A Fast GEMM Implementation On a Cypress GPU

N.Nakasato (University of Aizu)
in collaboration with
S.Sedukhin, K.Matsumoto, T.Sakai

Agenda

- Performance of GEMM on various accelerators
- Out Implementation on Cypress
- Performance Results for SP, DP, DDP
  - Single Precision, Double Precision, Double-Double Precision
- Summary

3 lines to remember

Programming GPU != CUDA

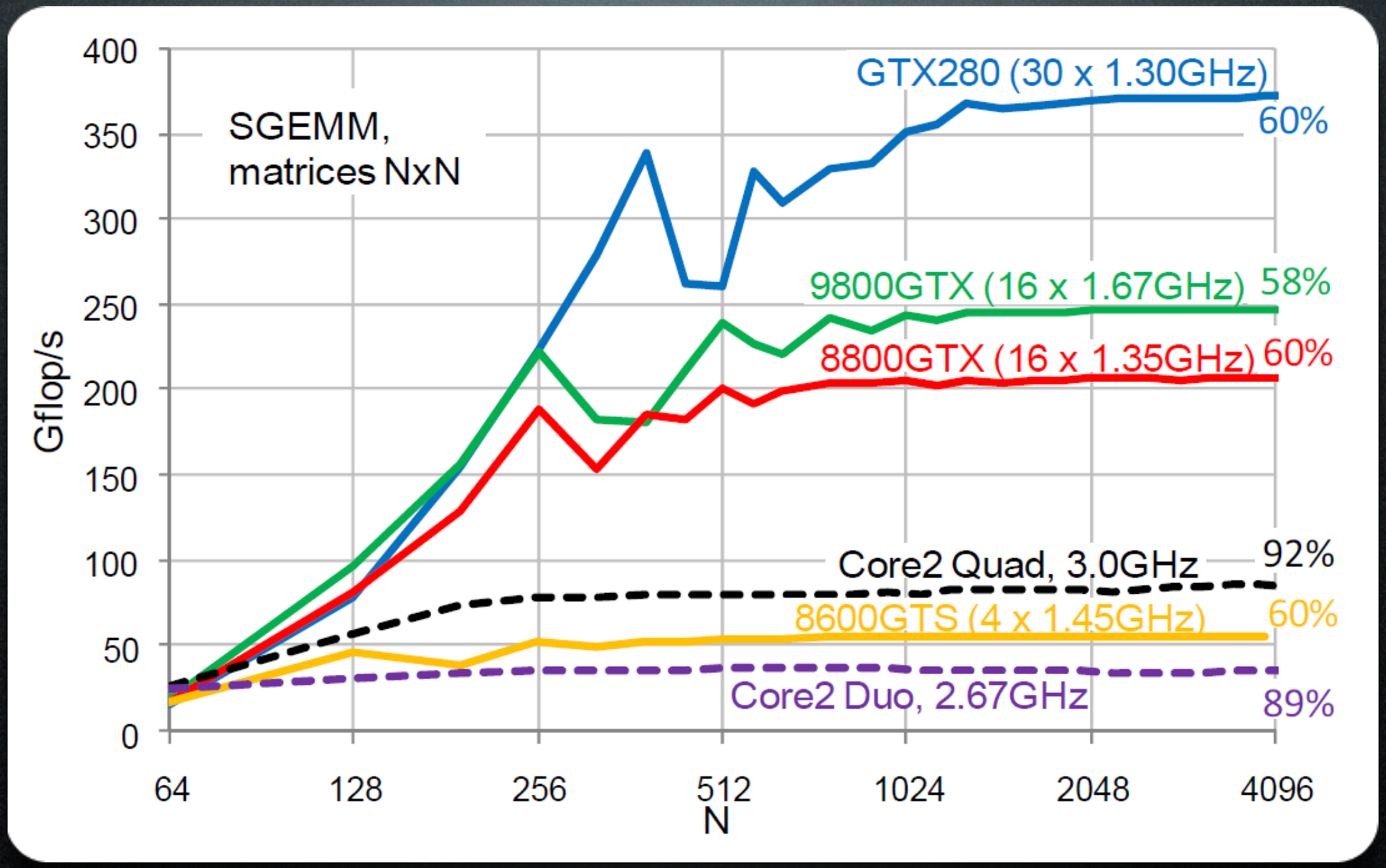
Cache on GPU == GOOD

Double-Double on GPU == FAST

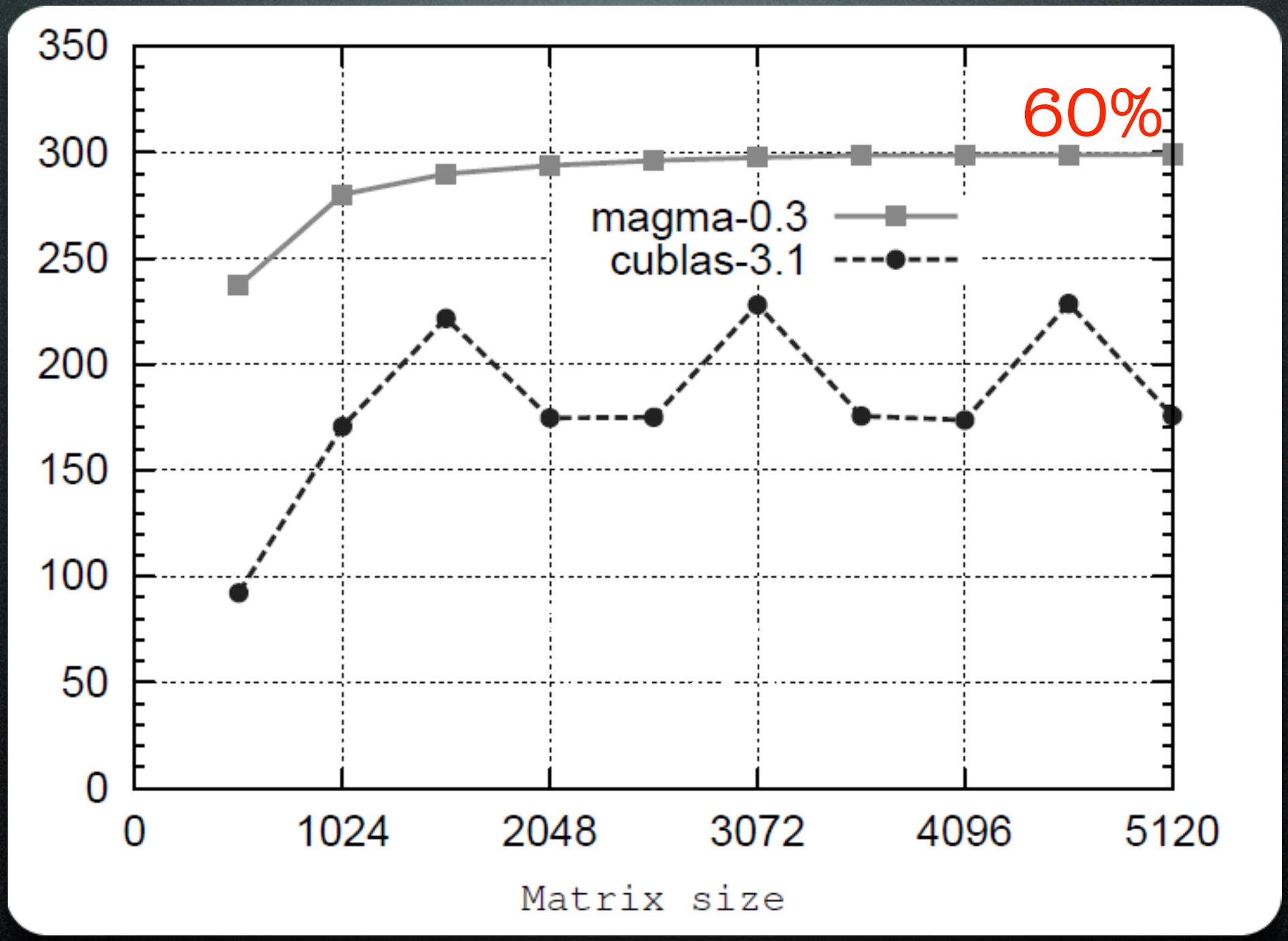
Accelerators

- Emergent architecture for HPC
  - "parallel computer" on a chip
  - Good for compute intensive app.

Complexity	Application	Sustained / Peak
$O(N^3)$ or more	Numerical Integration	100%
$O(N^2)$	N-Body	90% or more
$O(N^{1.5})$	Matrix Multiplication	60% (so far)
O(N log N)	Oct-tree method	1 - 2%
O(N)	Explicit Hydro code	very low in principle

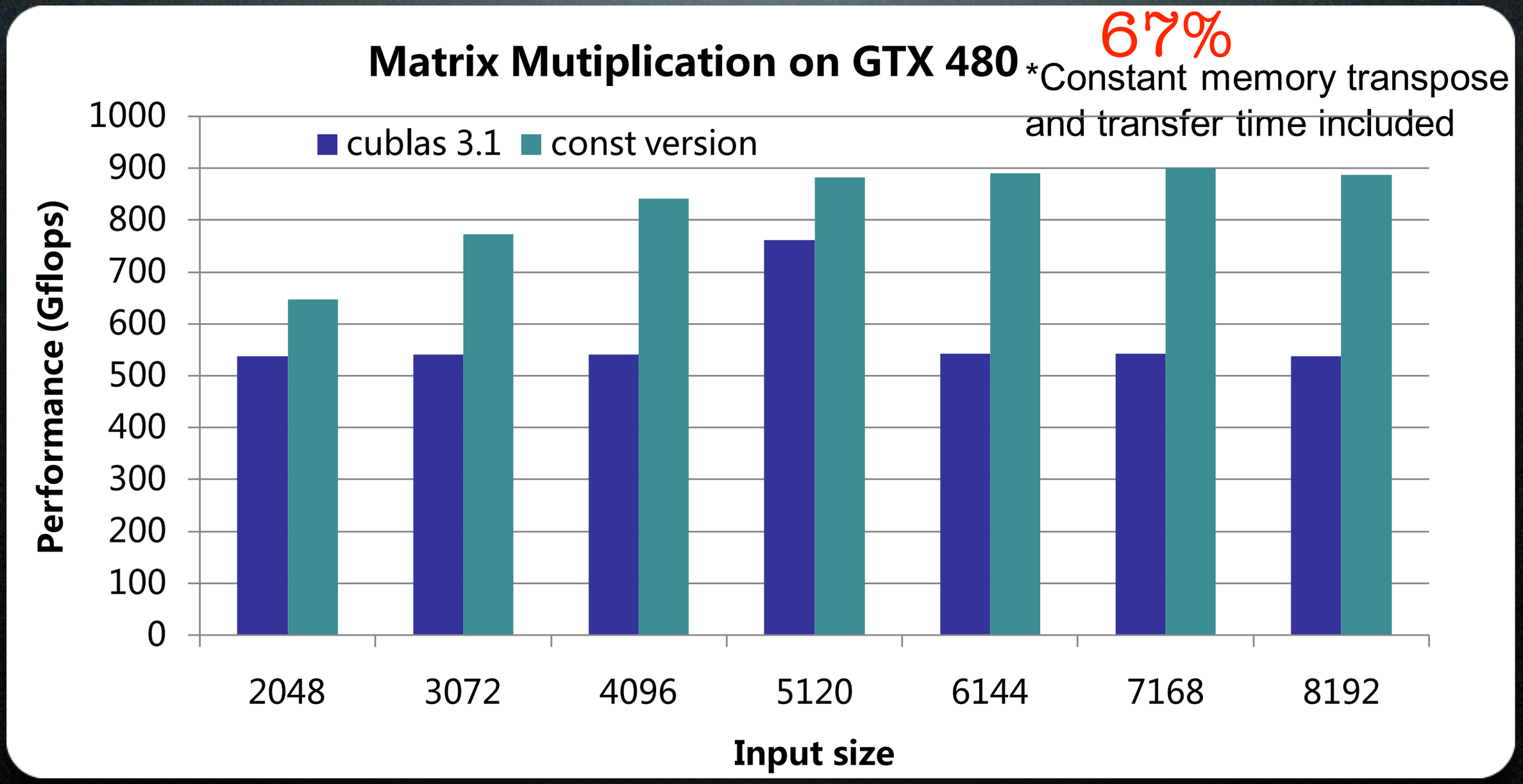


SGEMM on GTX280 etc.
Volkov & Demmel (2008)



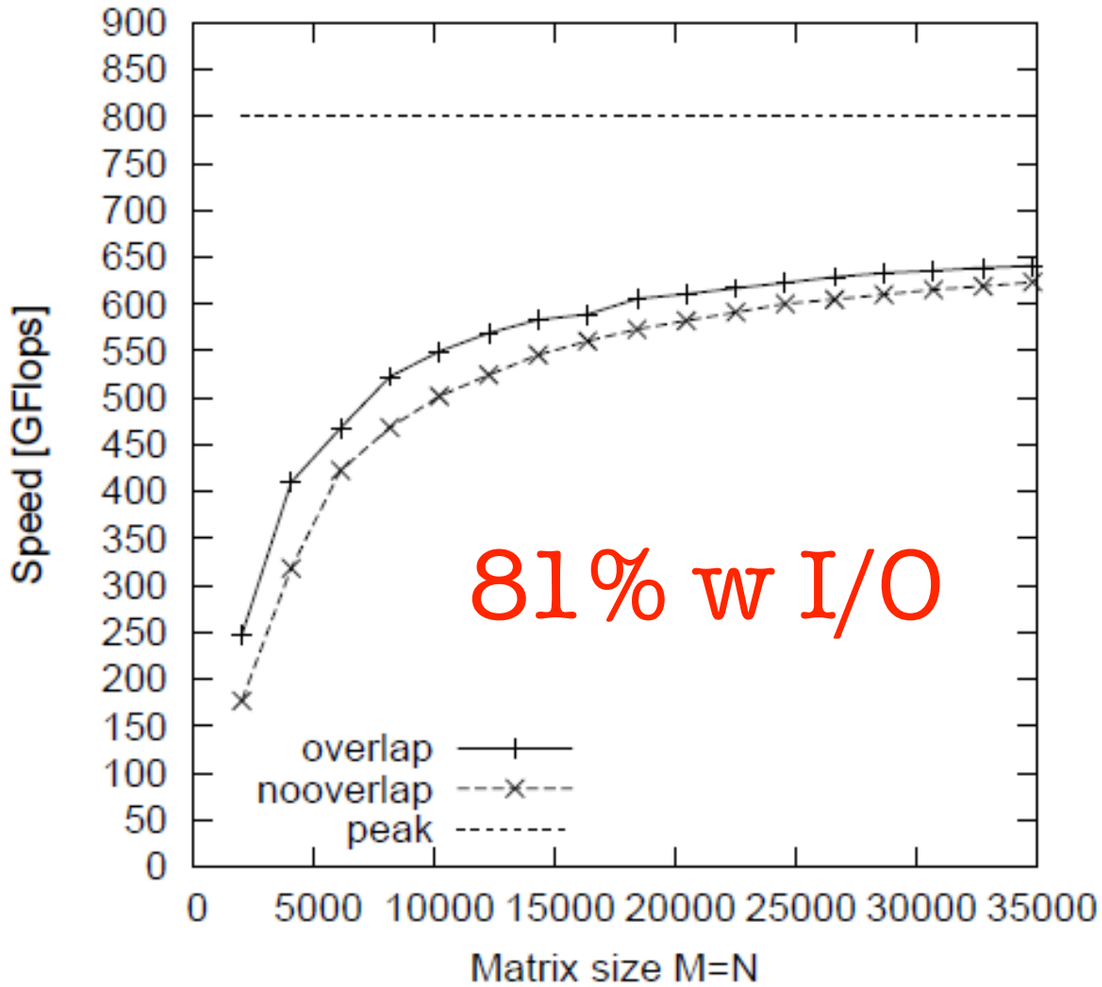
DGEMM on Fermi

Nath, Tomov, & Dongarra (2010)

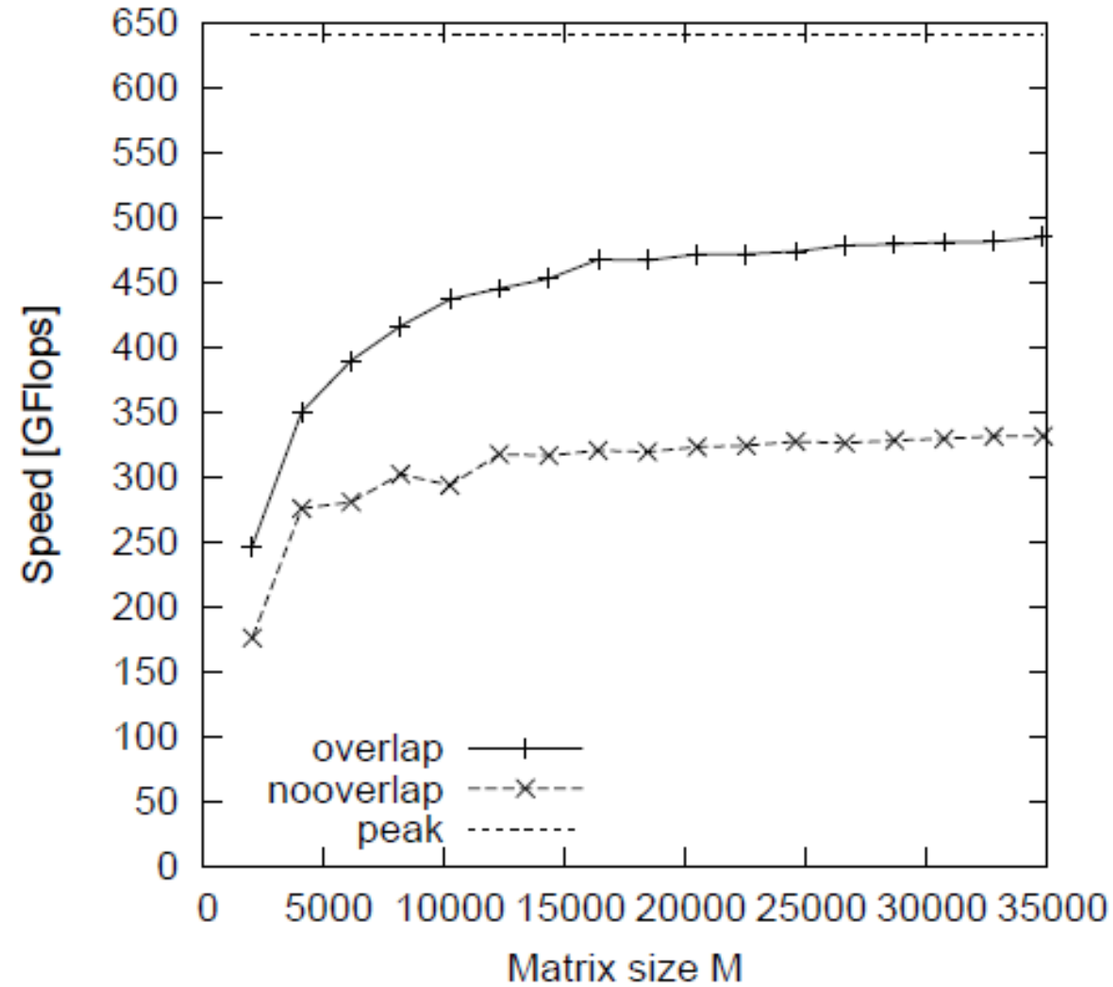


SGEMM on Fermi

Yang & Huiyang (2010)



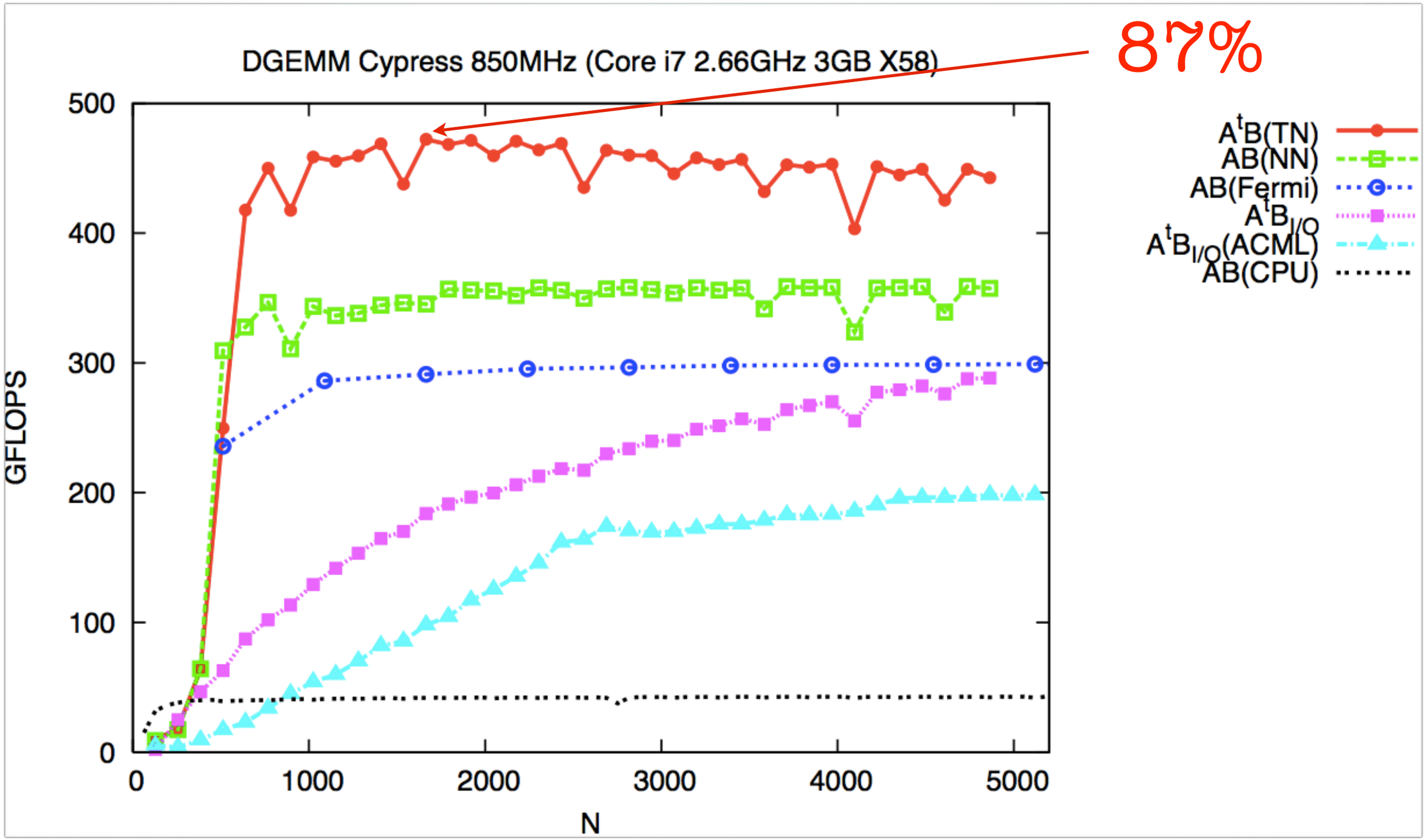
M=N, K=2048, 640 Gflops



N=K=2048, 450 Gflops

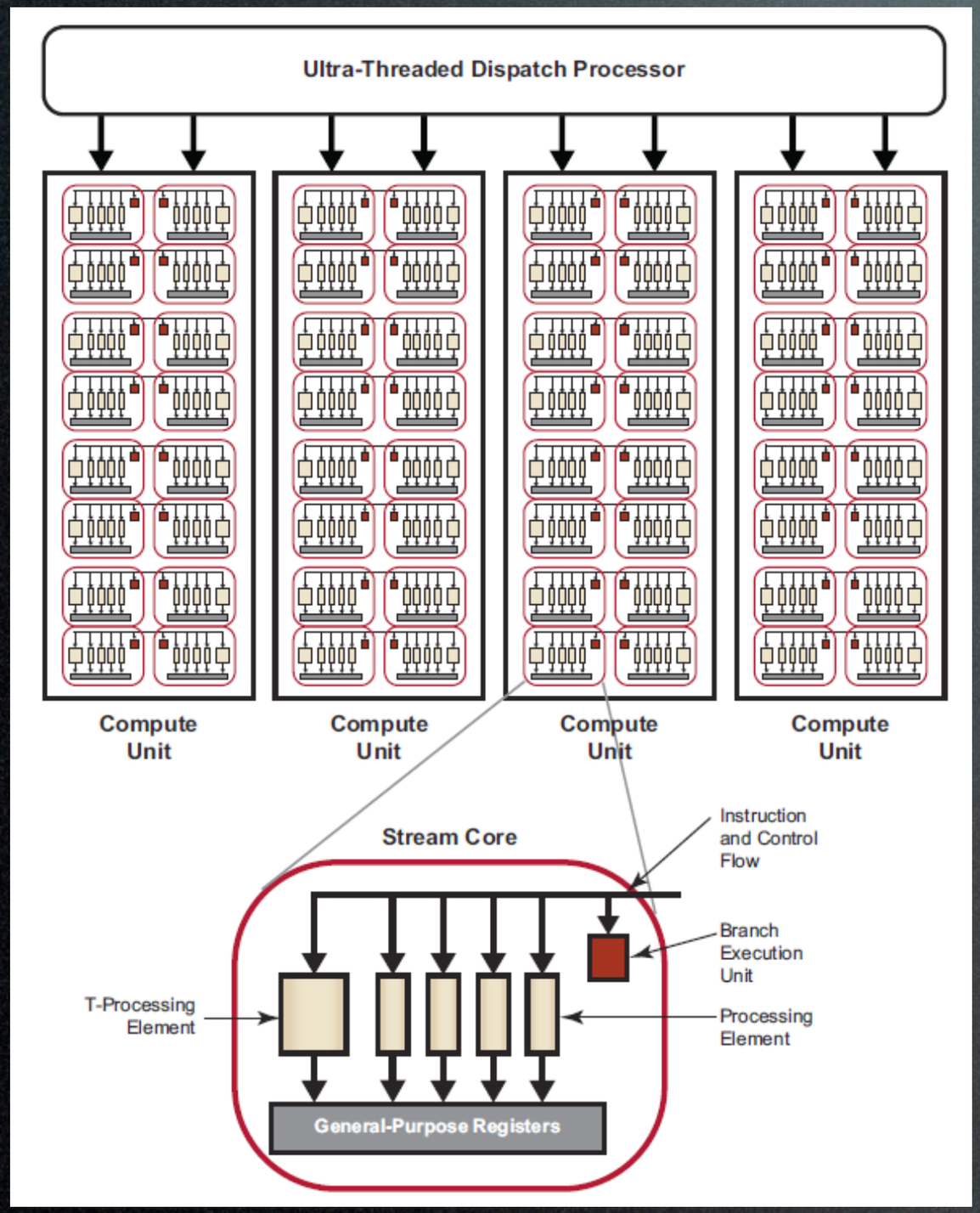
DGEMM on GRAPE-DR

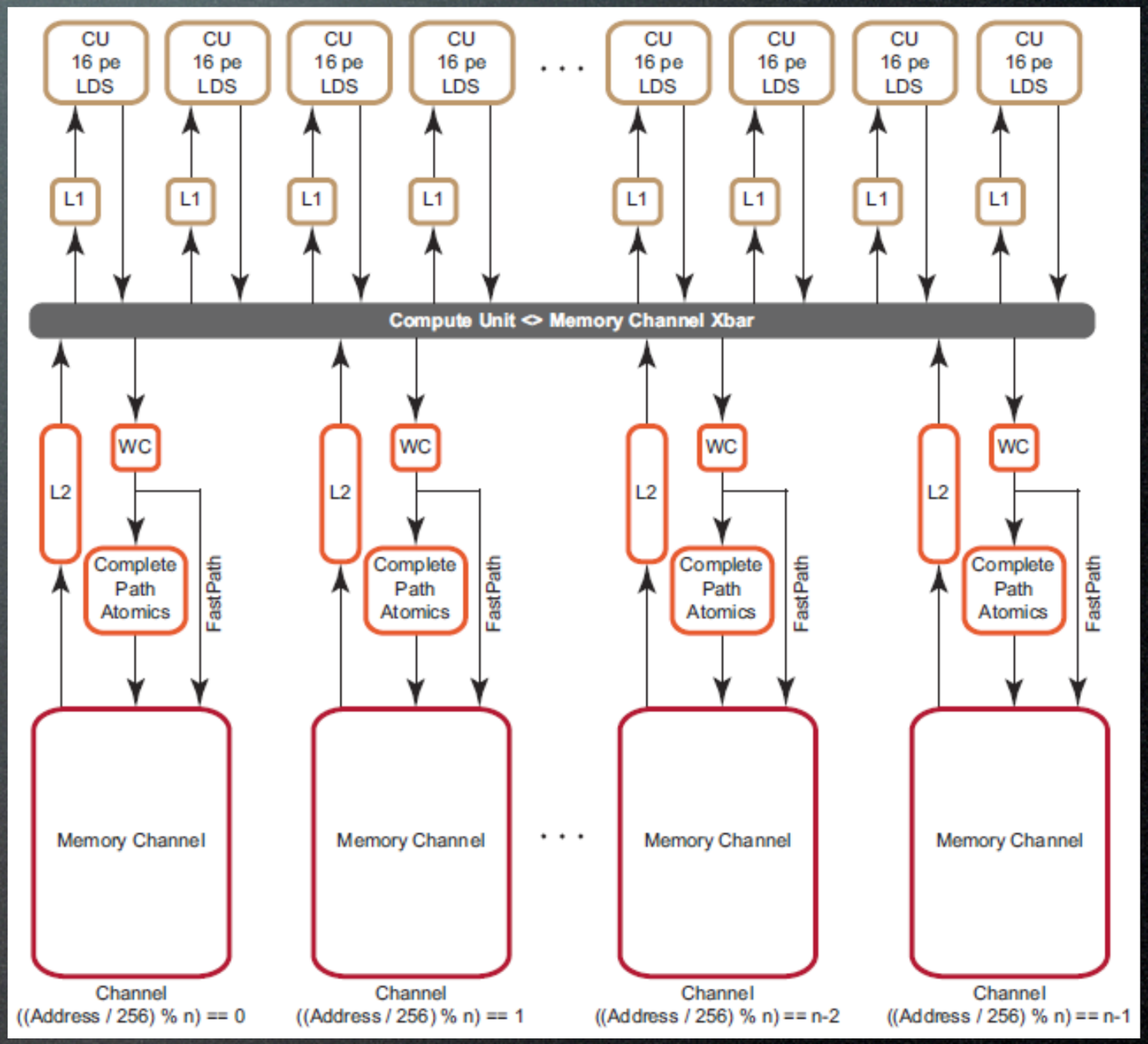
Makino, Koike & Daisaka (2010)



DGEMM on Cypress
Our New Results

Cypress architecture





5-way VLIW unit

- Cypress vs. Fermi

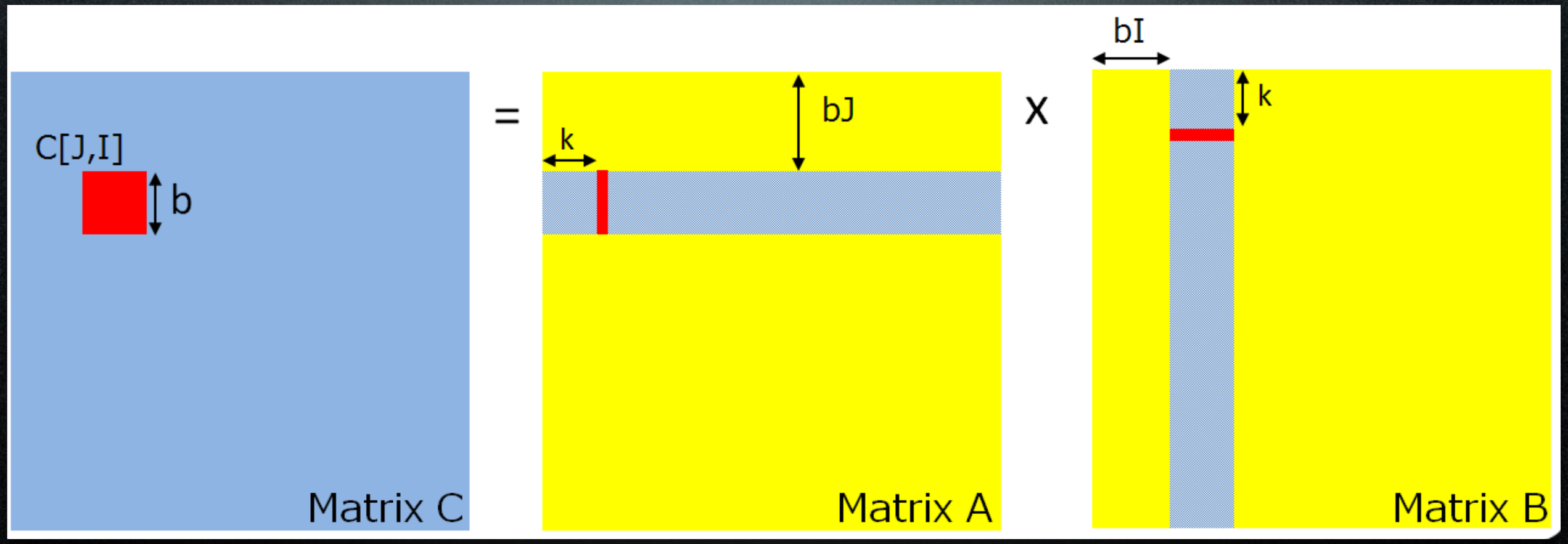
Architecture	Cypress	Fermi
Board Name	Radeon 5870	Tesla C2050
# of SP cores	1600	448
# of DP cores	320	224
# of vector cores	20	14
registers/core	256 KB	128 KB
tex. cache/core	8 KB	12 KB
shared mem./core	32 KB	64 KB
2nd cache	512 KB	768 KB
core clock(GHz)	0.85	1.15
SP peak(Tflop/s)	2.72	1.03
DP peak(Gflop/s)	544	515
memory clock(GHz)	1.2	0.75
memory bus	256	384
memory size(GB)	1	3
memory BW (GB/s)	153.6	144
tex. cache BW(GB/s)	54.5	

DP performance is in similar range

- Cypress memory system
  - Register:128 bit
    - 4, 2, 1 words in SP, DP, DDP
  - Local data share = shared memory
  - L1/L2 cache system for texture
    - high bandwidth > 1 TB/s

Memory type	Size/CU (KB)	Size/GPU (KB)	BW (TB/s)
register	256	5120	13
LDS	32	640	2.2
L1 cache	8	160	1.1
L2 cache	-	512	0.44
Global memory	-	1 GB	0.15

We assign a block of C to a thread



One problem is which memory type we put three matrices to maximize performance?

GEMM on GPU

Previous GEMM for GPU

- Blocking & Rank-1 update
  - Volkov&Demmel(2008) 60%(SP)
    - put A&C on registers
    - put B on shared memory
  - Nath, Tomov, & Dongrrra (2010) 60%(DP)
    - put C on registers
    - put A&B on shared memory
  - Yang&Zhou(2010) 67%(SP)
    - put A on constant cache
    - put B&C on registers

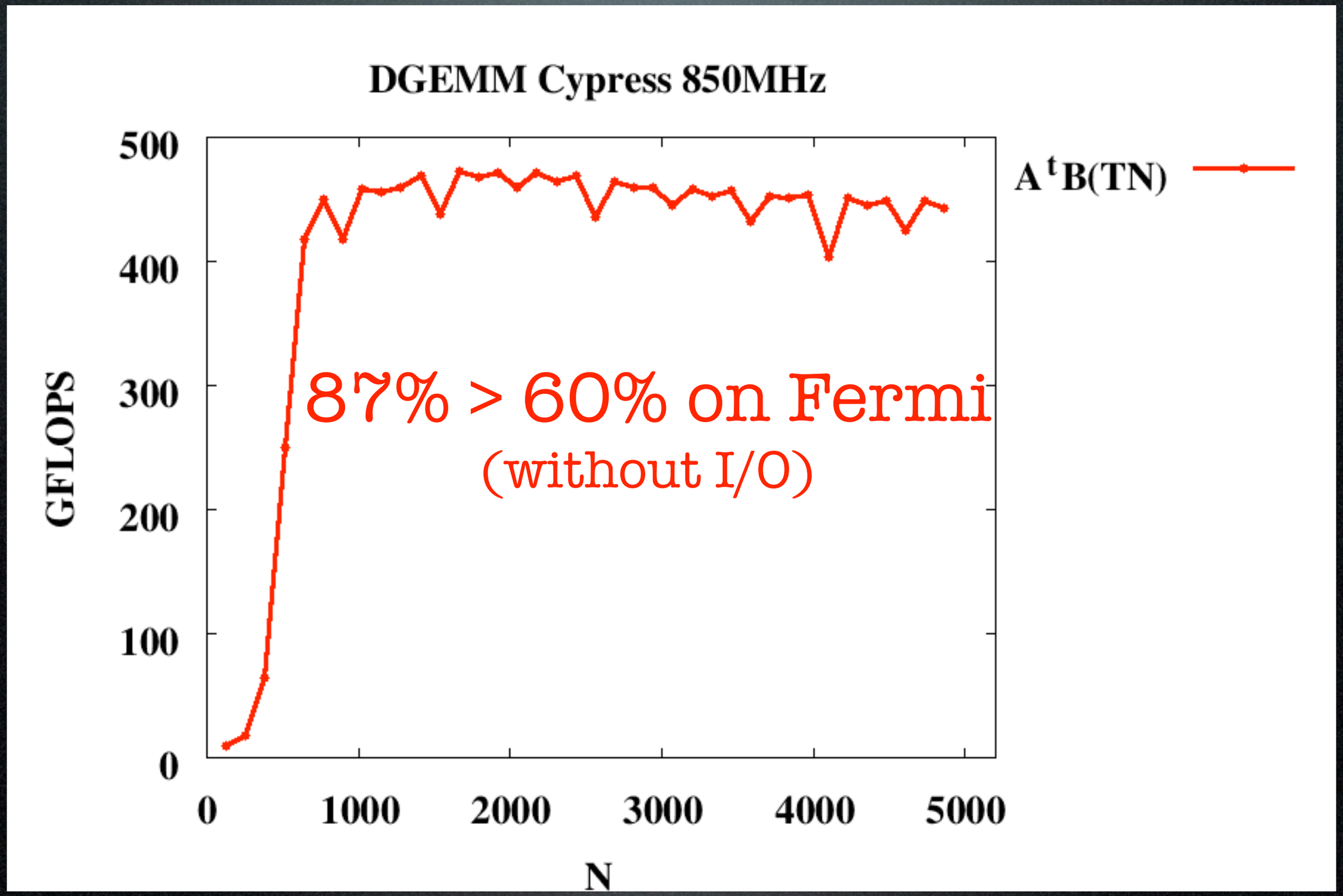
Our GEMM for Cypress

- Blocking & Rank-1 update
  - Put only C on registers
  - Rely on texture cache unit
- SP: 8x8 blocking require 1.36 TB/s
- DP: 4x4 blocking require 1.088 TB/s
- DDP: 2x2 blocking require 300 GB/s

$$\text{Required BW} = F\ S\ b^{-1}\ \text{bytes/s}$$

F : calculation speed in flop/s
S: size of a word in byte
b: blocking factor

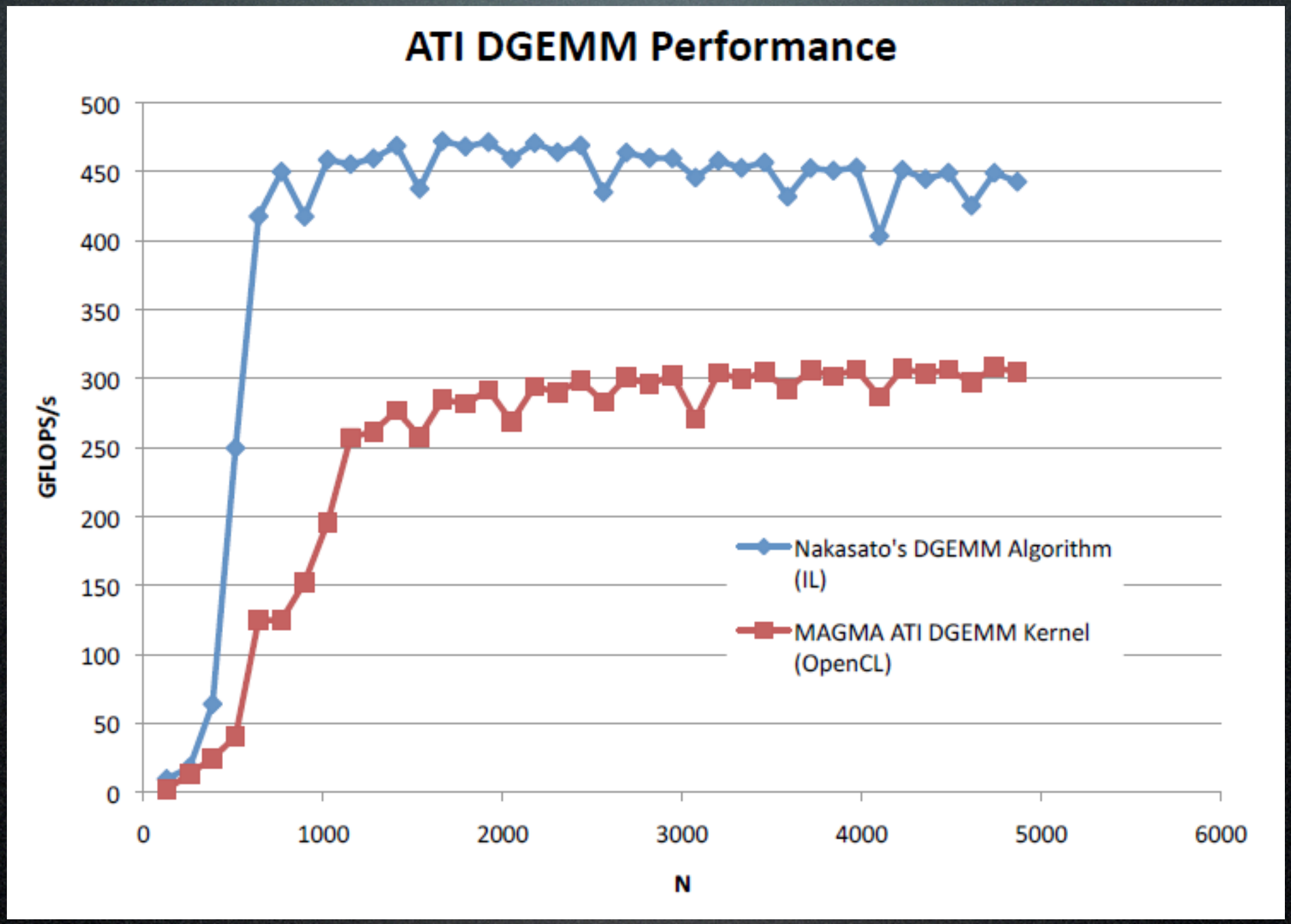
DGEMM Performance



Key techniques

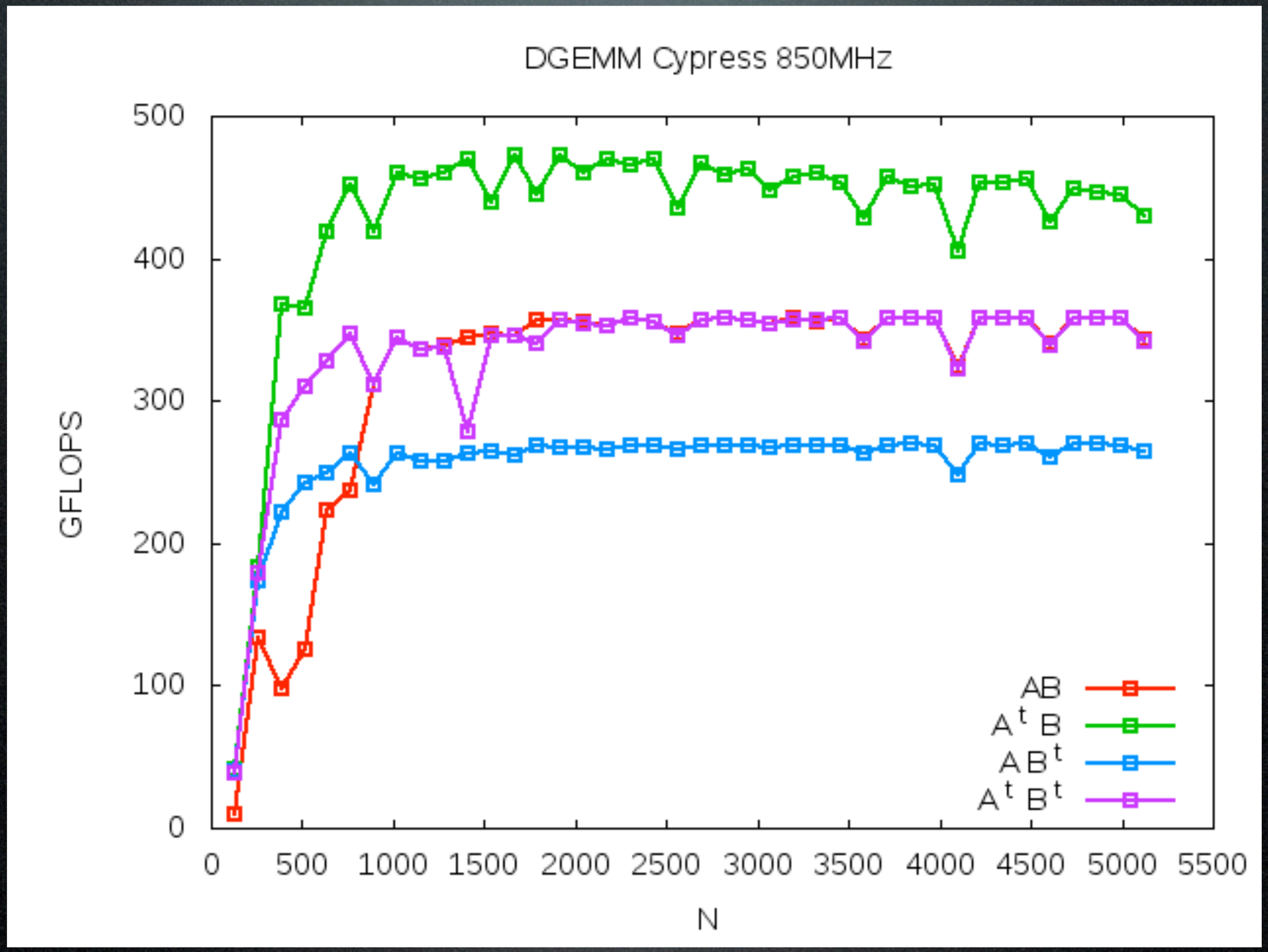
- Implemented in IL
- Texture memory with hardware cache
- Reduce register usage
- Loop unrolling
- FMA64 in DDP

IL vs. OpenCL

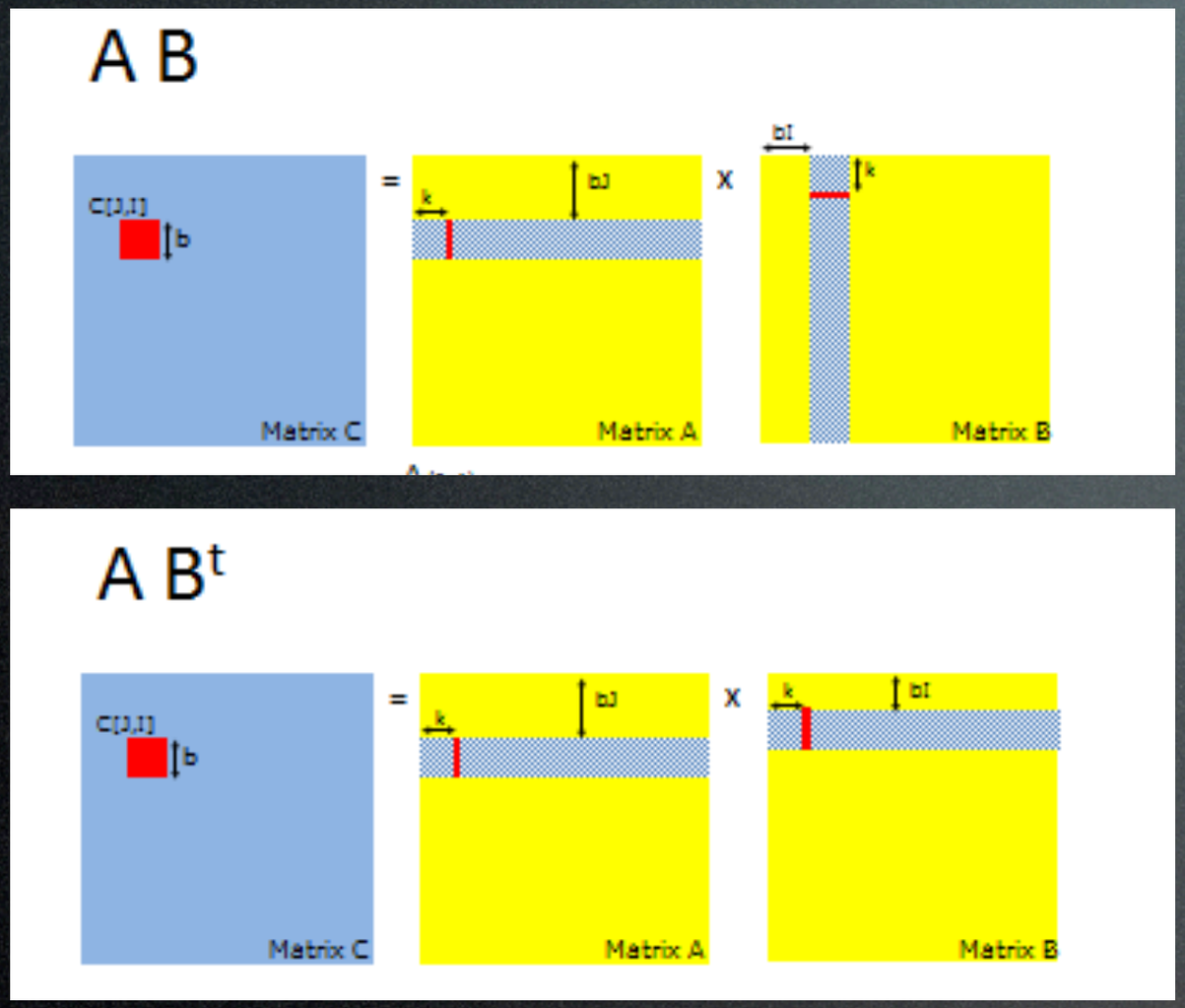


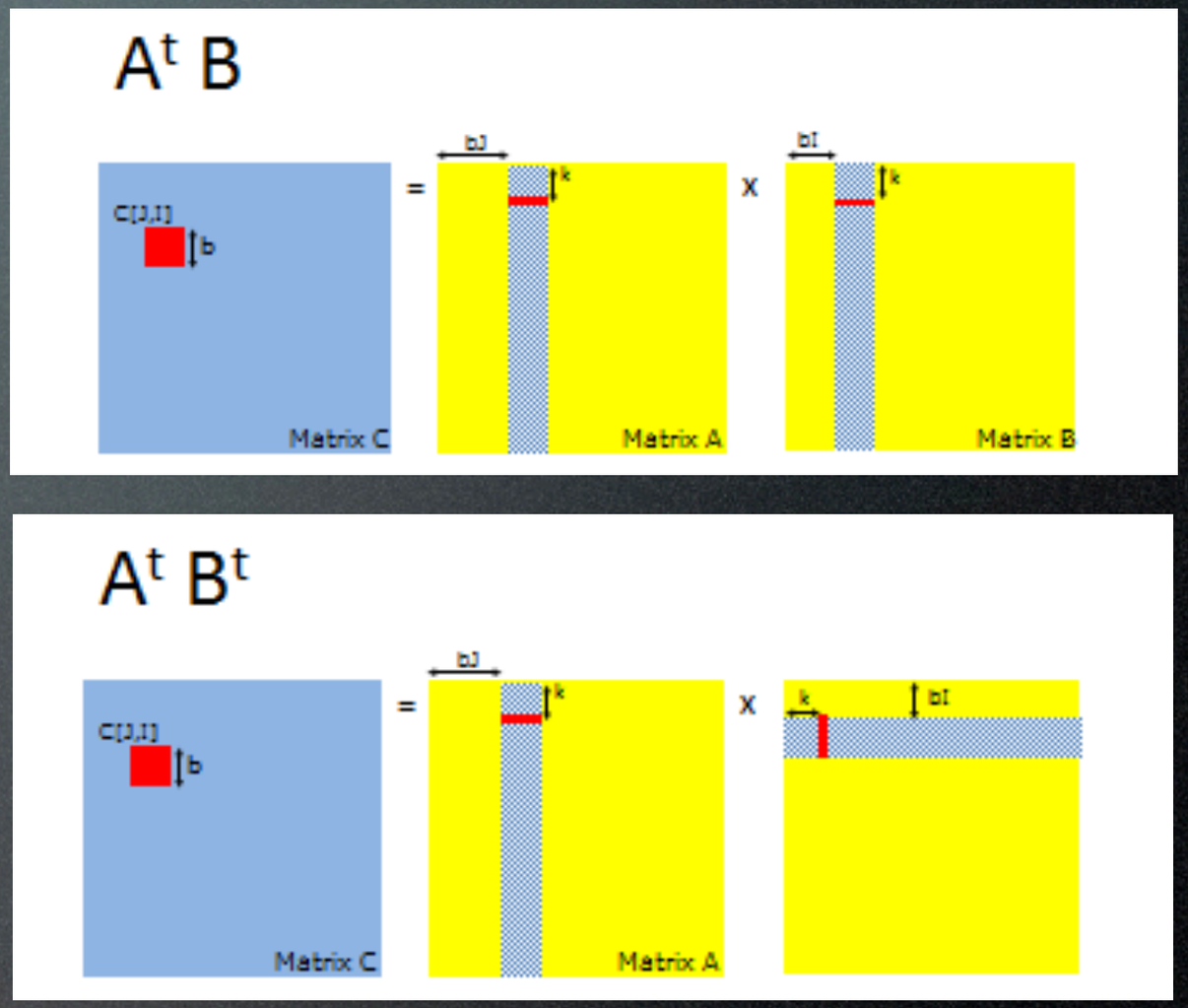
Du, Weber, Luszczek, Tomov, Peterson, Dongarra (2010)

DGEMM with transpose



Transpose options





Transpose options affect memory access patterns

SGEMM Performance

Double-Double on GPU

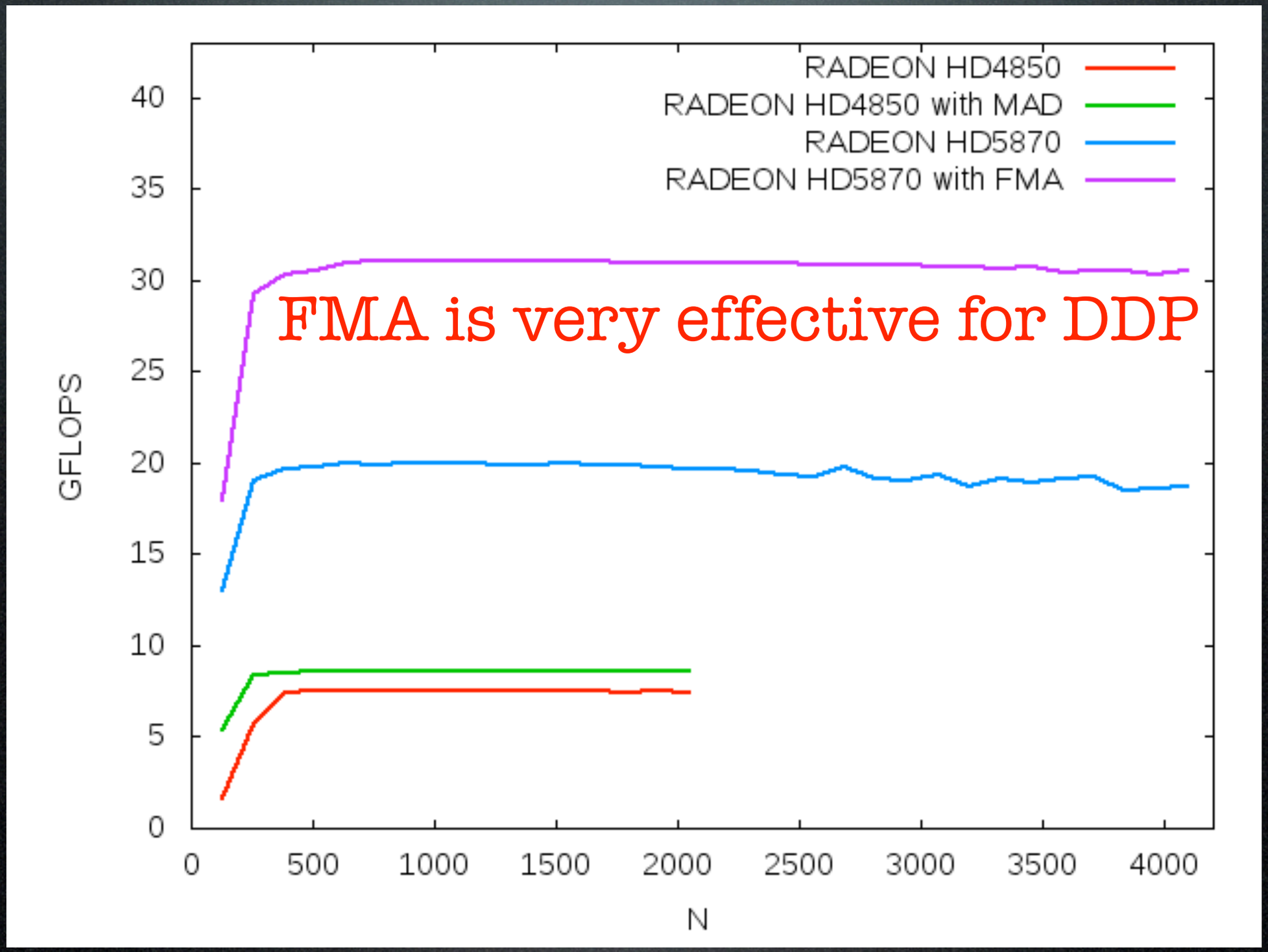
```
func 13
  dmul r2, r0, l9
  dadd r3, r2, r0_neg(yw)
  dadd r1, r2, r3_neg(yw)
  dadd r1.__zw, r0, r1_neg(yw)
  ret
endfunc

func 14
  dmul r11, r10.xy, r10.zw
  mov r0, r10
  call 13
  mov r12, r1
  mov r0, r10.zw
  call 13
  mov r13, r1
  dmad r18.xy, r12.xy, r13.xy, r11_neg(yw)
  dmad r18.xy, r12.xy, r13.zw, r18.xy
  dmad r18.xy, r12.zw, r13.xy, r18.xy
  dmad r11.__zw, r12.zw, r13.zw, r18.xy
  ret
endfunc

func 2
  mov r10.xy,   r30
  mov r10.__zw, r31.xyxy
  call 14
  mov r32, r11
  dmad r32.__zw, r30.xy, r31.zw, r32.zw
  dmad r32.__zw, r30.zw, r31.xy, r32.zw
  mov r0, r32
  call 12
  mov r32, r1
  ret
endfunc
```

35 lines

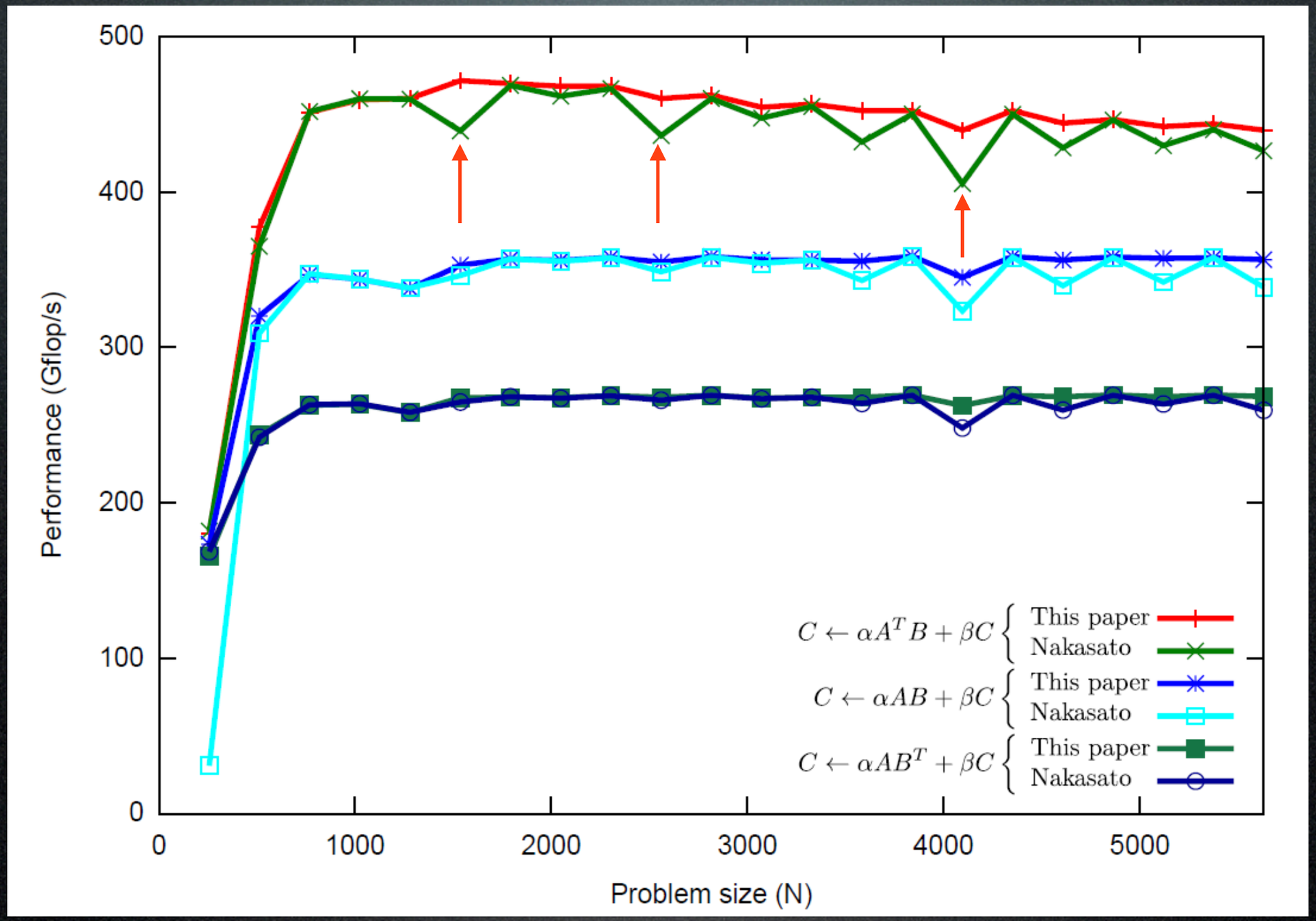
```
func 2
  mov r10.xy,   r30
  mov r10.__zw, r31.xyxy
  call 99
  mov r32, r11
  dmad r32.__zw, r30.xy, r31.zw, r32.zw
  dmad r32.__zw, r30.zw, r31.xy, r32.zw
  mov r0, r32
  call 12
  mov r32, r1
  ret
endfunc

func 99
  dmul r11.xy, r10.xy, r10.zw
  dmad r11.__zw, r10.xy, r10.zw, r11_neg(yw).xy
  ret
endfunc
```

18 lines

DDGEMM Performance

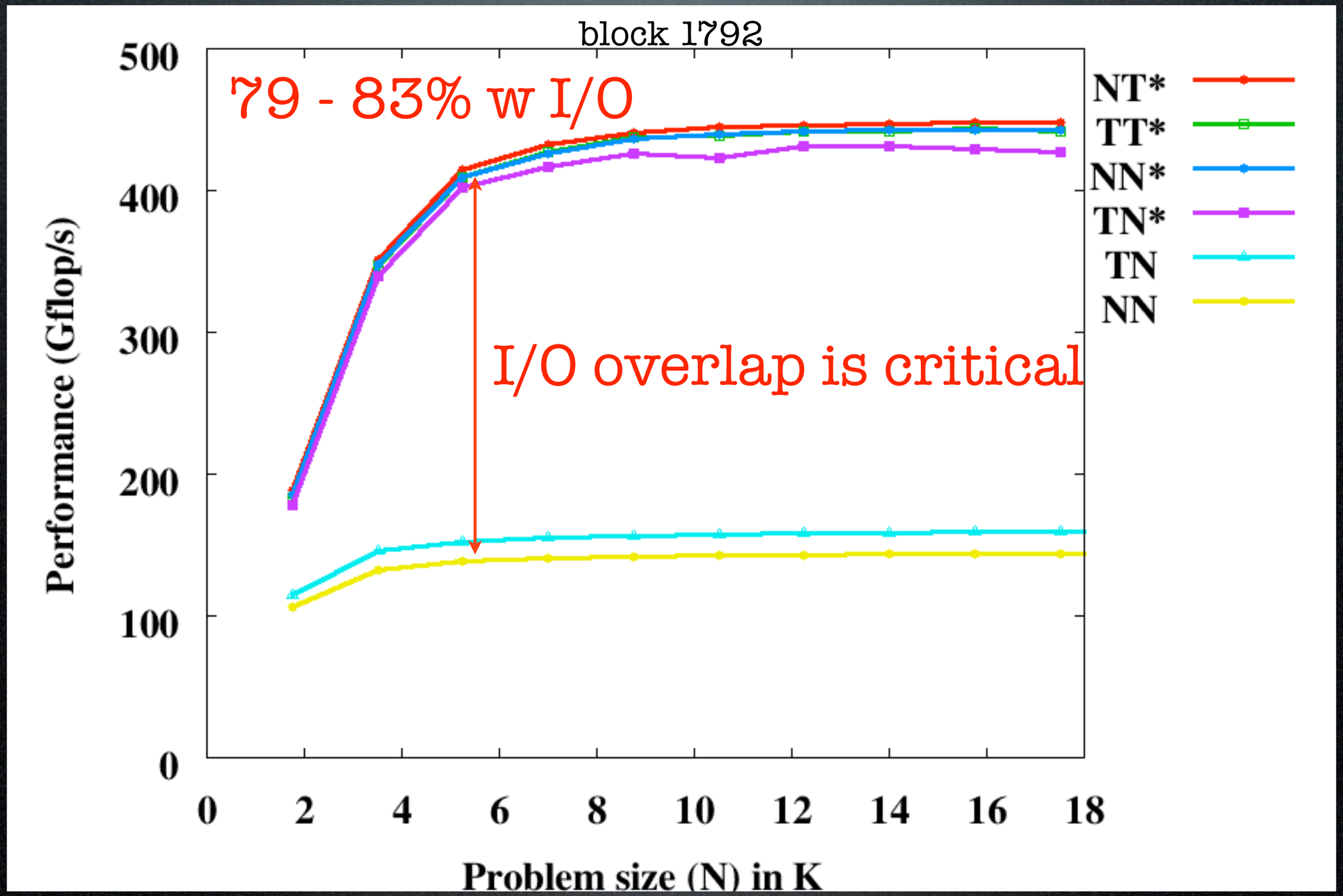


Further optimization

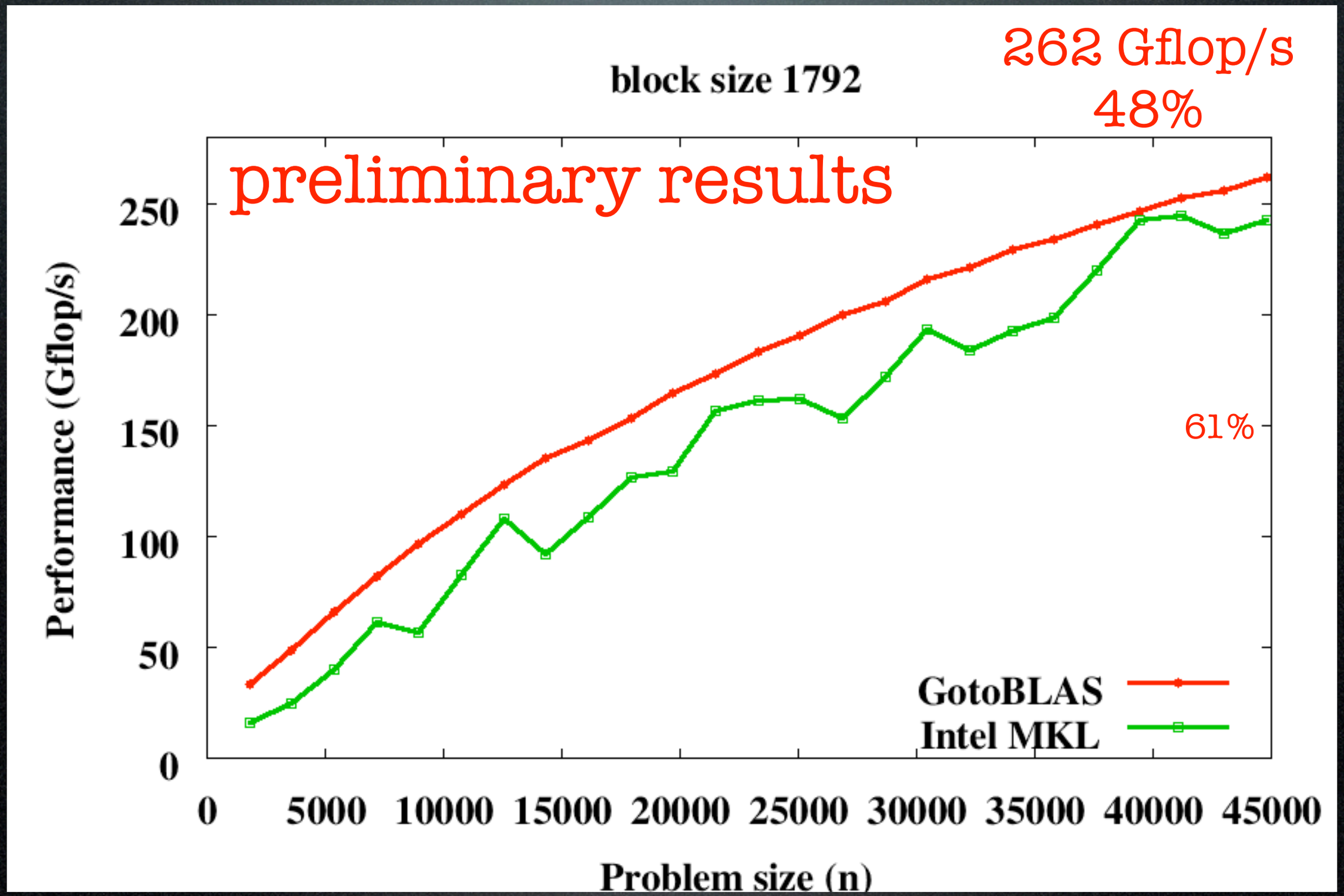


Matsumoto, NN, Sakai, Sedukhin (2011) submitted

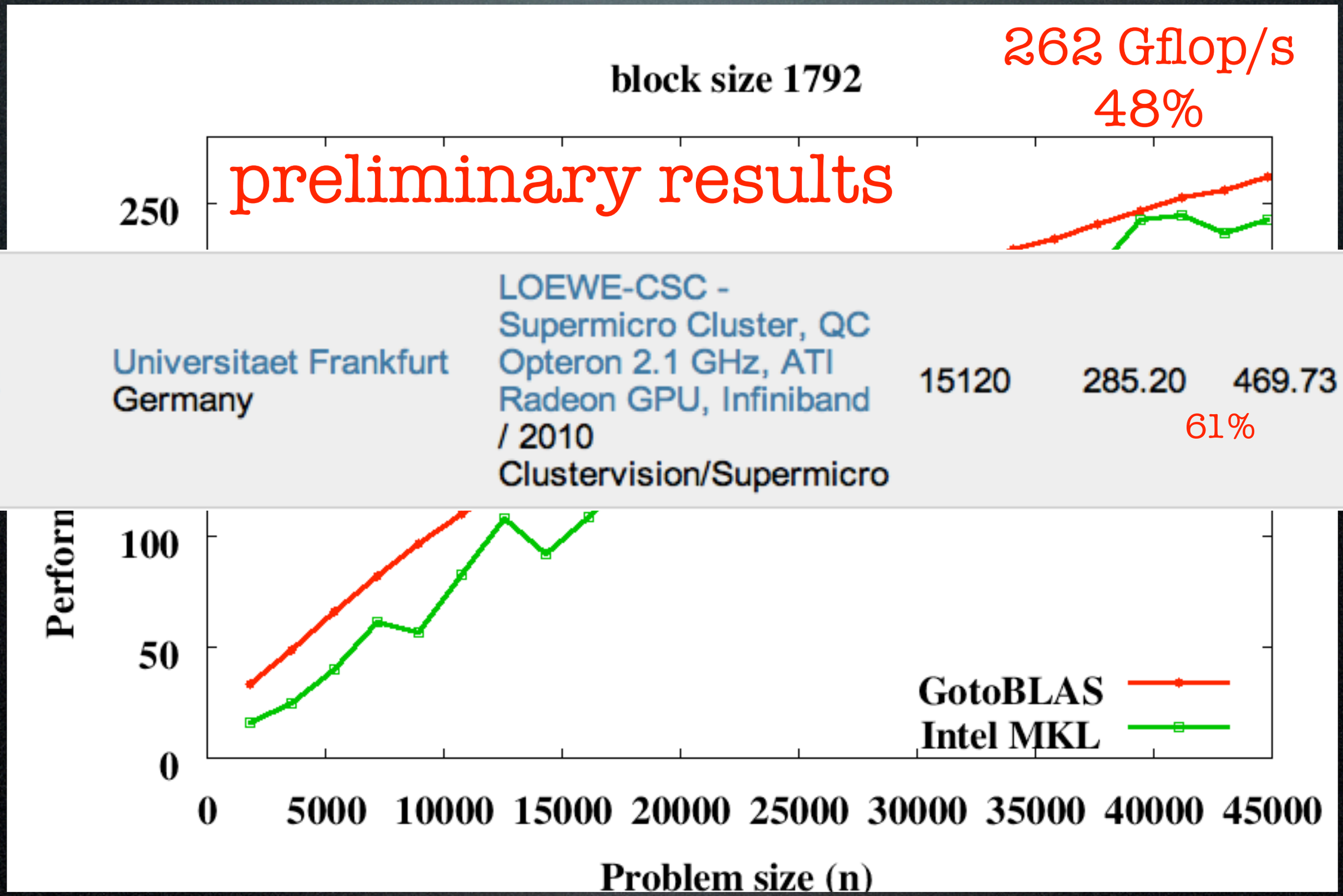
Large matrix



LU decomposition



LU decomposition



22	Universitaet Frankfurt Germany	LOEWE-CSC - Supermicro Cluster, QC Opteron 2.1 GHz, ATI Radeon GPU, Infiniband / 2010 Clustervision/Supermicro	15120	285.20	469.73

61%

Summary

- GEMM kernels on a Cypress GPU
  - S, D, DD kernels are implemented
- Highest efficiency so far : 87%
  - 79 - 83 % with I/O overlap
- Rely on cache system
  - Understandable/No auto-tuning
- DD emulation is fast on the GPU

4 lines to remember

Programming GPU != CUDA

Cache on GPU == GOOD

Double-Double on GPU == FAST

http://github.com/dadeba/dgemm_cypress/