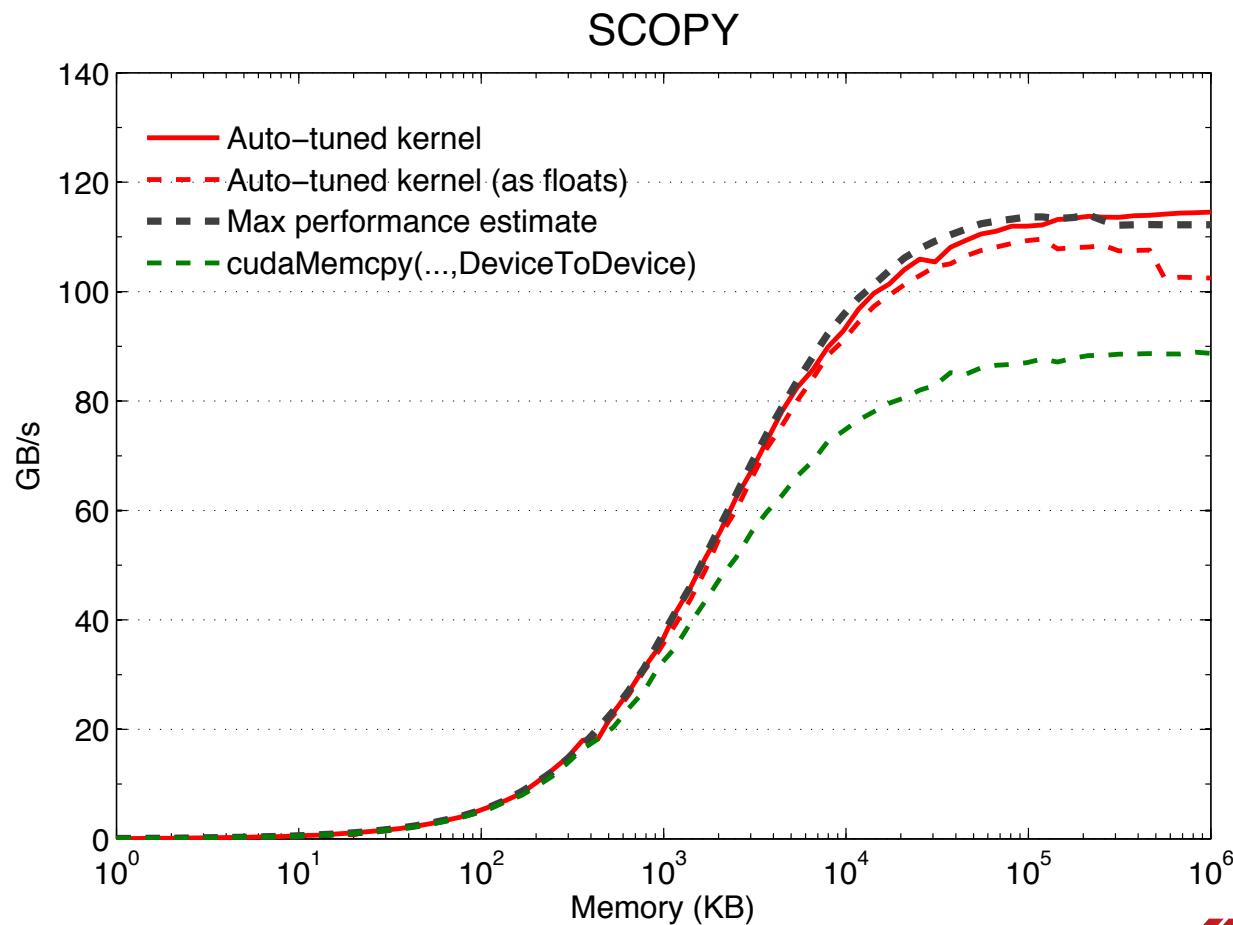
# Performance Prediction

SCOPY bandwidth =  $(1 \times \text{SREAD} + 1 \times \text{SMSET})/2$



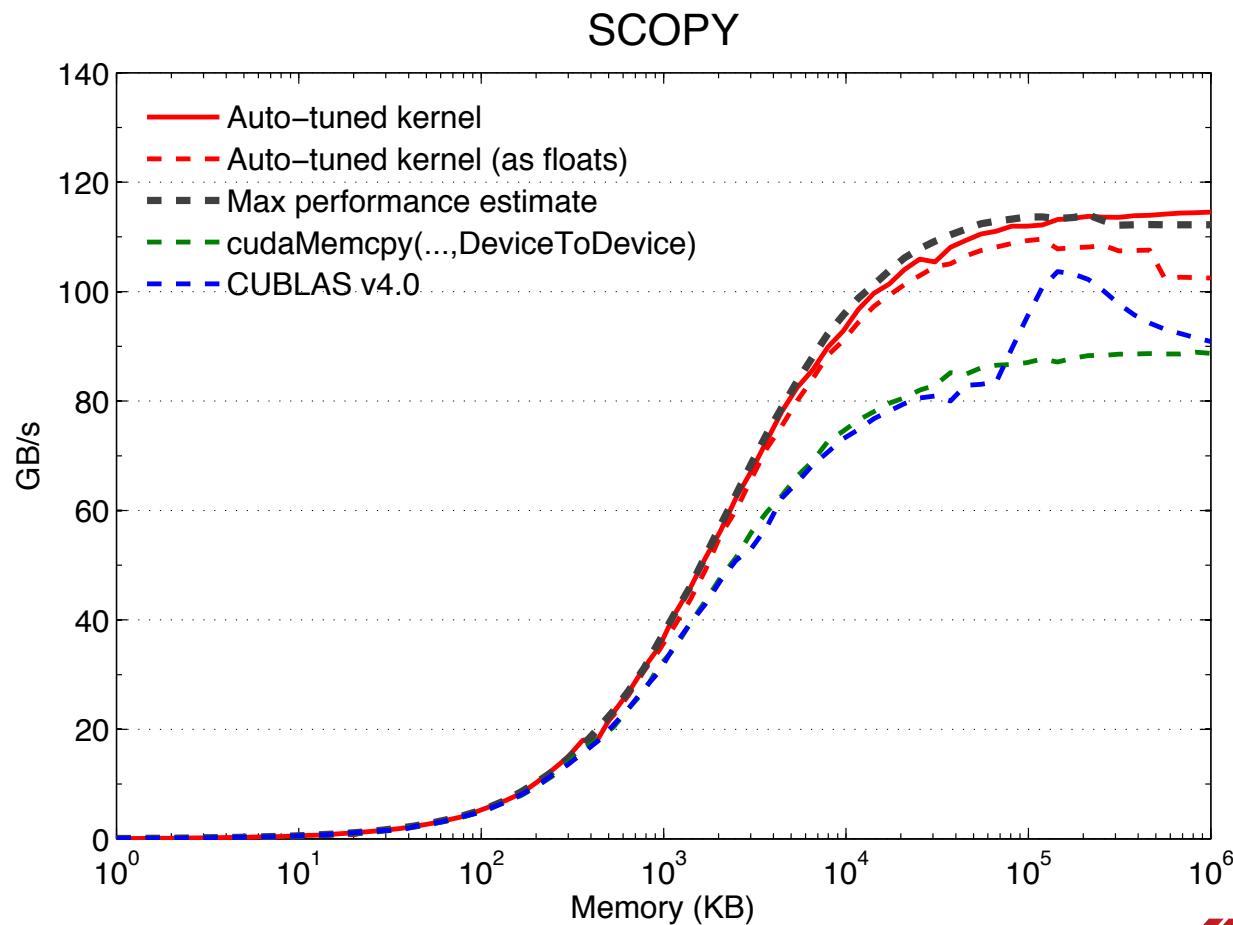
# Performance Prediction

$$\text{SCOPY bandwidth} = (1 \times \text{SREAD} + 1 \times \text{SMSET})/2$$



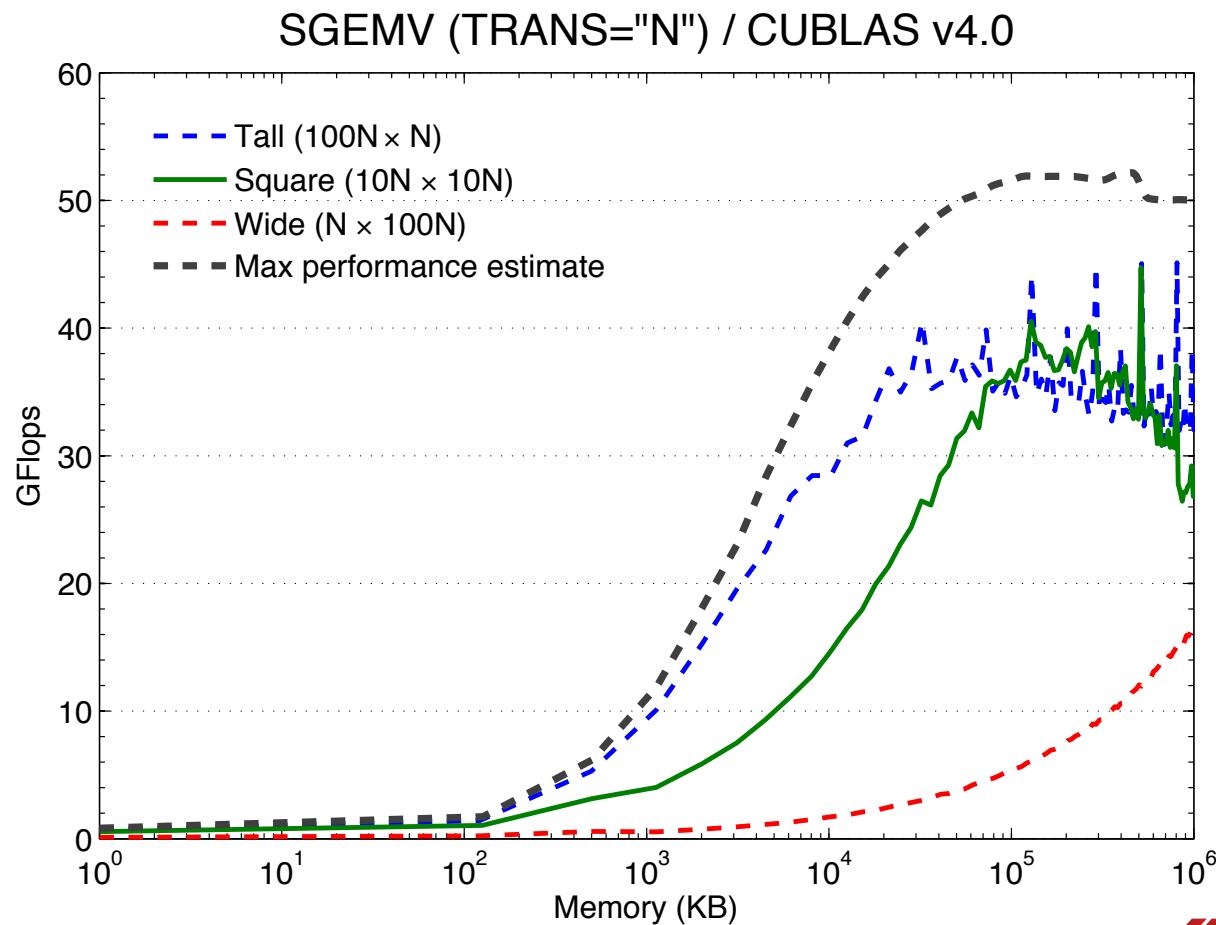
# Performance Prediction

$$\text{SCOPY bandwidth} = (1 \times \text{SREAD} + 1 \times \text{SMSET})/2$$



# Performance Prediction

$$\text{SGEMV} = ((M \times N + N) \times \text{SREAD} + M \times \text{SMSET}) / (M \times N + N + M)$$



# Auto-Tuning Results

# Auto-tuning

- Exhaustive search of pruned parameter space (Fermi):

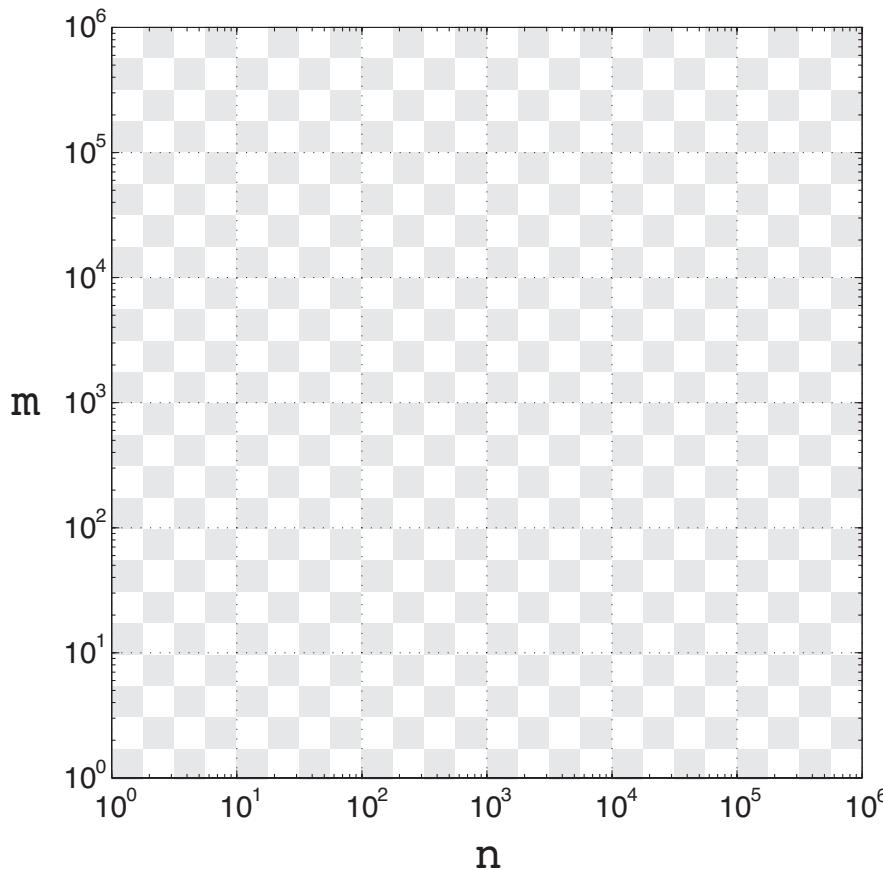
$$\text{BLOCKSIZE} \in \{32, 64, 96, 128, 160, 192, 224, 256\}$$
$$\text{UNROLL\_LEVEL} \in \{1, 2, 3, 4, 5, 6, 7, 8\}$$
$$\text{WORKSIZE\_n/m} \in \{1, 2, 3, 4, 5, 6, 7, 8\} \times \text{BLOCKSIZE}$$
$$\text{INNER/OUTER\_DIAGSIZE} \in \{1, 2, 4, 8, 16\} \times \text{BLOCKSIZE}$$

- Up to  $8^3 = 512$  kernel candidates per tuning run

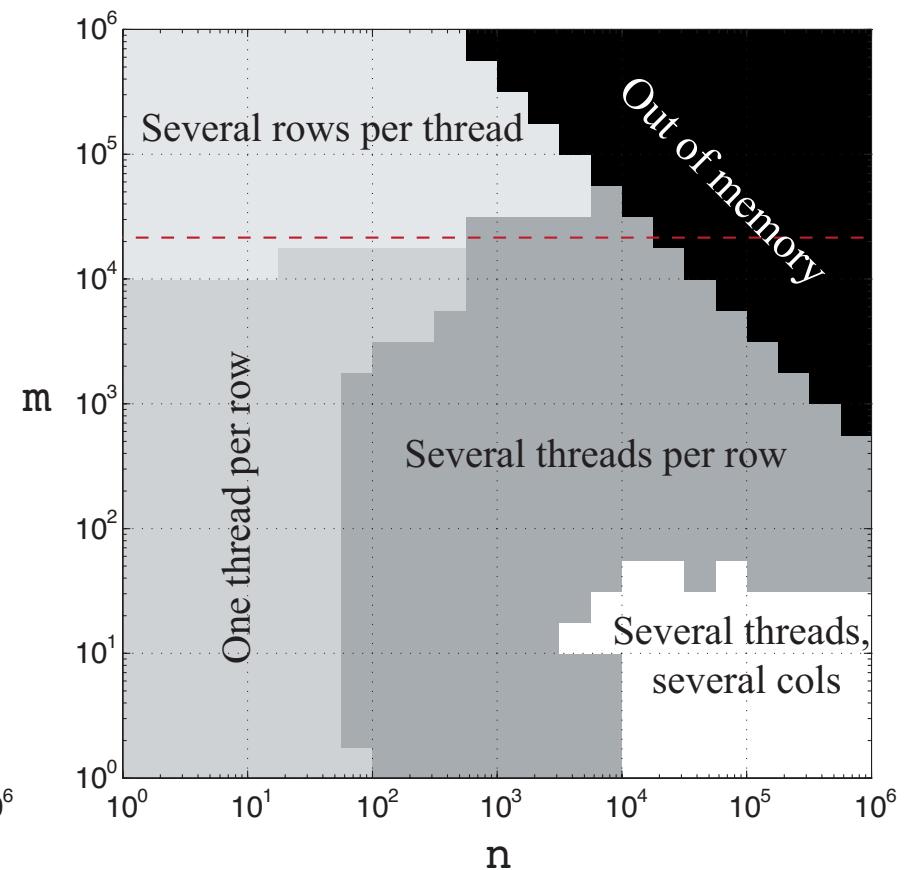
# Auto-tuning

- Tuning mesh in 2D and best kernel selection for SGEMV

Logarithmic mesh

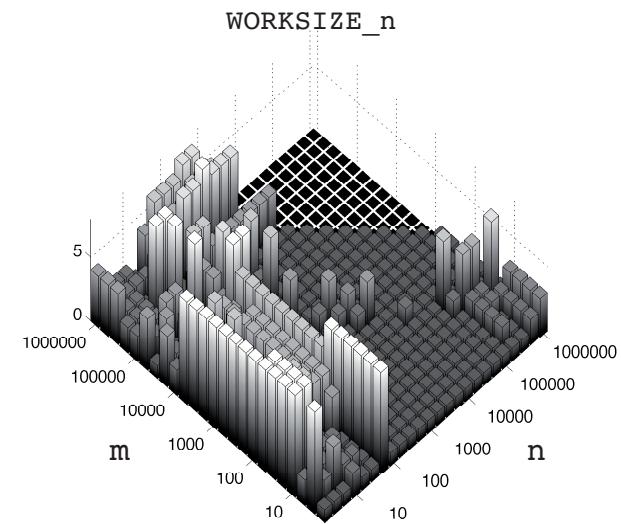
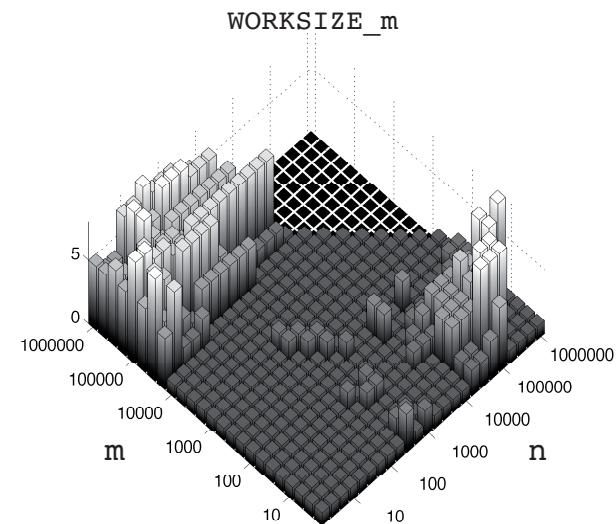
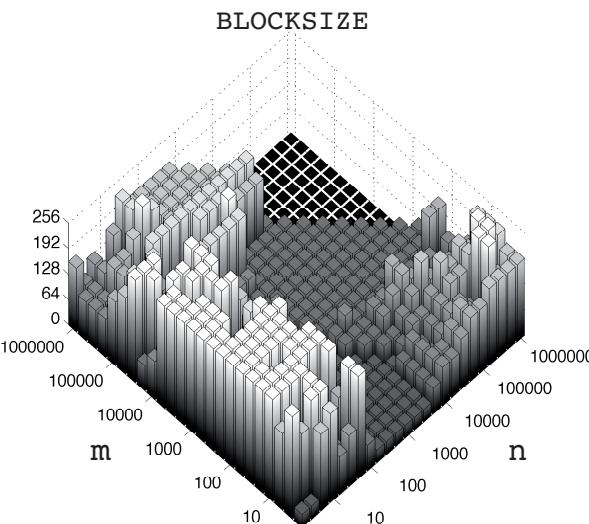


Best Kernel

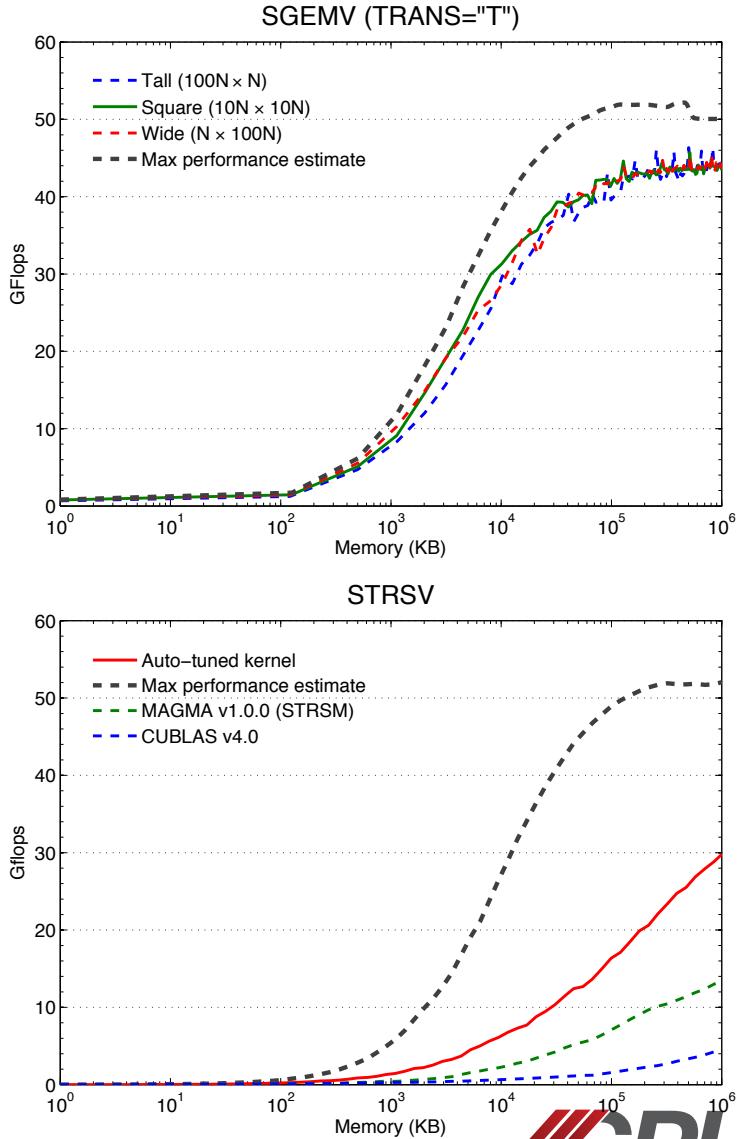
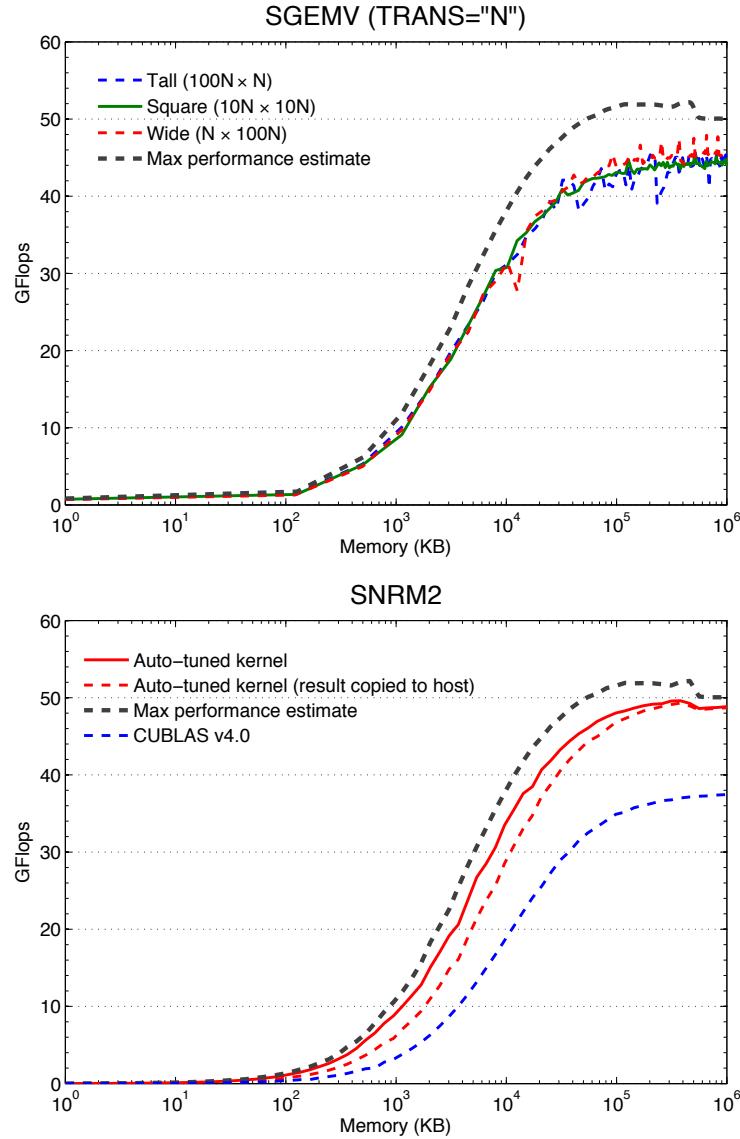


# Auto-tuning

- Best tuning parameters for SGEMV



# Results



# Summary

- BLAS level 1 and 2 kernels are completely memory bound.
  - GPU's effective bandwidth sets the max performance.
  - Simple auto-tuning can facilitate high-performance Vector and Matrix-Vector kernels for all input sizes and shapes.
- 
- GLAS for single precision is available for download at [gpulab.imm.dtu.dk](http://gpulab.imm.dtu.dk) (Open Source MIT License):
    - glas\_v0.2\_C2050\_cuda\_4.0\_linux.tar.gz
    - glas\_v0.2\_GTX590\_cuda\_4.0\_linux.tar.gz



Autotuning: Scam or artistic instrument?