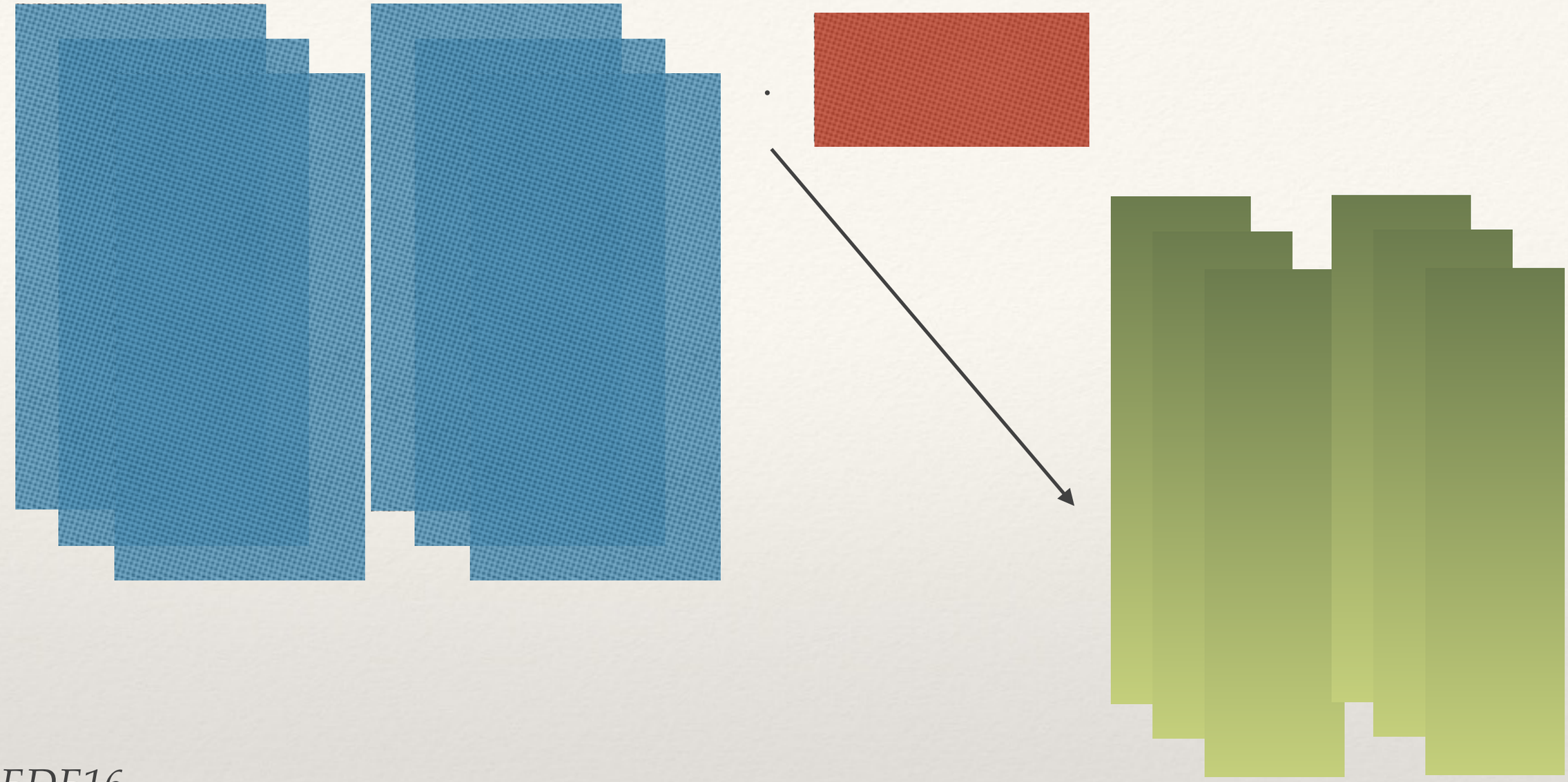


XSEDE16

Standard Linear Algebra Primitives For Tensors

David M. Rogers

USF
UNIVERSITY OF SOUTH FLORIDA

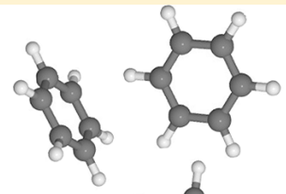
Tensor Contraction Overview

- Quantum Chemistry

  CCSD is the simplest benchmark-quality calculation

$$(pq|rs) \approx B^Q_{pq} B^Q_{rs}$$

$$E^{MO}_{CCSD(T)} \approx E^{NO}_{CCSD(T)} + \Delta E_{MP2}$$



8.5 days ⟶ 2.1 days

DePrince and Sherrill, JCTC 9:2687, 2013.
Hohenstein, Parrish, Sherrill, and Martinez, JCP 137, 2012.

"THC can be used to *both* reduce the scaling with respect to molecular size of coupled cluster singles and doubles (CCSD) (and related methods) by two orders [from $O(N^6)$ to $O(N^4)$] *and* remove the memory bottleneck associated with storage of the doubles amplitudes."

- Multi-dimensional data
- Continuum Mechanics
- Computational Fluid Dynamics

Tensor Contraction Overview

3. $\langle \Phi_{ij}^{ab} | [[\hat{H}, \hat{T}_1], \hat{T}_1] | \Phi_0 \rangle$

O(N⁴) proof, but no impl.

$$\langle \Phi_{ij}^{ab} | (\hat{V}\hat{T}_1\hat{T}_1)_c | \Phi_0 \rangle = \frac{1}{2} (ac|bd)\, t_i^c t_j^d + \frac{1}{2} (ik|jl)\, t_k^a t_l^b - (ik|bc)\, t_k^a t_j^c - (ia|kc)\, t_k^b t_j^c \tag{286}$$

$$\tilde{T}_{AB} \leftarrow \frac{1}{2} \tau_i^A \tau_a^A (ac|bd)\, t_i^c t_j^d \tau_j^B \tau_b^B \tag{287}$$

$$\tilde{T}_{AB} \leftarrow \frac{1}{2} \tau_i^A \tau_a^A X_a^P X_c^P Z^{PQ} X_b^Q X_d^Q t_i^c t_j^d \tau_j^B \tau_b^B \tag{288}$$

$$A_{iP} = t_i^c X_c^P \tag{289}$$

$$B_{jQ} = t_j^d X_d^Q \tag{290}$$

$$C_{PA} = A_{iP} \tau_i^A \tag{291}$$

$$D_{PA} = X_a^P \tau_a^A \tag{292}$$

$$E_{PA} = D_{PA} E_{PA} \tag{293}$$

$$F_{QB} = B_{jQ} \tau_j^B \tag{294}$$

$$G_{QB} = X_b^Q \tau_b^B \tag{295}$$

$$H_{QB} = F_{QB} G_{QB} \tag{296}$$

$$I_{PB} = H_{QB} Z^{PQ} \tag{297}$$

$$\tilde{T}_{AB} \leftarrow \frac{1}{2} E_{PA} I_{PB} \tag{298}$$

Hohenstein, Parrish, Sherrill, and Martinez, JCP 137, 2012.

Outline

- Definitions
- High-level interface
  - Example uses
  - Internal representation
- Low-level kernel details
  - Timings of Conventional Algorithms
  - Loop Nesting Optimization
  - Code Generator Operation

Definition

$$(A \cdot B)_{ij} = \sum_k A_{ik} B_{kj}$$

tensordot(A, B, indices=[(1,), (0,)])

Specifies indices of A and B to be ‘contracted’

Contraction = match and sum over

4 different kinds!

$$(A \cdot B)_{ij} = \sum_k A_{ik} B_{kj}$$ *nn* tensordot(A, B, indices=[(1,), (0,)])

$$(A \cdot B^T)_{ij} = \sum_k A_{ik} B_{jk}$$ *nt* tensordot(A, B, indices=[(1,), (1,)])

$$(A^T \cdot B)_{ij} = \sum_k A_{ki} B_{kj}$$ *tn* tensordot(A, B, indices=[(0,), (0,)])

$$(A^T \cdot B^T)_{ij} = \sum_k A_{ki} B_{jk}$$ *tt* tensordot(A, B, indices=[(0,), (1,)])

$$(A \cdot B)^T = B^T \cdot A^T$$ can reorder arguments to transpose

General Form

$$C_{ij} = \sum_{kl} A_{kil} B_{ljk}$$

- may have multiple contracted indices
- may have multiple output indices
- no unique ordering can be assumed

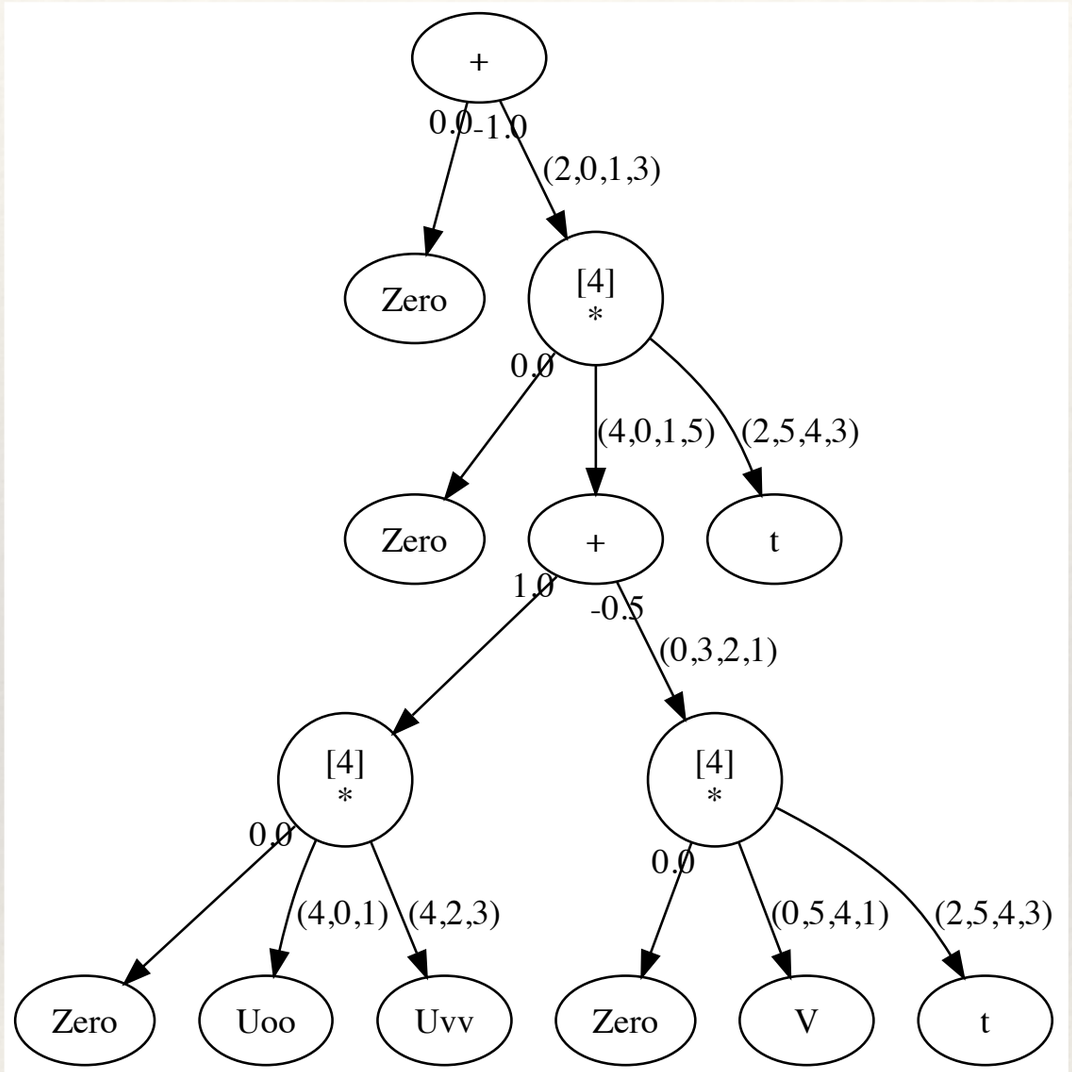
Answer 1 (numpy): rearrange A and B to matrices

- tensordot becomes _GEMM
- only need to specify “contracted” indices from A,B
- output order is A then B

Answer 2: create a “logical output” ordering

- include both contracted and uncontracted indices
- Need to specify output index for every index of A,B

High-Level Interface



- Just turn my equations into code!
- Run DAG of operations starting from given inputs.
- Return the result (top)

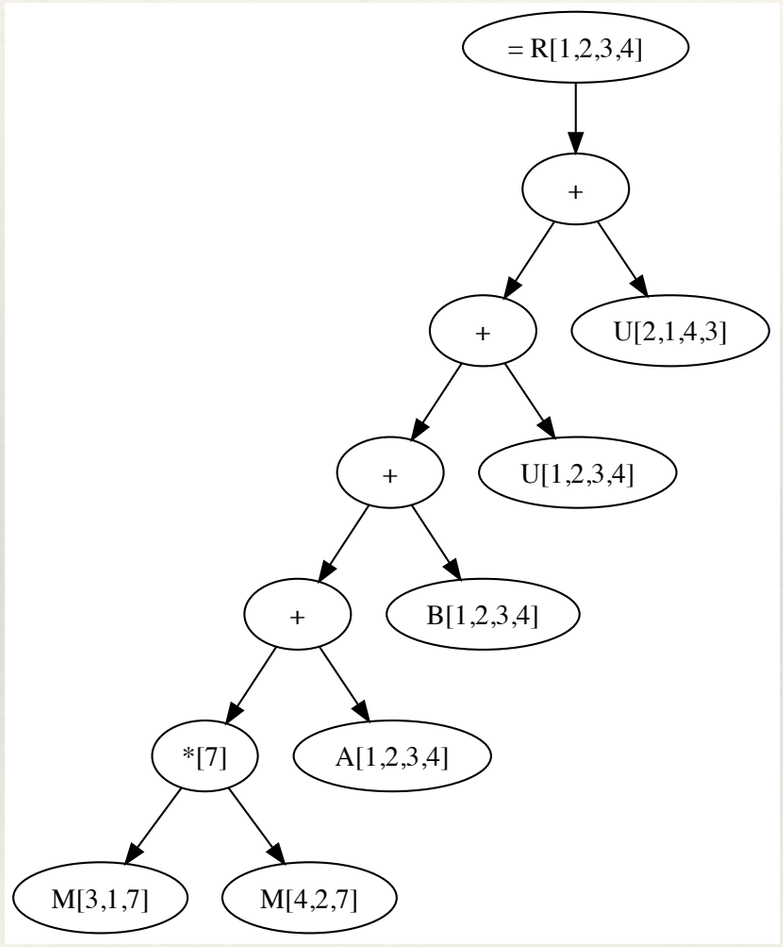
$$C_{ij}^{ab} = -\sum_{kc}\left(\sum_{Q}\mathrm{Uoo}_{ki}^{Q}\mathrm{Uvv}_{ac}^{Q} - 0.5\sum_{ld}V_{kdlc}t_{li}^{ad}\right)t_{kj}^{bc}$$

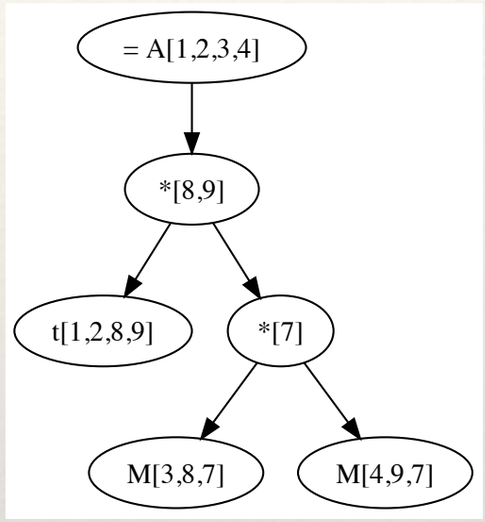
github.com/frobnitzem/slack

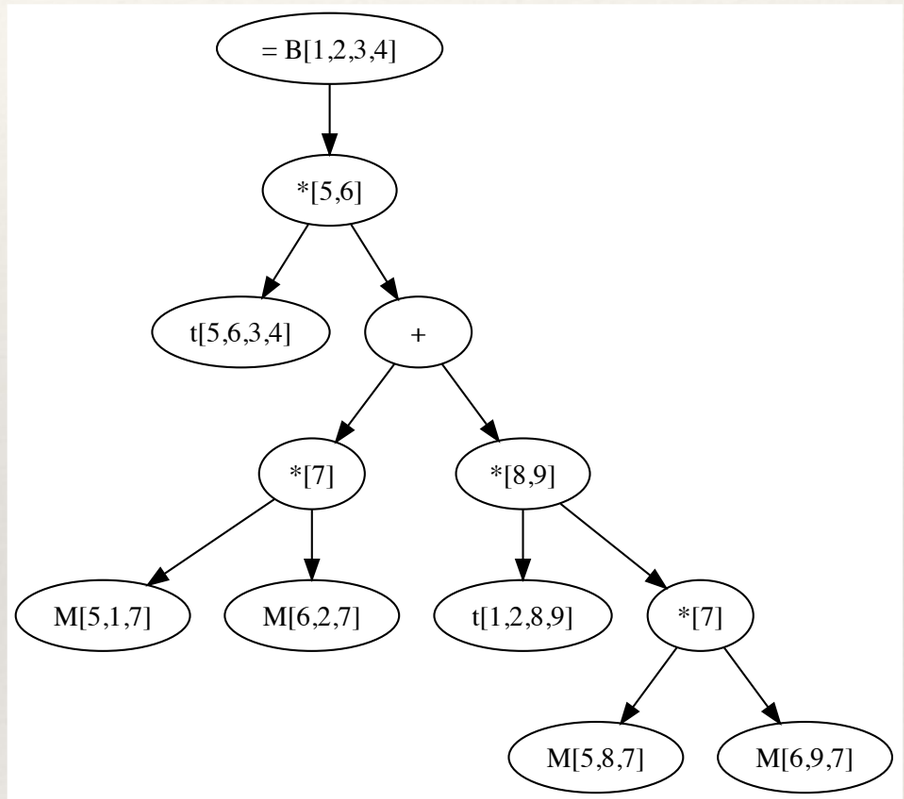
Example Use

- CCSD (FNO) residual

DePrince and Sherrill, JCTC 9:2687, 2013.







$$R_{ij}^{ab} = \sum_Q M_{ai}^Q M_{bj}^Q + A_{ij}^{ab} + B_{ij}^{ab} + U_{ij}^{ab} + U_{ji}^{ba}$$

$$A_{ij}^{ab} = \sum_{cd} t_{ij}^{cd} \sum_Q M_{ac}^Q M_{bd}^Q$$

$$B_{ij}^{ab} = \sum_{kl} t_{kl}^{ab} \left(\sum_Q M_{ki}^Q M_{lj}^Q + \sum_{cd} t_{ij}^{cd} \sum_Q M_{kc}^Q M_{ld}^Q \right)$$

Example Use

$$T = T_{\text{serial}} + T_{\text{parallel}}/P$$

- Amdahl's Law:
  - Simultaneous operations can be parallelized
  - Individual Matrix Multiplications can be parallelized
  - T_{serial} is just scheduler overhead!

- Many Projects Related to Scheduling:
  - Intel: CILK, TBB
  - Heterogeneous Computing: QUARK, StarPU
  - Cluster Computing: Swift/Fireworks/Tigres/Taskfarmer and Condor/Pegasus

Workflows: https://www.nersc.gov/users/data-analytics/workflow-tools/

Kepler: https://kepler-project.org/ See S3D, Chen et. al., Comput. Sci. & Disc. 2, 2009.

Condor: As of April, 2015. Metascheduling is no longer supported in XSEDE

Pegasus: https://pegasus.isi.edu/documentation/xsede.php

Internal Representation

- Code parses to operation DAG
- Nodes can be programmatically added
  - e.g. computer algebra system
- 2 node types:
  - operation (Dot/Add/Scale)
  - fixed input (Base)
    - includes empty (zero)

```
struct Ast_s {
    enum AstType type;

    union {
        struct Base   *base;
        struct Dot    *dot;
        struct Add    *add;
        struct Scale  *scale;
        char   *ref;
    };
};
```

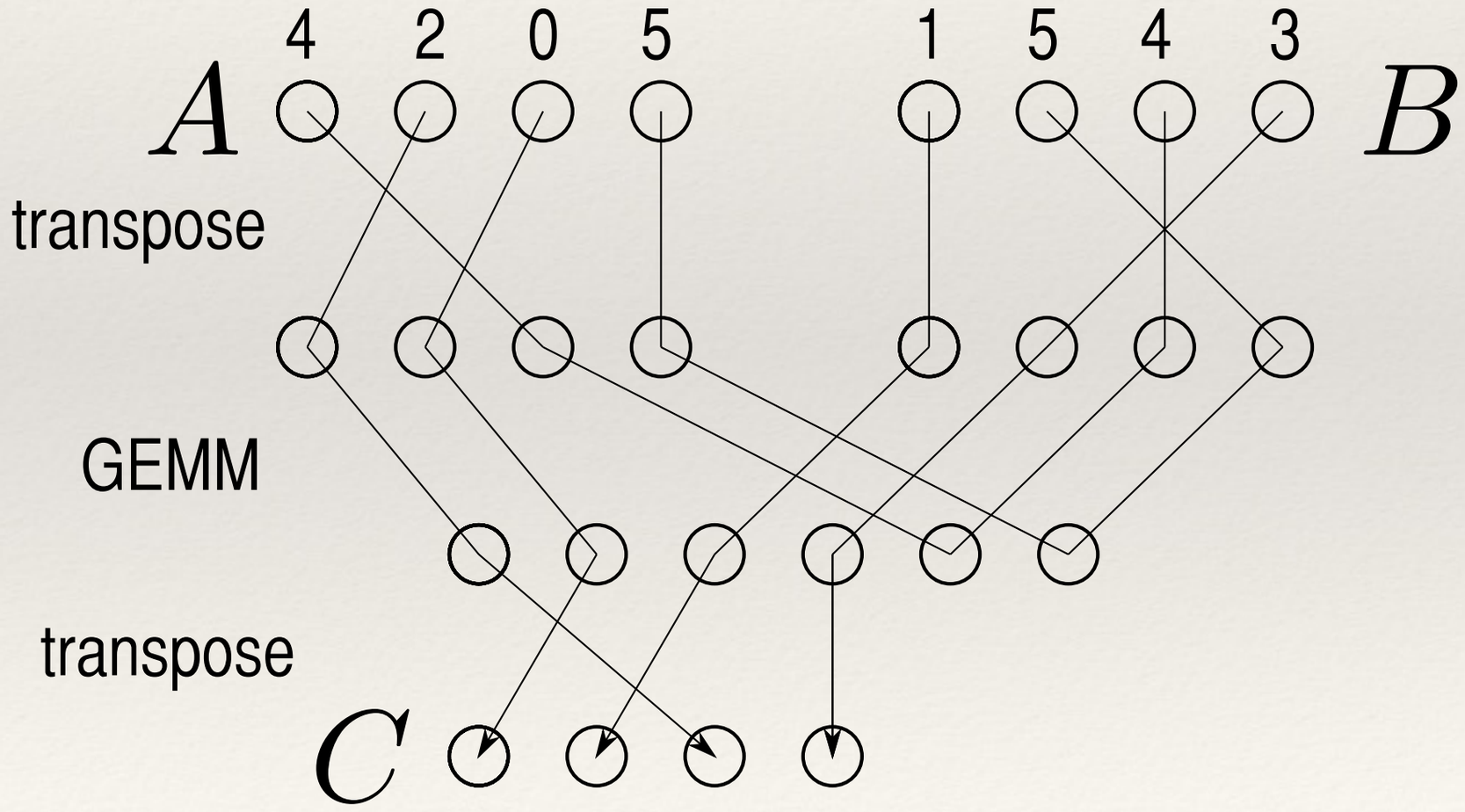
github.com/frobnitzem/slack

Outline

- Definitions
- High-level interface
  - Example uses
  - Internal representation
- Low-level kernel details
  - Timings of Conventional Algorithms
  - Loop Nesting Optimization
  - Code Generator Operation

Conventional Algorithm

- Transpose-_GEMM-Transpose



Conventional Algorithm

- But CUDA GEMM is really fast!

Kurzak, Tomov, and Dongarra, "Autotuning GEMMS for Fermi," SC11.

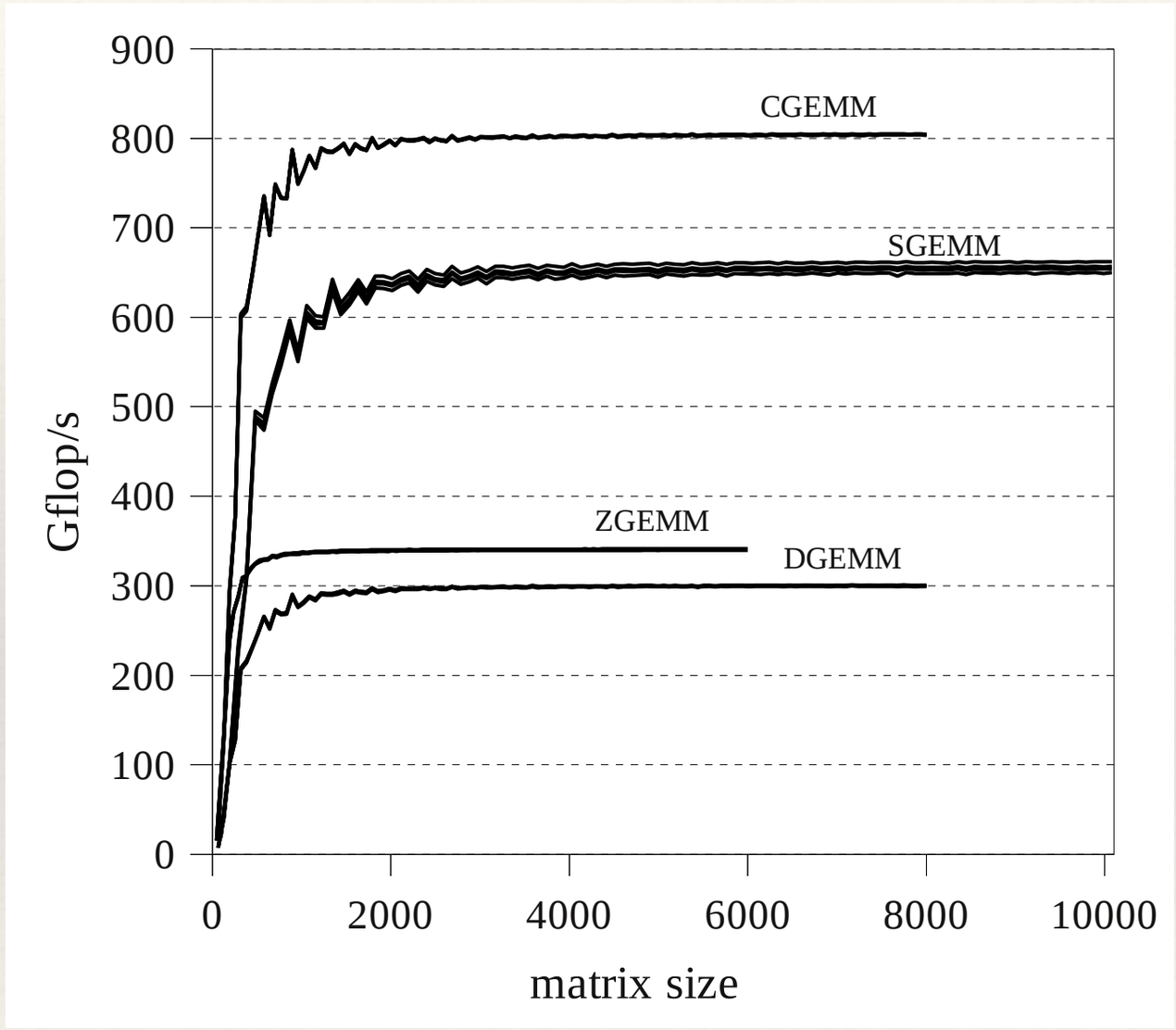


Figure 6: GEMM Performance on a 1.147 GHz Fermi GPU

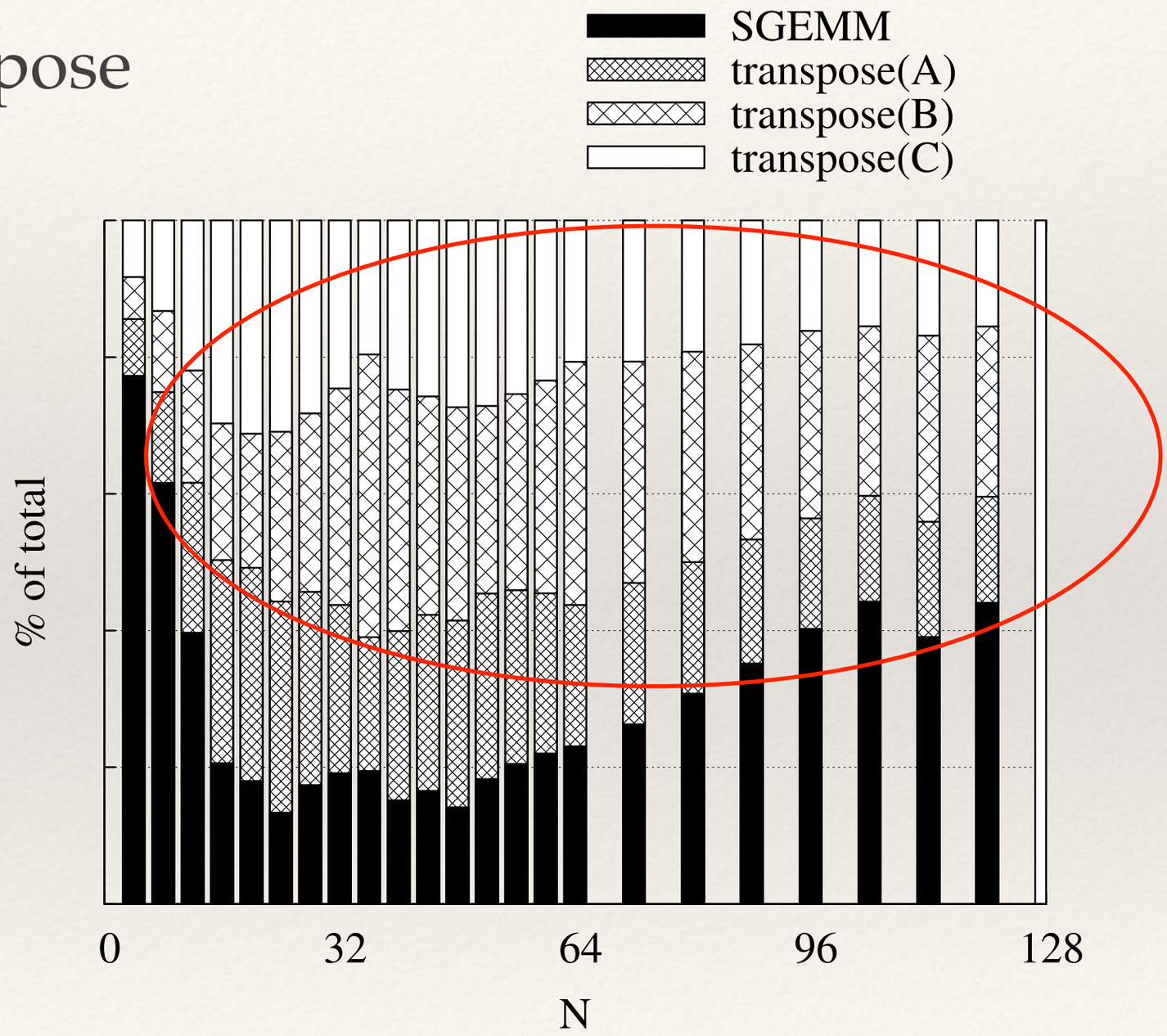
Conventional Algorithm

- … so fast the transpose can't keep up.

>50% of total time!

Rogers, XSEDE16
1.38 GHz GTX980

SGEMM peak: 4 Tflops!
DGEMM peak: 150 Gflops

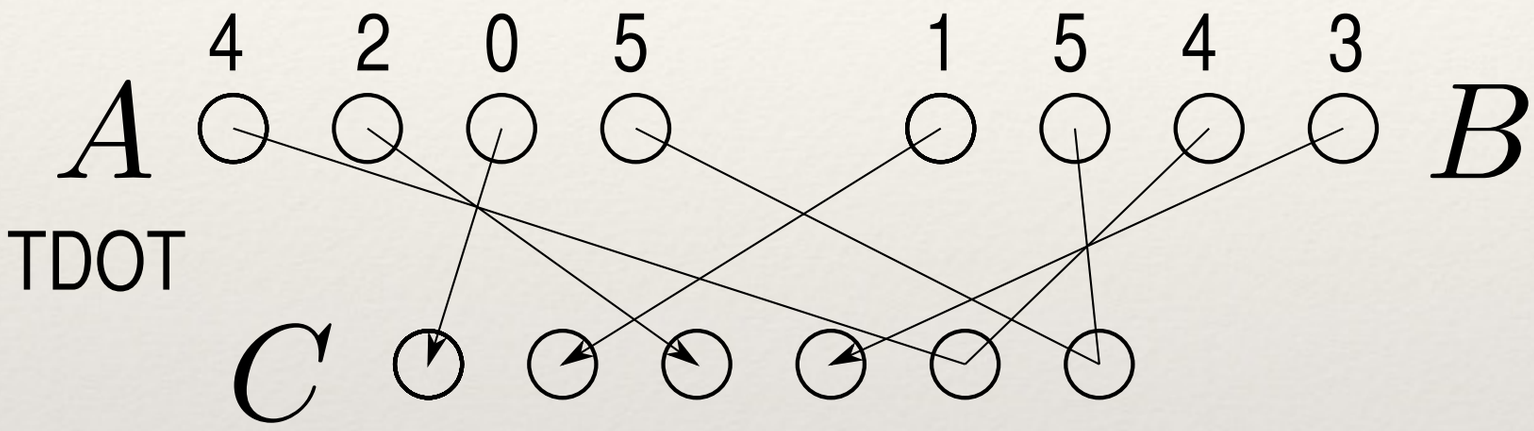


Alternative

- Hypothesis:
  - Speed of low-level kernel is due to groups of sequential reads and optimal cache usage.
  - This can be achieved just as well for indexed tensors.
  - Key requirement is optimizing access patterns.

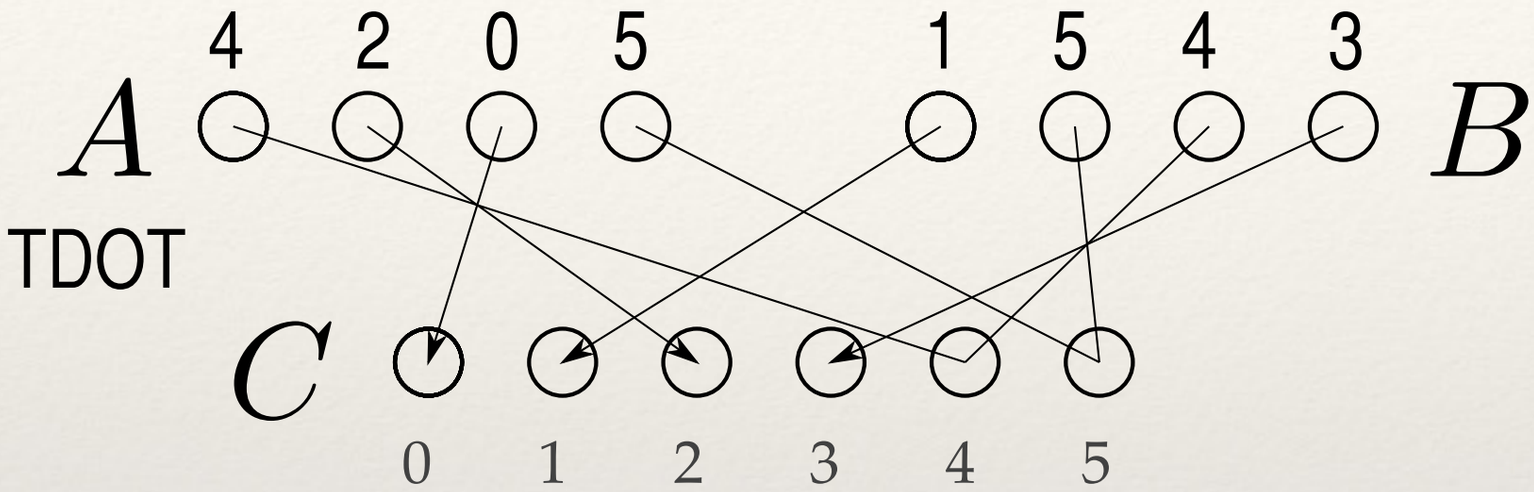
Alternative

… or get a better transpose:
(https://github.com/DmitryLyakh/TAL_SH)



- Direct Multiplication
  - Skip transpose steps (and memory, work, etc.)
  - requires complicated access pattern

Critical Concept



- Permutation = function from input to output "order"
  - $\Pi_A = (4,2,0,5)$
  - $\Pi_B = (1,5,4,3)$
- Technical analysis is all about combinatorics.

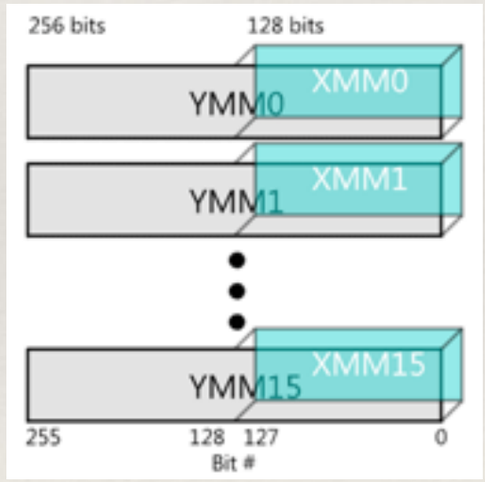
Optimal Flop and Cache Usage?

Maxwell GPU

Figure 2: GM204 Full-chip block diagram

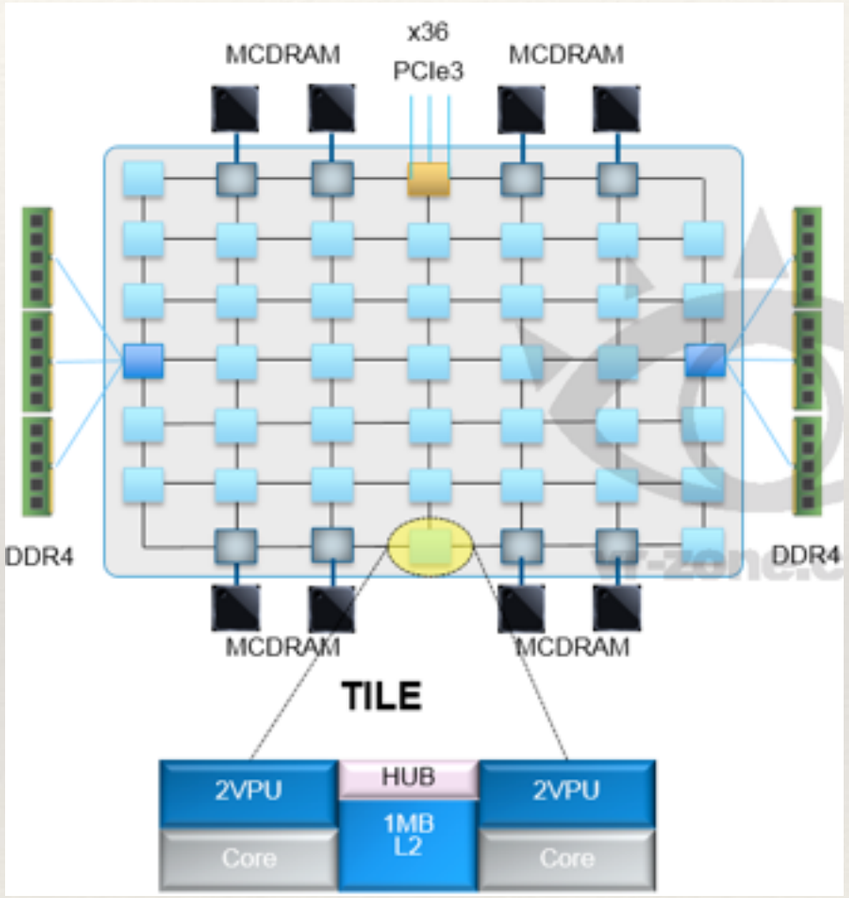
NVIDIA whitepaper
32 float registers per SMM
32 SMM-s (1024 flop/cyc)

Xeon AVX256



Intel AVX Docs
16, 4-double registers
(64 flop/cyc)

Knight's Landing



VR-zone
72 cores with 2xAVX512
(2304 flop/cyc)

MAGMA GPU Kernel

Achieves optimality by finding a wide data path.

Figure 2 shows how a $M_{blk} \times N_{blk} \times K_{blk}$ partial result is produced in one iteration of the outermost loop of the thread block's code. The figure shows parallelization at the thread level. The light shade shows the shape of the thread grid and the dark shade shows the elements involved in the operations of a single thread.

Figure 3 shows the operation from the perspective of one thread. Each thread streams in elements of A and B from the shared memory to the registers and accumulates the matrix multiplication results in C, residing in registers. This is the ideal situation. Whether it actually is the case depends on the actual tiling factors at each level. Whenever the compiler runs out of registers, register spills to memory will occur. (Which on Fermi is mitigated to some extent by the existence of the L1 cache.)

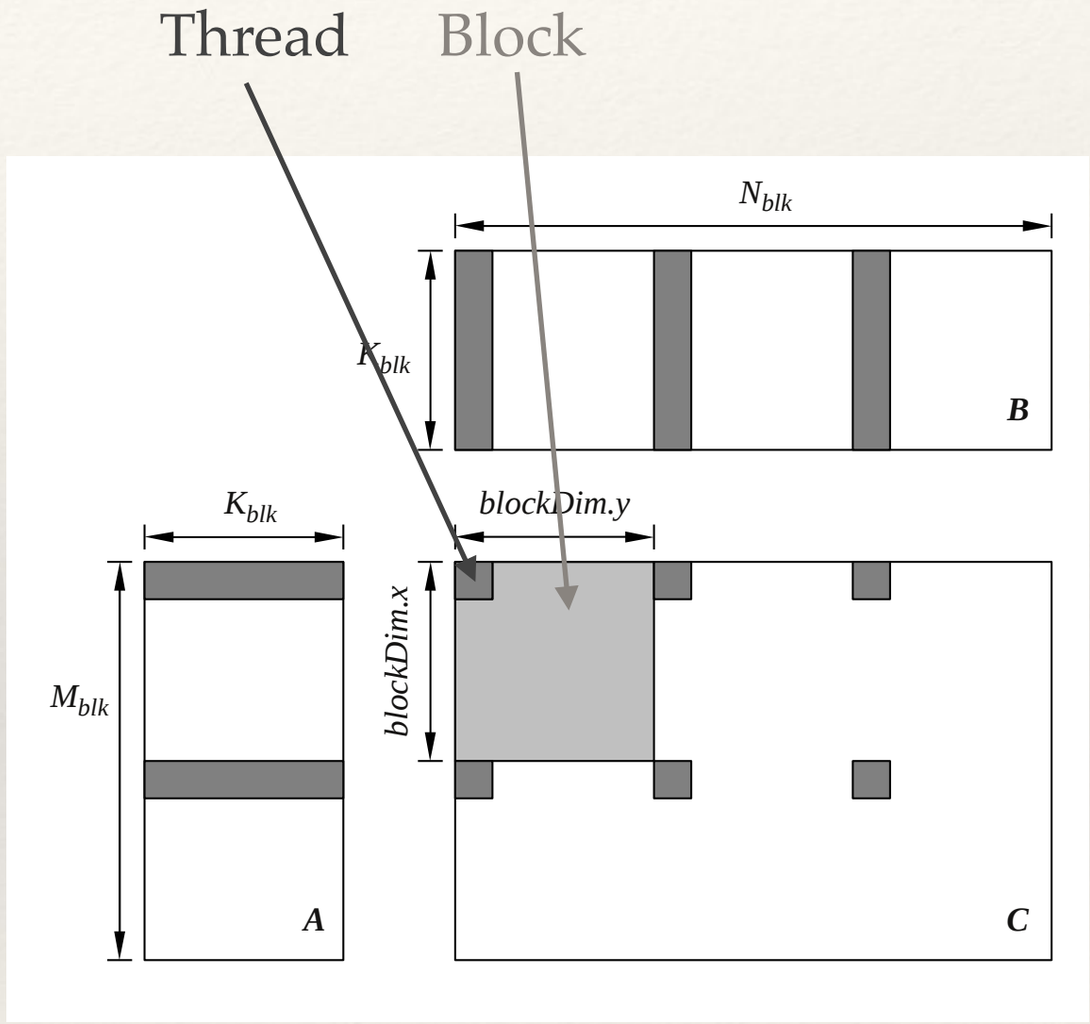


Figure 2: GEMM at the block level.

Nath, Tomov, Dongarra, "Accelerating GPU kernels for dense linear algebra." VECPAR10.

Kurzak, Tomov, and Dongarra, "Autotuning GEMMS for Fermi," SC11.

SLACK GPU Kernel

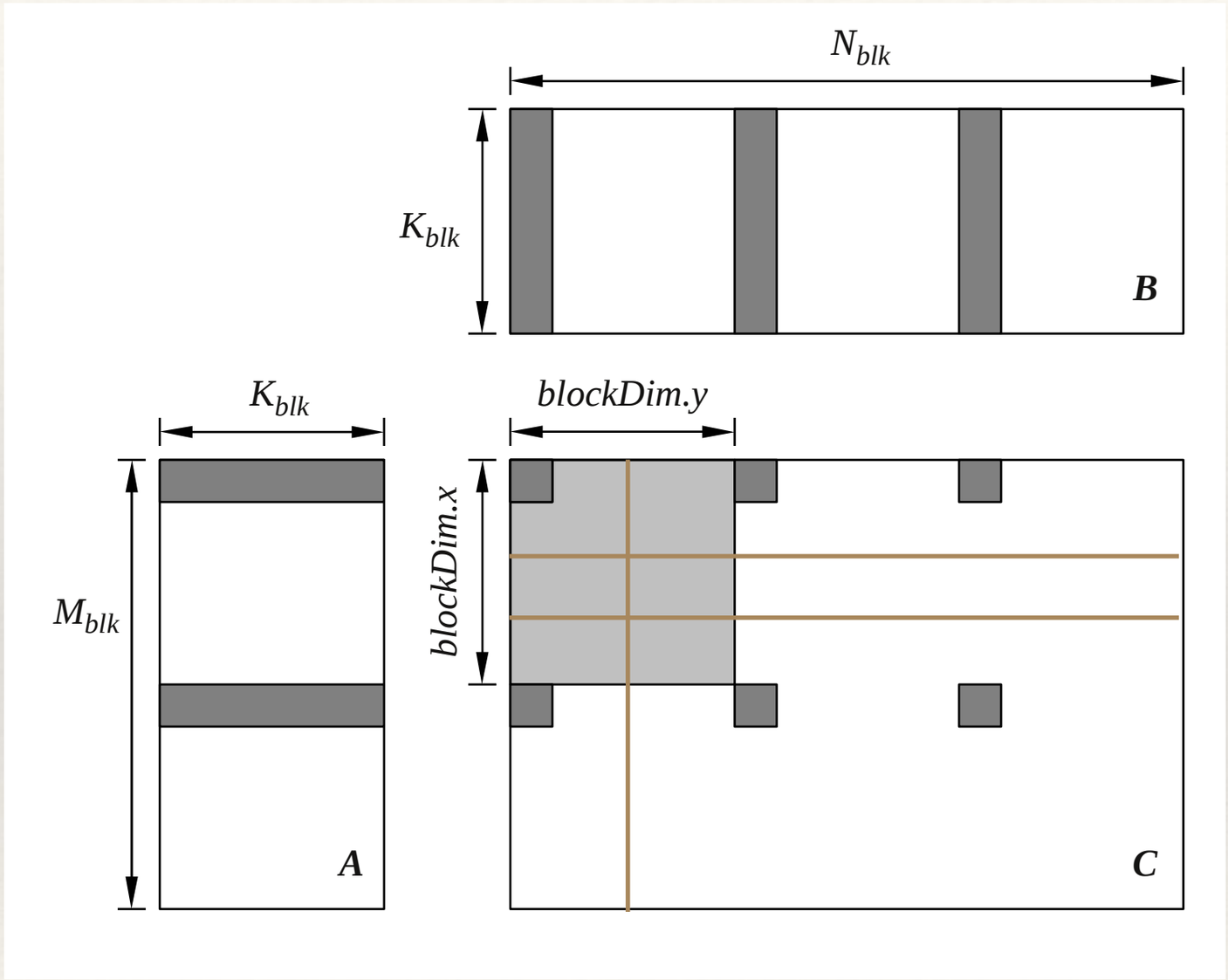


Figure 2: GEMM at the block level.

Kurzak, Tomov, and Dongarra, "Autotuning GEMMS for Fermi," SC11.

- Same kernel structure
- Now all inner, outer indices are tensor indexed.

Code Generator Secret Sauce

- Index Class
  - sequential -> multi-index
- Splitting Methods
  - Known shape -> explicit calculation
    - (inner loops)
  - Unknown shape -> symbolic calculation
    - (outer loops)

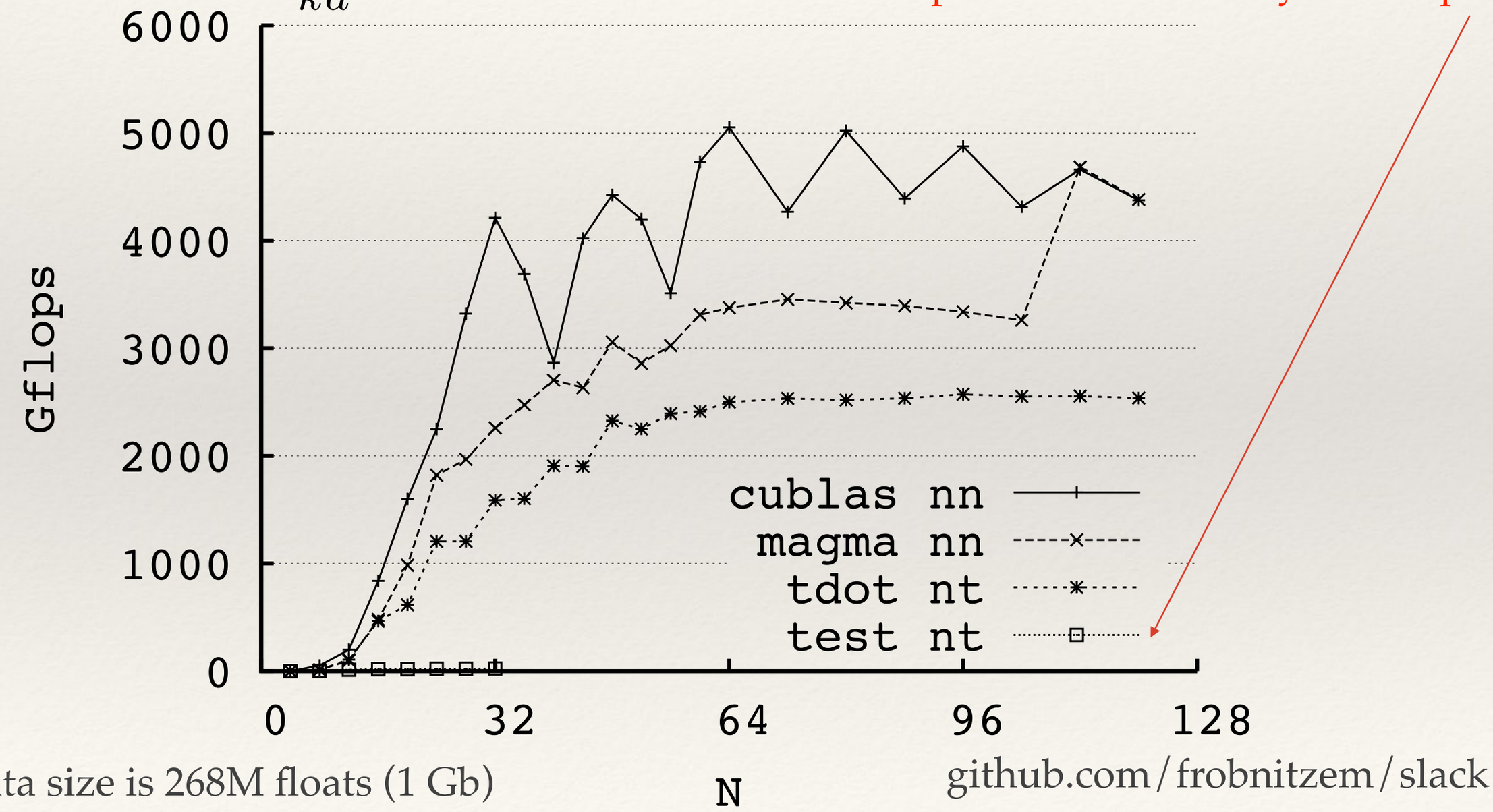
```
>>> A = TensIdx((8,4,3))
>>> A[1,2,0]  = 1*(4*3) + 2*(3)
18
>>> len(A)
96
>>> g = A.__iter__()
>>> g.next()
[0, 0, 0]
>>> g.next()
[0, 0, 1]

>>> B = TensIdx_sym("B", [0,1,2])
>>> B["a","b","c"]
' a*sB_stride0 +  b*sB_stride1 +  c'
```

github.com/frobnitzem/slack

Timing Results (all single precision)

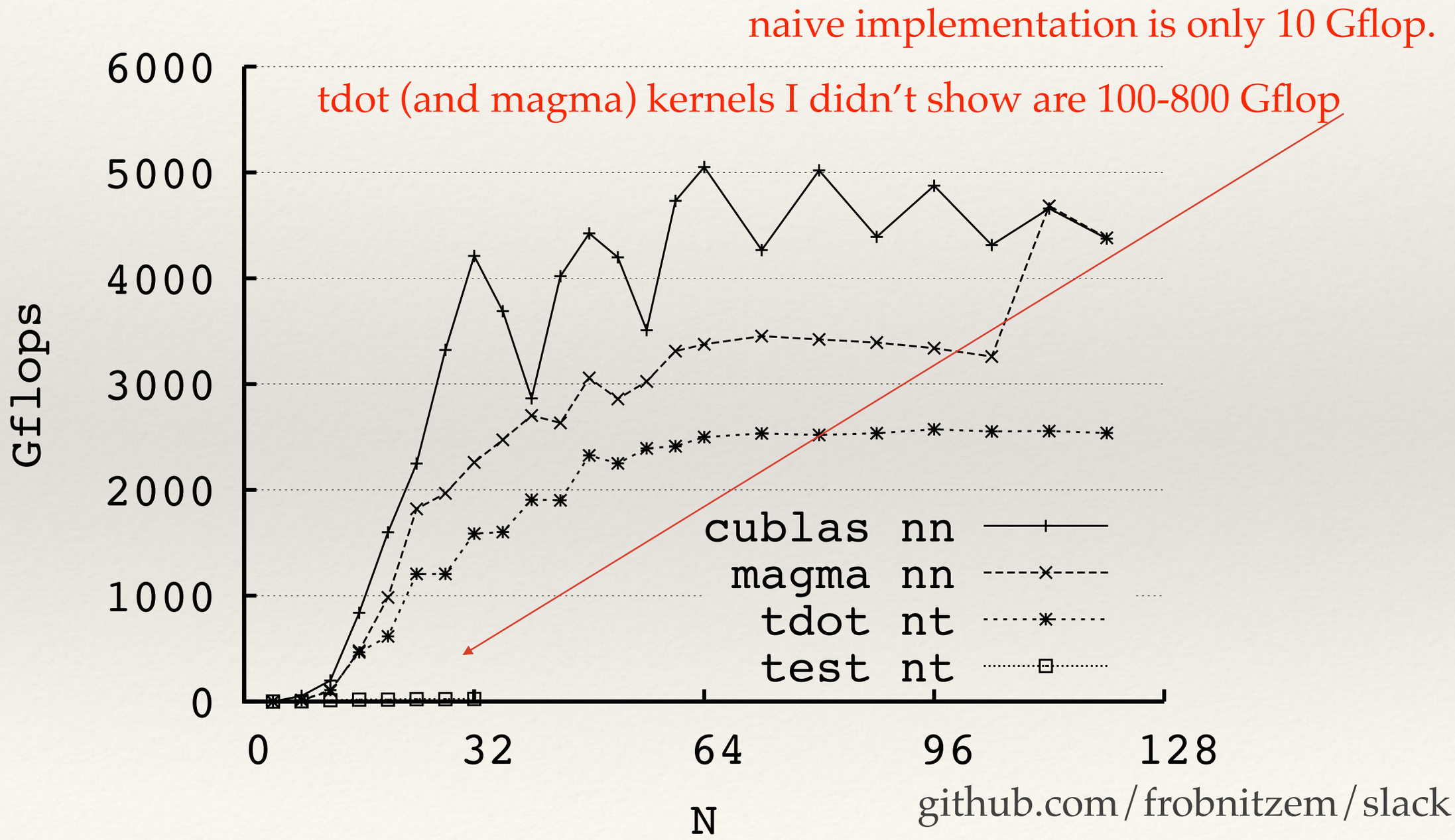
$$C_{ijab} = \sum_{kd} A_{iakd} B_{kdjb}$$



Max Data size is 268M floats (1 Gb)

github.com/frobnitzem/slack

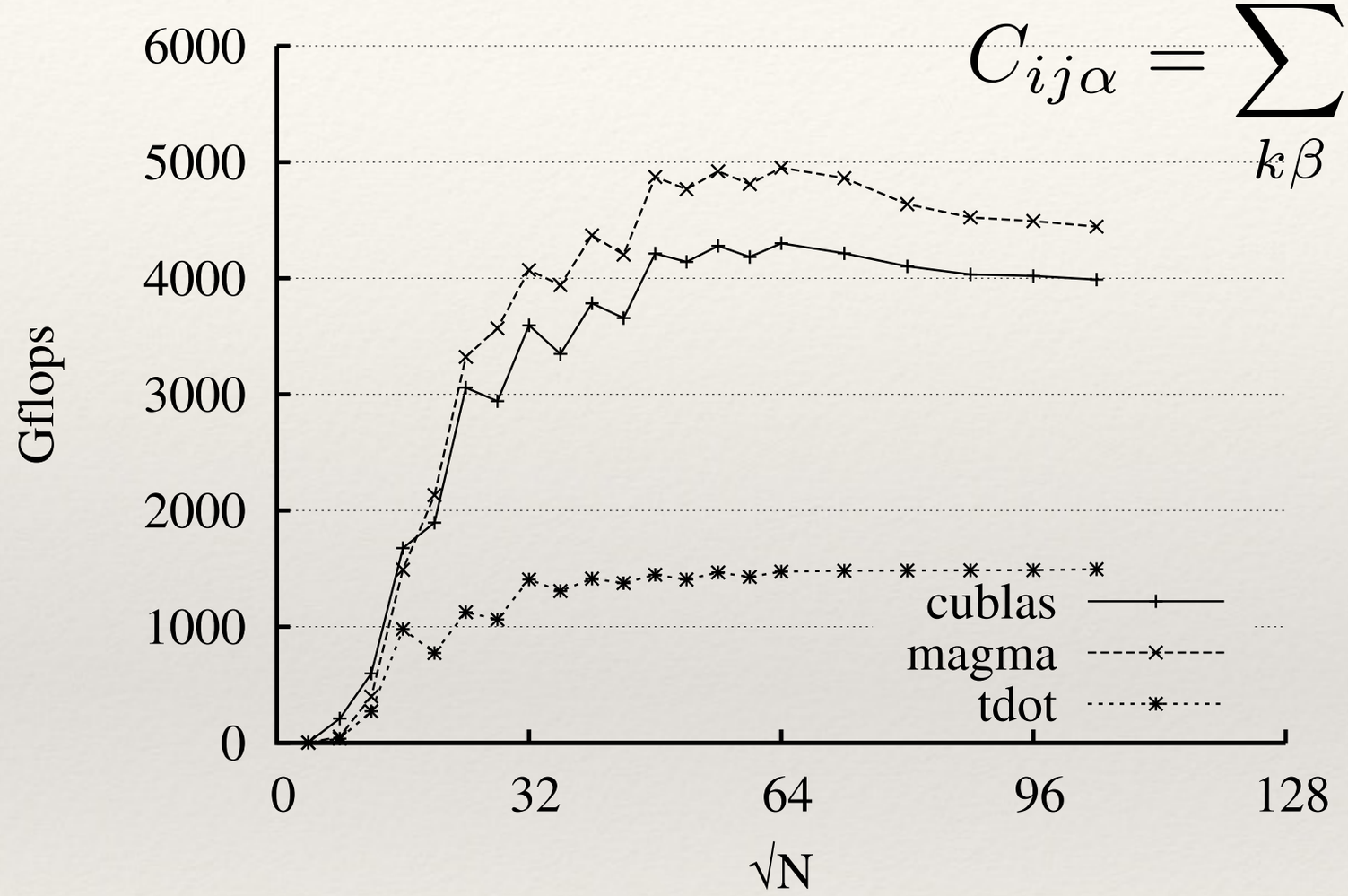
Timing Results (all single precision)



github.com/frobnitzem/slack

Timing Results

$$C_{ij\alpha} = \sum_{k\beta} A_{ik\alpha\beta} B_{kj\beta}$$

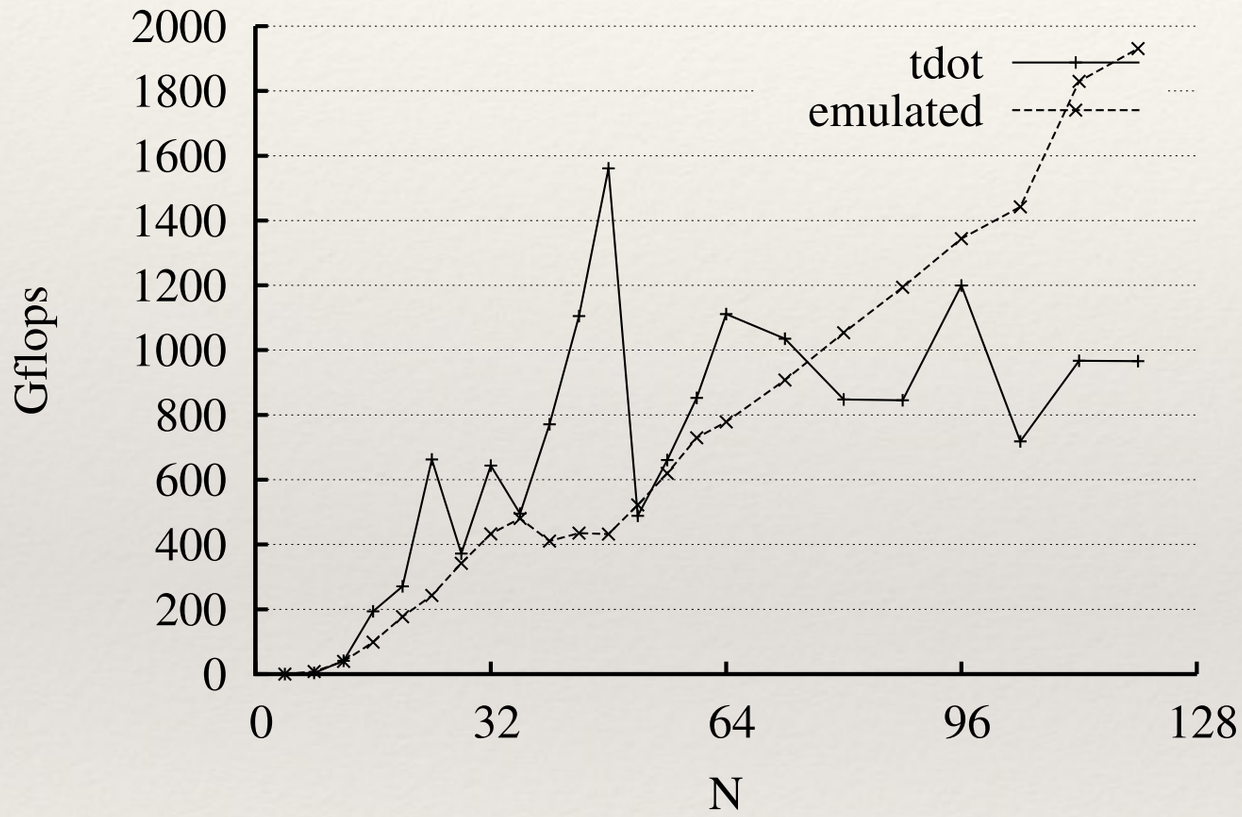


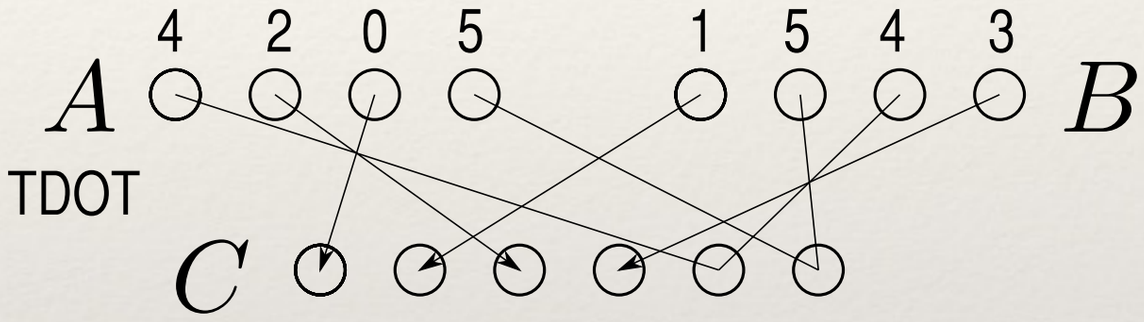
Same number of ops, but 1/2 data movement
(tdot used full data movement)

max at 5 teraflops, near device peak
bus speed is the limiting factor!

github.com/frobnitzem/slack

Timing Results (all single precision)





Note max at 2 teraflops.

(emulation speed does not account for data movement.)

github.com/frobnitzem/slack

Conclusions

- Direct tensor contraction can be made efficient.
  - Eliminates space, communication, and run-time for transpose.
  - Exposes simple tensor primitives.
    - Memory access is now the critical bottleneck on coprocessors.
- High-level code generation strategies can now save developer time and reduce errors.
  - But we're 3 steps behind: need high-level frameworks, matrix -specific interface, and computer algebra assistance!

Future Work

- Apply to more problems!
  - Exploit sparsity and symmetry, extend interface.
- Improve connections to parallel schedulers.
  - Enable distributed, large calculations.
- Automate search for optimal kernel parameters.
  - FFTW/Magma style.
- Extend DAG generation interface (more code gen).
  - Input using a more general programming language.

Acknowledgments

Collaborators:

Jeff Hammond (Intel)

Eugene DePrince (FSU)

Brian Space (USF)

Sameer Varma (USF)

Students:

Corbin Rodier

Andres Saez

Devin Thorton

Funding:

NSF MRI CHE-1531590

XSEDE TG-ASC130043