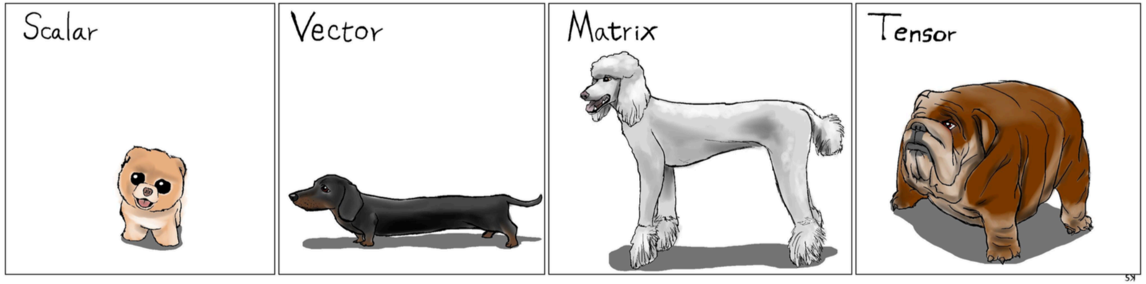
PPAM 17

Tensor Contractions and Low-Communication FFT

Cris Cecka
NVIDIA Research, Stanford University

NVIDIA

TENSOR CONTRACTIONS



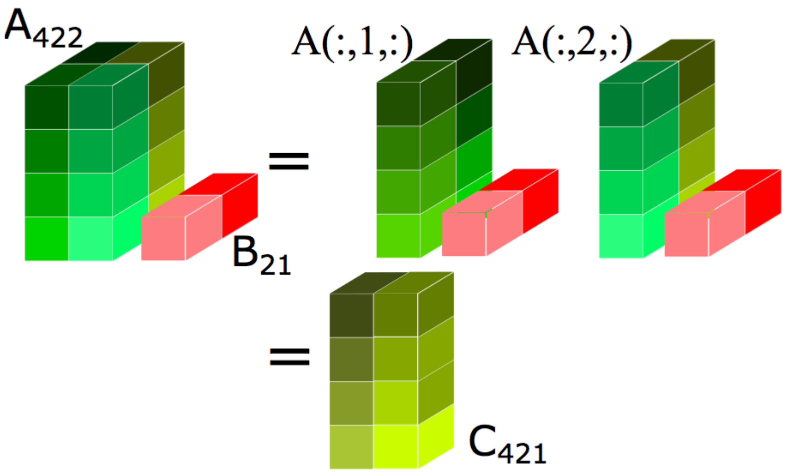
Modern data is inherently multi-dimensional.

NVIDIA especially interested due to ML.

Portability, BLAS spec, efficiency.

TENSOR CONTRACTIONS

$$C_{\mathcal{C}} = A_{\mathcal{A}}\, B_{\mathcal{B}}$$



$$\text{e.g.} \qquad C_{mnp} = A_{mnk}\, B_{kp}$$

- Core primitive of multilinear algebra
- BLAS level 3 — unbounded compute intensity.

What do we have?

Tensor computation libraries

1. Arbitrary/restricted tensor operation of any order and dimension
    1. Tensortoolbox (Matlab)
    2. FTensor (C++)
    3. Cyclops (C++)
    4. BTAS (C++)
    5. All the Python...

Efficient computing frame

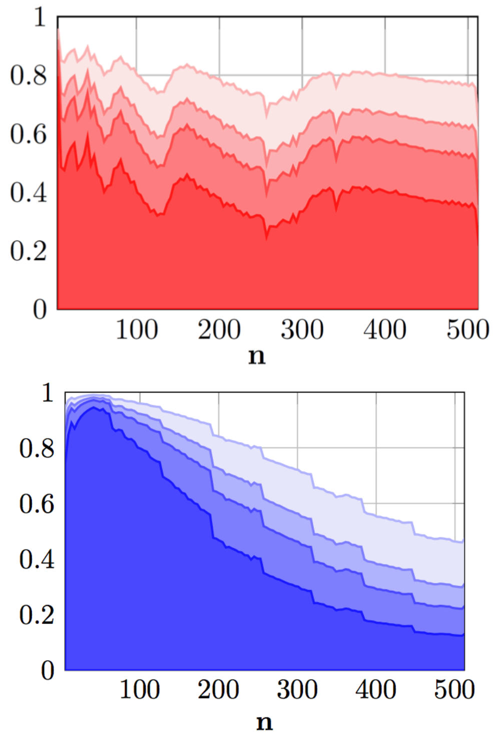
1. Static analysis solutions
    1. PPCG [ISL] (polyhedral)
    2. TCE (DSL)
2. Parallel and distributed primitives
    1. BLAS, cuBLAS
    2. BLIS, BLASX, cuBLASXT

TENSOR LIBRARIES

Explicit permutation dominates.

Consider $C_{mnp} = A_{km} B_{pkn}$.

1. $A_{km} \rightarrow A_{mk}$
2. $B_{pkn} \rightarrow B_{kpn}$
3. $C_{mnp} \rightarrow C_{mpn}$
4. $\boxed{C_{m(pn)} = A_{mk} B_{k(pn)}}$
5. $C_{mpn} \rightarrow C_{mnp}$



(Top) CPU. (Bottom) GPU. The fraction of time spent in copies/transpositions. Lines are shown with 1, 2, 3, and 6 transpositions.

EXISTING PRIMITIVES

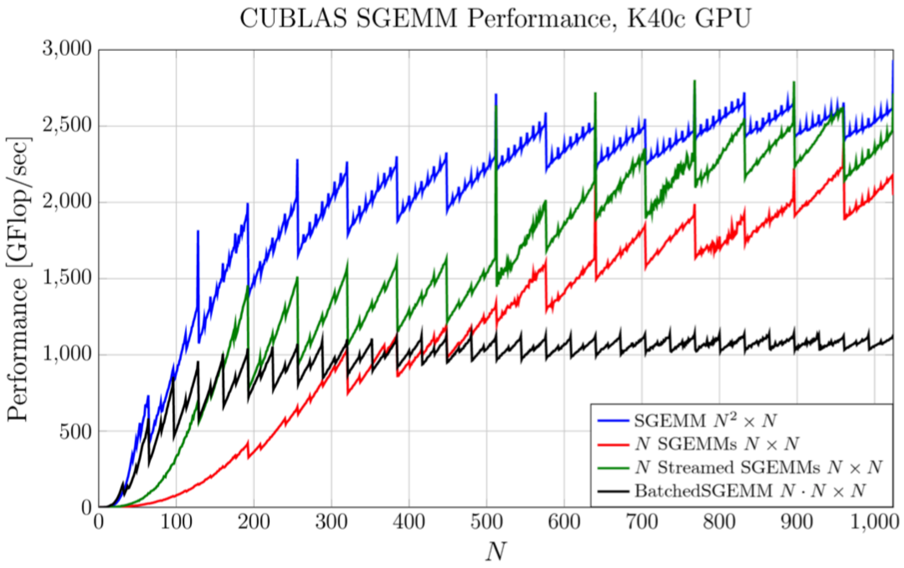
- GEMM
  - Suboptimal for many small matrices
- Pointer-to-pointer BatchedGEMM
  - Available since MKL 13.1b and cuBLAS 4.0

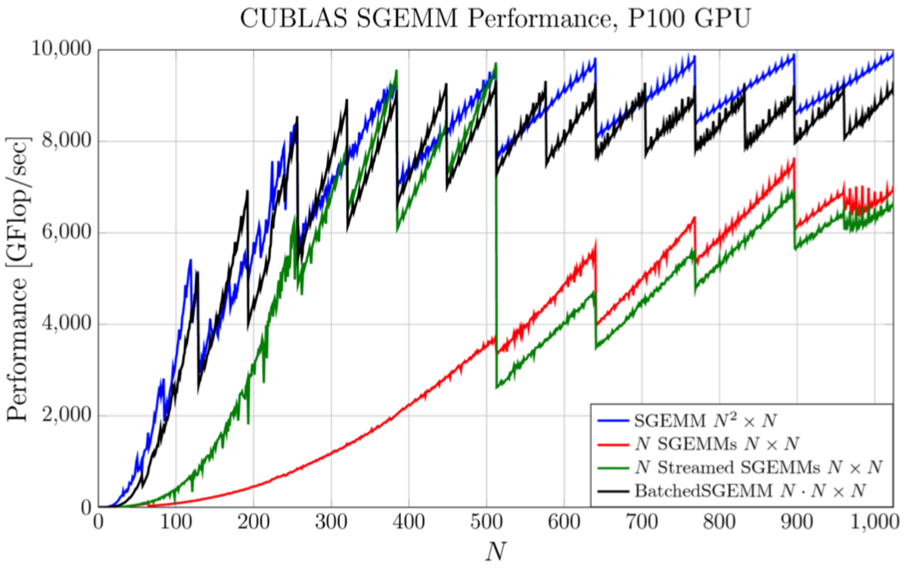
$$C[p] = \alpha \, \mathrm{op}(A[p]) \, \mathrm{op}(B[p]) + \beta \, C[p]$$

```
cublas<T>gemmBatched(cublasHandle_t handle,
                     cublasOperation_t transA, cublasOperation_t transB,
                     int M, int N, int K,
                     const T* alpha,
                     const T** A, int ldA,
                     const T** B, int ldB,
                     const T* beta,
                     T** C, int ldC,
                     int batchCount)
```

EXISTING PRIMITIVES

Pointer-to-Pointer BatchedGEMM

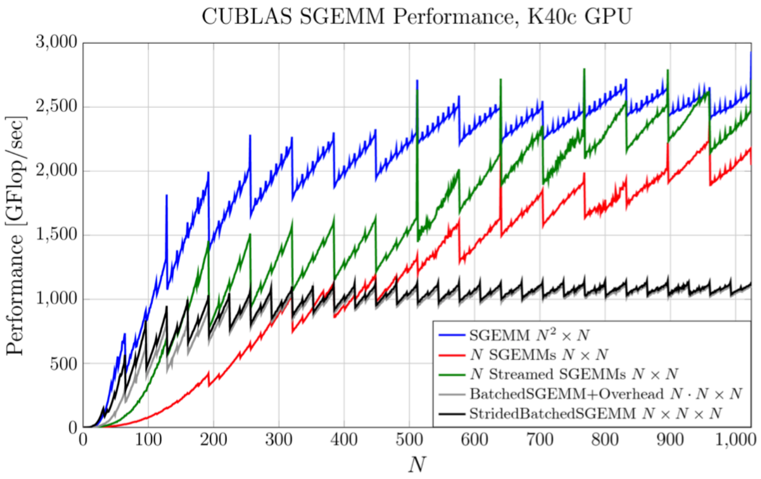


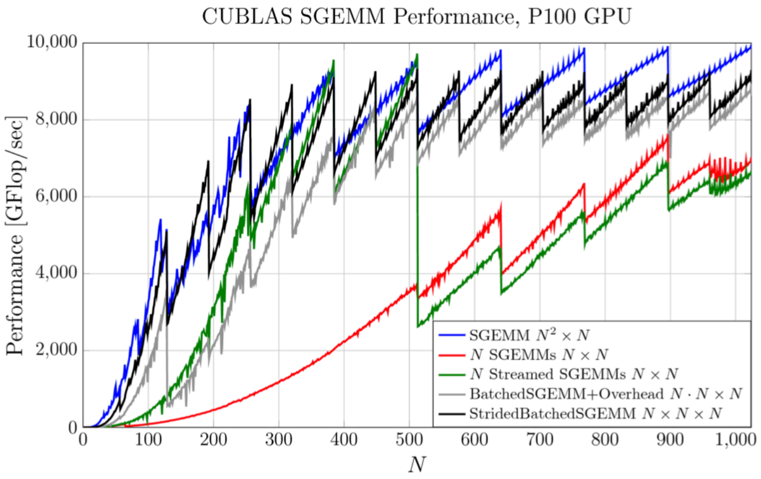


EXISTING PRIMITIVES

Pointer-to-Pointer BatchedGEMM

Except actually...





Solution: StridedBatchedGEMM

STRIDED BATCHED GEMM

```
cublas<T>gemmStridedBatched(cublasHandle_t handle,
                            cublasOperation_t transA, cublasOperation_t transB,
                            int M, int N, int K,
                            const T* alpha,
                            const T* A, int ldA1, int strideA,
                            const T* B, int ldB1, int strideB,
                            const T* beta,
                            T* C, int ldC1, int strideC,
                            int batchCount)
```

- Common use case for pointer-to-pointer batched GEMM
- No pointer-to-pointer data structure or overhead
- Performance on par with pure GEMM (P100 and beyond)

CONTRACTIONS

$$C_{mnp} = A_{**} \times B_{***}$$

$$C_{mnp} \equiv C[m + n \cdot \mathrm{ldC1} + p \cdot \mathrm{ldC2}]$$

: Single GEMM

(Provided compact layout)

Case	Contraction	Kernel1	Kernel2	Case	Contraction	Kernel1	Kernel2
1.1	$A_{mk}B_{knp}$	$C_{m(np)} = A_{mk}B_{k(np)}$	$C_{mn[p]} = A_{mk}B_{kn[p]}$	4.1	$A_{kn}B_{kmp}$	$C_{mn[p]} = B^{\top}_{km[p]}A_{kn}$	
1.2	$A_{mk}B_{kpn}$	$C_{mn[p]} = A_{mk}B_{k[p]n}$	$C_{m[n]p} = A_{mk}B_{kp[n]}$	4.2	$A_{kn}B_{kpm}$	$C_{mn[p]} = B^{\top}_{k[p]m}A_{kn}$	
1.3	$A_{mk}B_{nkp}$	$C_{mn[p]} = A_{mk}B^{\top}_{nk[p]}$		4.3	$A_{kn}B_{mkp}$	$C_{mn[p]} = B_{mk[p]}A_{kn}$	
1.4	$A_{mk}B_{pkn}$	$C_{m[n]p} = A_{mk}B^{\top}_{pk[n]}$		4.4	$A_{kn}B_{pkm}$		
1.5	$A_{mk}B_{npk}$	$C_{m(np)} = A_{mk}B^{\top}_{(np)k}$	$C_{mn[p]} = A_{mk}B^{\top}_{n[p]k}$	4.5	$A_{kn}B_{mpk}$	$C_{mn[p]} = B_{m[p]k}A_{kn}$	
1.6	$A_{mk}B_{pnk}$	$C_{m[n]p} = A_{mk}B^{\top}_{p[n]k}$		4.6	$A_{kn}B_{pmk}$		
2.1	$A_{km}B_{knp}$	$C_{m(np)} = A^{\top}_{km}B_{k(np)}$	$C_{mn[p]} = A^{\top}_{km}B_{kn[p]}$	5.1	$A_{pk}B_{kmn}$	$C_{(mn)p} = B^{\top}_{k(mn)}A^{\top}_{pk}$	$C_{m[n]p} = B^{\top}_{km[n]}A^{\top}_{pk}$
2.2	$A_{km}B_{kpn}$	$C_{mn[p]} = A^{\top}_{km}B_{k[p]n}$	$C_{m[n]p} = A^{\top}_{km}B_{kp[n]}$	5.2	$A_{pk}B_{knm}$	$C_{m[n]p} = B^{\top}_{k[n]m}A^{\top}_{pk}$	
2.3	$A_{km}B_{nkp}$	$C_{mn[p]} = A^{\top}_{km}B^{\top}_{nk[p]}$		5.3	$A_{pk}B_{mkn}$	$C_{m[n]p} = B_{mk[n]}A^{\top}_{pk}$	
2.4	$A_{km}B_{pkn}$	$C_{m[n]p} = A^{\top}_{km}B^{\top}_{pk[n]}$		5.4	$A_{pk}B_{nkm}$		
2.5	$A_{km}B_{npk}$	$C_{m(np)} = A^{\top}_{km}B^{\top}_{(np)k}$	$C_{mn[p]} = A^{\top}_{km}B^{\top}_{n[p]k}$	5.5	$A_{pk}B_{mnk}$	$C_{(mn)p} = B_{(mn)k}A^{\top}_{pk}$	$C_{m[n]p} = B_{m[n]k}A^{\top}_{pk}$
2.6	$A_{km}B_{pnk}$	$C_{m[n]p} = A^{\top}_{km}B^{\top}_{p[n]k}$		5.6	$A_{pk}B_{nmk}$		
3.1	$A_{nk}B_{kmp}$	$C_{mn[p]} = B^{\top}_{km[p]}A^{\top}_{nk}$		6.1	$A_{kp}B_{kmn}$	$C_{(mn)p} = B^{\top}_{k(mn)}A_{kp}$	$C_{m[n]p} = B^{\top}_{km[n]}A_{kp}$
3.2	$A_{nk}B_{kpm}$	$C_{mn[p]} = B^{\top}_{k[p]m}A^{\top}_{nk}$		6.2	$A_{kp}B_{knm}$	$C_{m[n]p} = B^{\top}_{k[n]m}A_{kp}$	
3.3	$A_{nk}B_{mkp}$	$C_{mn[p]} = B_{mk[p]}A^{\top}_{nk}$		6.3	$A_{kp}B_{mkn}$	$C_{m[n]p} = B_{mk[n]}A_{kp}$	
3.4	$A_{nk}B_{pkm}$			6.4	$A_{kp}B_{nkm}$		
3.5	$A_{nk}B_{mpk}$	$C_{mn[p]} = B_{m[p]k}A^{\top}_{nk}$		6.5	$A_{kp}B_{mnk}$	$C_{(mn)p} = B_{(mn)k}A_{kp}$	$C_{m[n]p} = B_{m[n]k}A_{kp}$
3.6	$A_{nk}B_{pmk}$			6.6	$A_{kp}B_{nmk}$		

CONTRACTIONS

$$C_{mnp} = A_{**} \times B_{***}$$

$$C_{mnp} \equiv C[m + n \cdot \mathrm{ldC1} + p \cdot \mathrm{ldC2}]$$

: Single SB-GEMM
(Any layout)

Case	Contraction	Kernel1	Kernel2	Case	Contraction	Kernel1	Kernel2
1.1	$A_{mk}B_{knp}$	$C_{m(np)} = A_{mk}B_{k(np)}$	$C_{mn[p]} = A_{mk}B_{kn[p]}$	4.1	$A_{kn}B_{kmp}$	$C_{mn[p]} = B^\top_{km[p]}A_{kn}$	
1.2	$A_{mk}B_{kpn}$	$C_{mn[p]} = A_{mk}B_{k[p]n}$	$C_{m[n]p} = A_{mk}B_{kp[n]}$	4.2	$A_{kn}B_{kpm}$	$C_{mn[p]} = B^\top_{k[p]m}A_{kn}$	
1.3	$A_{mk}B_{nkp}$	$C_{mn[p]} = A_{mk}B^\top_{nk[p]}$		4.3	$A_{kn}B_{mkp}$	$C_{mn[p]} = B_{mk[p]}A_{kn}$	
1.4	$A_{mk}B_{pkn}$	$C_{m[n]p} = A_{mk}B^\top_{pk[n]}$		4.4	$A_{kn}B_{pkm}$		
1.5	$A_{mk}B_{npk}$	$C_{m(np)} = A_{mk}B^\top_{(np)k}$	$C_{mn[p]} = A_{mk}B^\top_{n[p]k}$	4.5	$A_{kn}B_{mpk}$	$C_{mn[p]} = B_{m[p]k}A_{kn}$	
1.6	$A_{mk}B_{pnk}$	$C_{m[n]p} = A_{mk}B^\top_{p[n]k}$		4.6	$A_{kn}B_{pmk}$		
2.1	$A_{km}B_{knp}$	$C_{m(np)} = A^\top_{km}B_{k(np)}$	$C_{mn[p]} = A^\top_{km}B_{kn[p]}$	5.1	$A_{pk}B_{kmn}$	$C_{(mn)p} = B^\top_{k(mn)}A^\top_{pk}$	$C_{m[n]p} = B^\top_{km[n]}A^\top_{pk}$
2.2	$A_{km}B_{kpn}$	$C_{mn[p]} = A^\top_{km}B_{k[p]n}$	$C_{m[n]p} = A^\top_{km}B_{kp[n]}$	5.2	$A_{pk}B_{knm}$	$C_{m[n]p} = B^\top_{k[n]m}A^\top_{pk}$	
2.3	$A_{km}B_{nkp}$	$C_{mn[p]} = A^\top_{km}B^\top_{nk[p]}$		5.3	$A_{pk}B_{mkn}$	$C_{m[n]p} = B_{mk[n]}A^\top_{pk}$	
2.4	$A_{km}B_{pkn}$	$C_{m[n]p} = A^\top_{km}B^\top_{pk[n]}$		5.4	$A_{pk}B_{nkm}$		
2.5	$A_{km}B_{npk}$	$C_{m(np)} = A^\top_{km}B^\top_{(np)k}$	$C_{mn[p]} = A^\top_{km}B^\top_{n[p]k}$	5.5	$A_{pk}B_{mnk}$	$C_{(mn)p} = B_{(mn)k}A^\top_{pk}$	$C_{m[n]p} = B_{m[n]k}A^\top_{pk}$
2.6	$A_{km}B_{pnk}$	$C_{m[n]p} = A^\top_{km}B^\top_{p[n]k}$		5.6	$A_{pk}B_{nmk}$		
3.1	$A_{nk}B_{kmp}$	$C_{mn[p]} = B^\top_{km[p]}A^\top_{nk}$		6.1	$A_{kp}B_{kmn}$	$C_{(mn)p} = B^\top_{k(mn)}A_{kp}$	$C_{m[n]p} = B^\top_{km[n]}A_{kp}$
3.2	$A_{nk}B_{kpm}$	$C_{mn[p]} = B^\top_{k[p]m}A^\top_{nk}$		6.2	$A_{kp}B_{knm}$	$C_{m[n]p} = B^\top_{k[n]m}A_{kp}$	
3.3	$A_{nk}B_{mkp}$	$C_{mn[p]} = B_{mk[p]}A^\top_{nk}$		6.3	$A_{kp}B_{mkn}$	$C_{m[n]p} = B_{mk[n]}A_{kp}$	
3.4	$A_{nk}B_{pkm}$			6.4	$A_{kp}B_{nkm}$		
3.5	$A_{nk}B_{mpk}$	$C_{mn[p]} = B_{m[p]k}A^\top_{nk}$		6.5	$A_{kp}B_{mnk}$	$C_{(mn)p} = B_{(mn)k}A_{kp}$	$C_{m[n]p} = B_{m[n]k}A_{kp}$
3.6	$A_{nk}B_{pmk}$			6.6	$A_{kp}B_{nmk}$		

FAST FOURIER TRANSFORM

$$[\mathbf{F}_N]_{ij} = e^{-2\pi \imath i j / N}$$

Ubiquitous primitive in computational sciences

Arguably one of the oldest and most well-studied

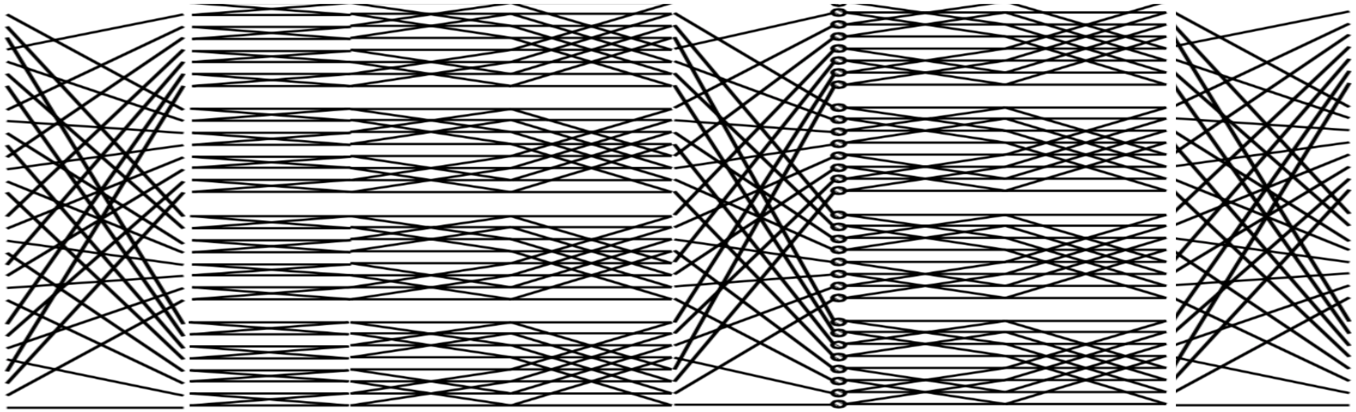
Performance on modern architectures…

SPLIT-RADIX FFT

Algorithm

$$\mathbf{F}_N = \mathbf{\Pi}_{M,P}\left(\mathbf{I}_M \otimes \mathbf{F}_P\right)\mathbf{\Pi}_{P,M}\,\mathbf{T}_{M,P}\left(\mathbf{I}_P \otimes \mathbf{F}_M\right)\mathbf{\Pi}_{M,P}$$

1. Transpose $P \to M$.
2. P local FFTs of size M.
3. Twiddle factors $\mathbf{T}$.
4. Transpose $M \to P$.
5. M local FFTs of size P.
6. Transpose $P \to M$.

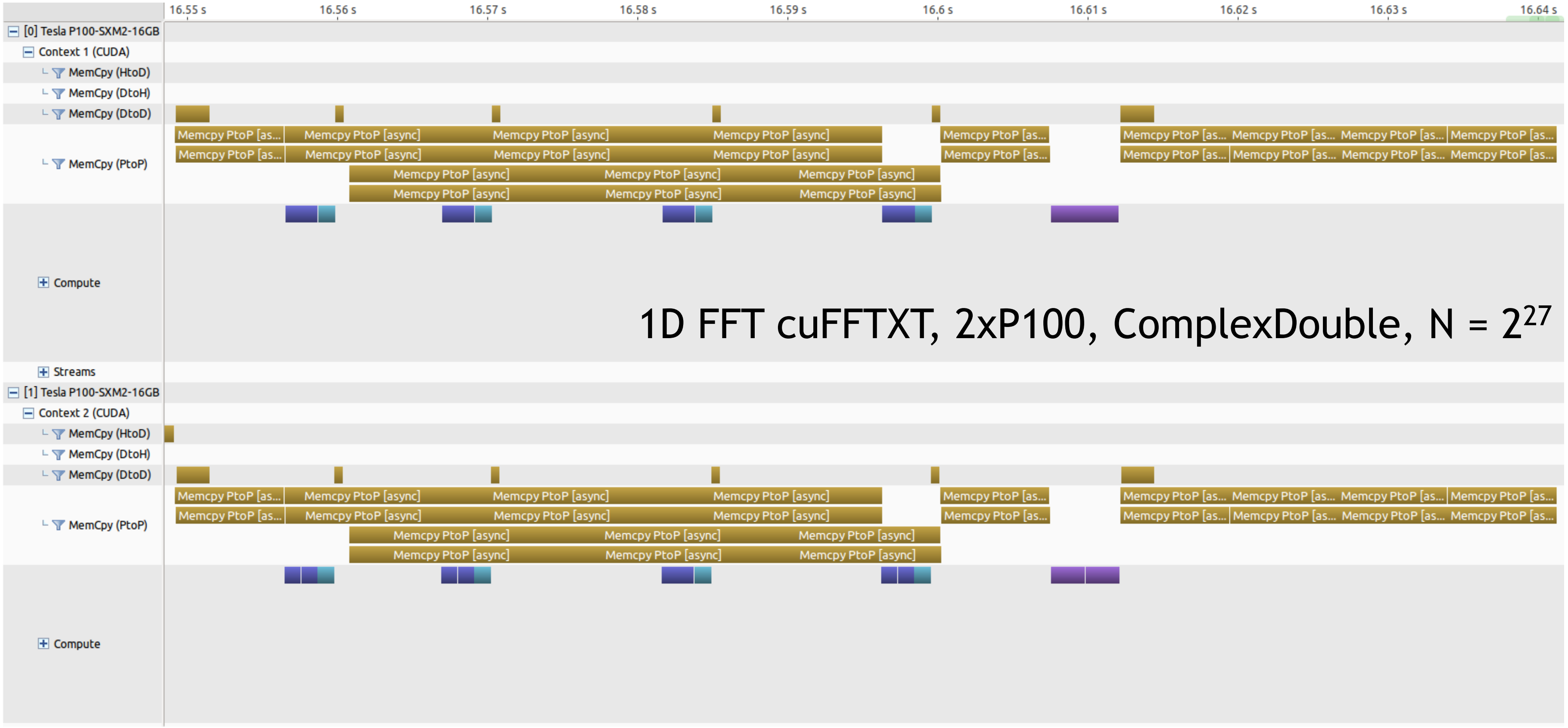


SPLIT-RADIX FFT

Profile

GPU 1

GPU 2



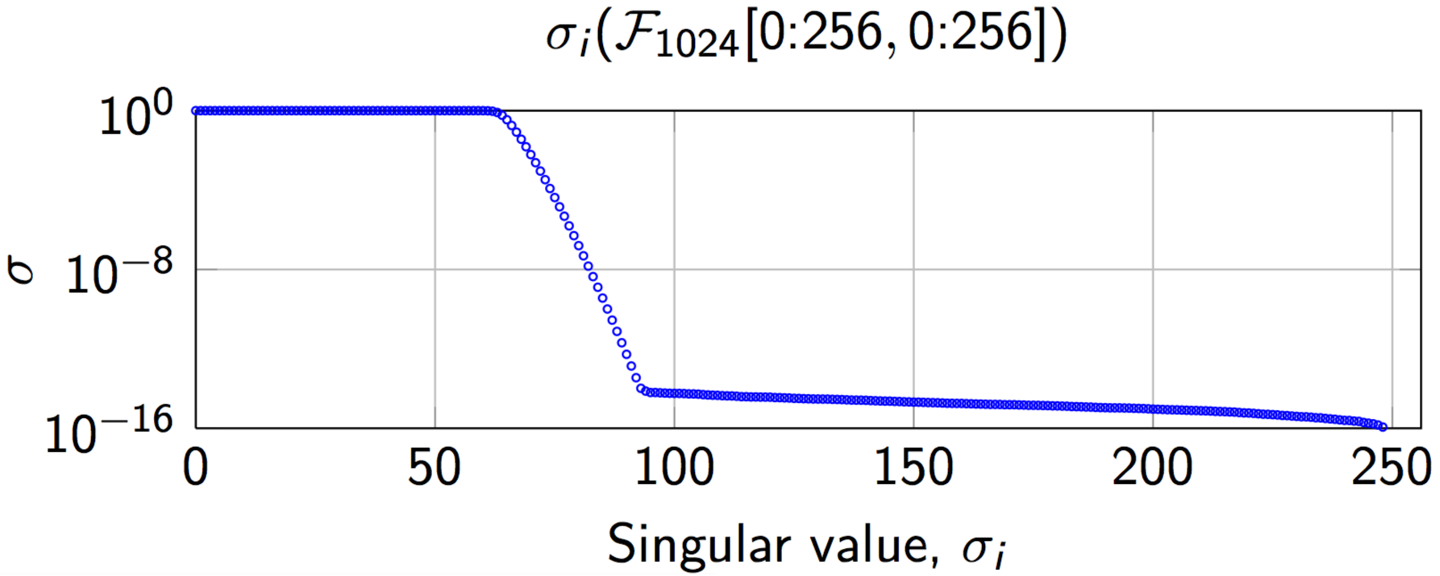
1D FFT cuFFTXT, 2xP100, ComplexDouble, $N = 2^{27}$

FMM-FFT

Edelman et al. 1999

$$F_N = \begin{pmatrix} F_{N|P} & \cdots & D^{P-1}F_{N|P} \\ F_{N|P}D & \cdots & D^{P-1}F_{N|P}D \\ \vdots & & \vdots \\ F_{N|P}D^{P-1} & \cdots & D^{P-1}F_{N|P}D^{P-1} \end{pmatrix}$$

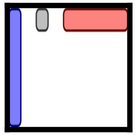
Low-rank structure

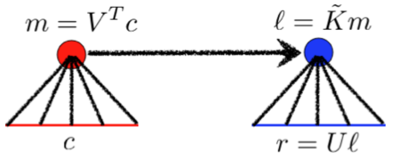


STRUCTURED DENSE MATRICES AND FMM

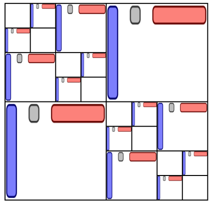
- SVD: $$A = U\,D\,V^*$$

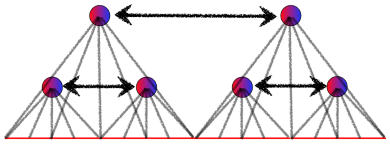
- Low-Rank: $$K = U\,\tilde{K}_{r\times r}\,V^*$$



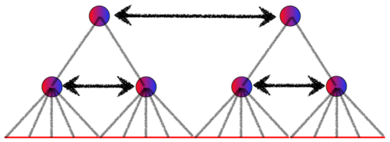


- Hierarchically LR: $$K_{\mathcal{I}\mathcal{J}} = U_{\mathcal{I}}\,\tilde{K}_{\mathcal{I}\mathcal{J}}\,V^*_{\mathcal{J}}$$





- H-Semi-Separable: $$K_{\mathcal{I}\mathcal{J}} = U_{\mathcal{I}}\,\tilde{U}_{\tilde{\mathcal{I}}}\,\tilde{K}_{\tilde{\mathcal{I}}\tilde{\mathcal{J}}}\,\tilde{V}^*_{\tilde{\mathcal{J}}}\,V^*_{\mathcal{J}}$$



- H^2-Matrix/FMM

FMM-FFT

Edelman et al. 1999 Algorithm

$$\mathbf{F}_N = (\mathbf{I}_P \otimes \mathbf{F}_M)\, \mathbf{\Pi}_{M,P}\, (\mathbf{I}_M \otimes \mathbf{F}_P)\, \mathbf{M}_{M,P}$$

$$\mathbf{M}_{M,P} = \mathbf{\Pi}_{P,M}\, \mathrm{diag}(\mathbf{I}_M, \mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_p)\, \mathbf{\Pi}_{M,P}$$

Block diagonal matrix with P blocks of M x M.

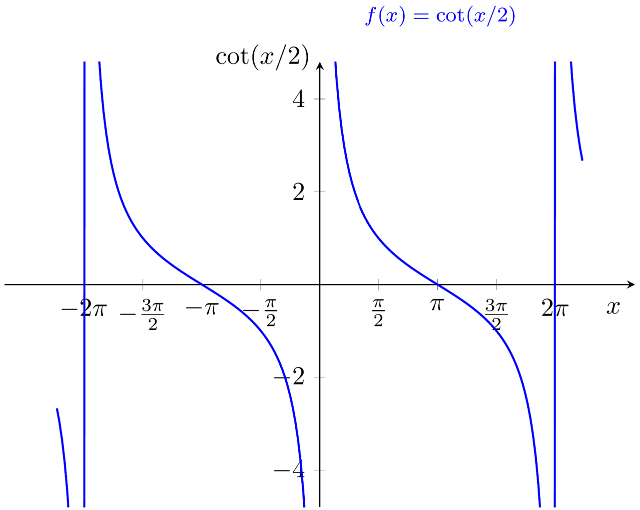
Direct application would destroy asymptotic complexity of the FFT and not effect communication.

However, each C is a *dense kernel matrix*.

COTANGENT FMM

$$[\mathbf{C}_p]_{mn} = \rho_p \left[\cot\left(\frac{\pi}{M}\left(n - m + \frac{p}{P}\right)\right) + \imath\right]$$

- One dimensional
- Uniform – source/target are the integers
- Periodic
- Distributed
- Size M-by-M
- P of them!
    - Interleaved



FMM-FFT

Edelman et al. 1999 Algorithm

$$\mathbf{F}_N = (\mathbf{I}_P \otimes \mathbf{F}_M)\, \mathbf{\Pi}_{M,P}\, (\mathbf{I}_M \otimes \mathbf{F}_P)\, \mathbf{M}_{M,P}$$

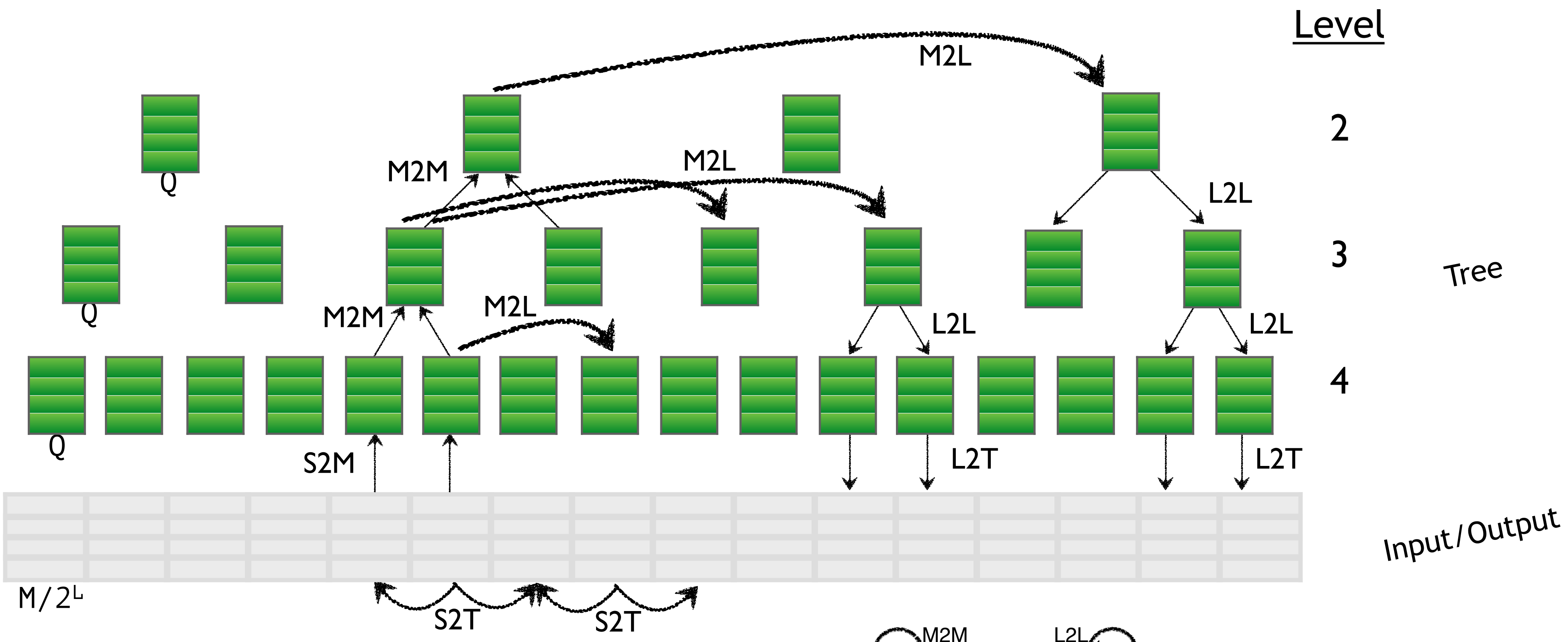
$$\mathbf{M}_{M,P} = \mathbf{\Pi}_{P,M}\, \mathrm{diag}(\mathbf{I}_M, \mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_p)\, \mathbf{\Pi}_{M,P}$$

1. Compute $P-1$ distributed FMMs of size $M \times M$.
2. M local FFTs of size P.
3. Transpose $P \to M$.
4. P local FFTs of size M.

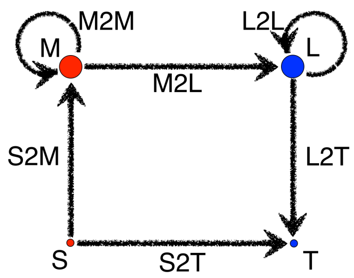
$$\left.\begin{array}{l} \text{2. } M \text{ local FFTs of size } P. \\ \text{3. Transpose } P \to M. \\ \text{4. } P \text{ local FFTs of size } M. \end{array}\right\} \text{2D } M \times P \text{ FFT}$$

FMM OPERATORS

- S: "Source"
- T: "Target"
- M: "Multipole"
- L: "Local"



Each operator is an (implicit) matrix.



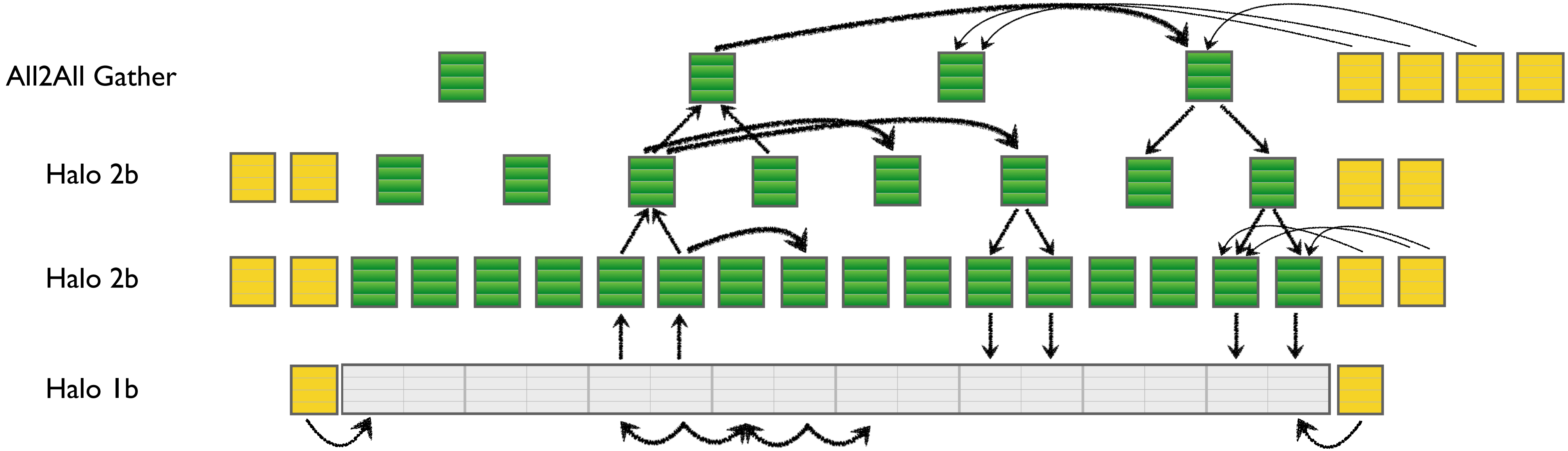
PARAMETERS OF THE FMM-FFT

- FFT $$N = M\,P$$

  P FMMs to perform of size M x M

- FMM
  - Rank Q
  - Leaf box size M_L
  - Base level B
  - Leaf level $L = \log_2(M/M_L)$

DISTRIBUTED FMM

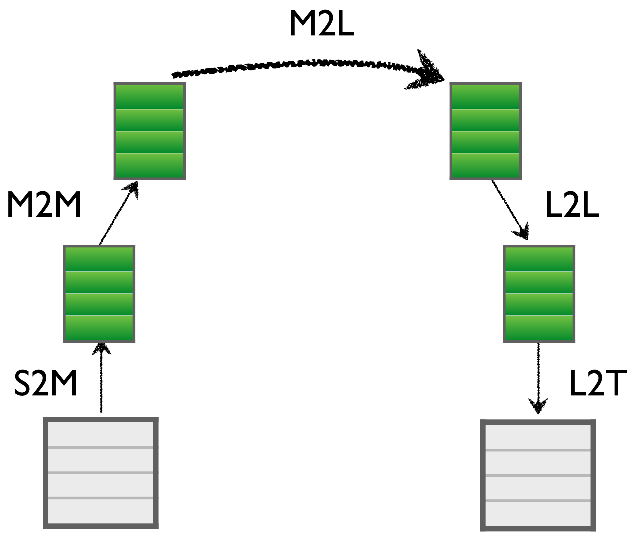


INTERPOLATIVE FMM

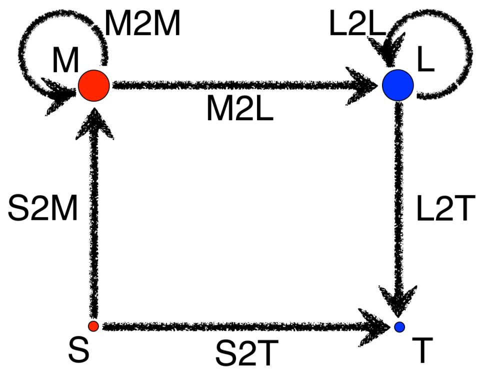
Chebyshev interpolations

- Use Chebyshev interpolations as the intermediate data
  - Same operators across all boxes
  - Same operators across all levels
  - Almost same operators across all FMMs

- Express as tensors and efficiently compute.

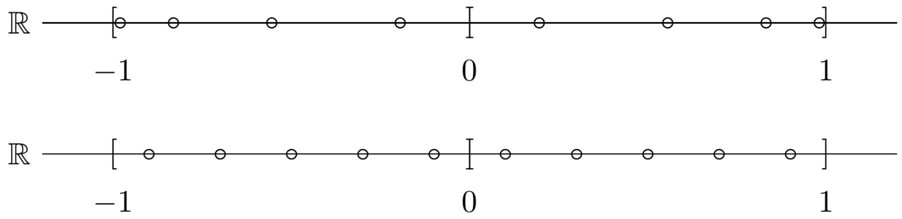


S2M/L2T



$$\mathcal{M}^{L}_{(p-1)qb} = S2M_{qm}\,\mathcal{S}_{pmb}$$

$$\mathcal{T}_{pmb} = L2T_{mq}\,\mathcal{L}_{(p-1)qb} + \mathcal{T}_{pmb}$$



Tree Leaf (M)

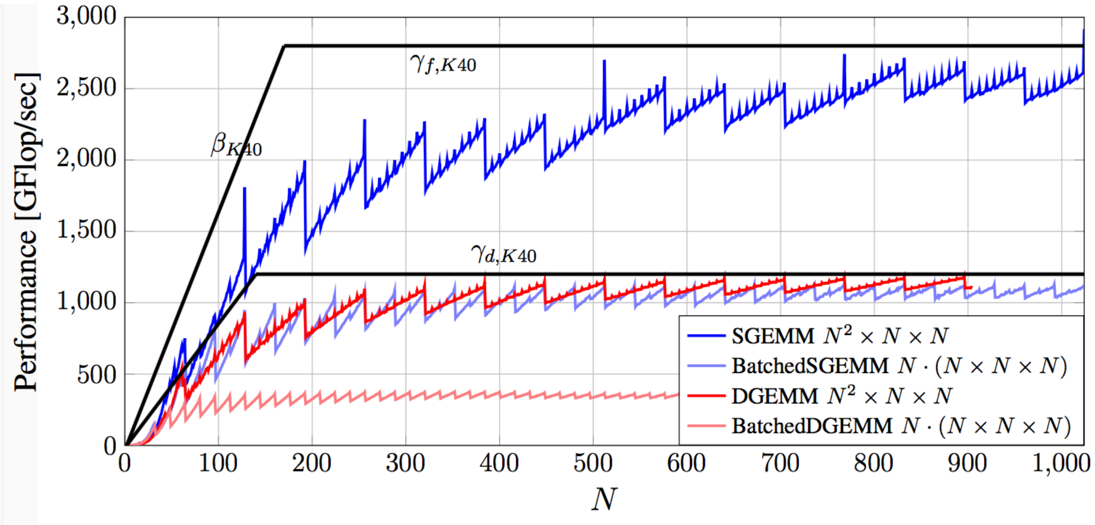
Sources (S)

Computed with single BatchedGEMM

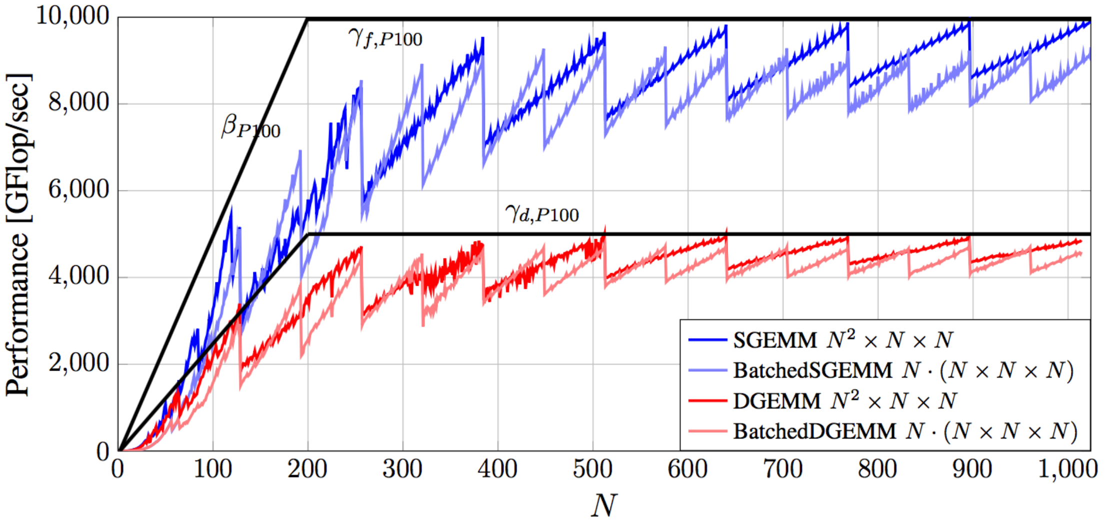
BATCHED MATRIX-MATRIX MULTIPLY

cublas<T>gemmStridedBatched

in cuBLAS 8.0

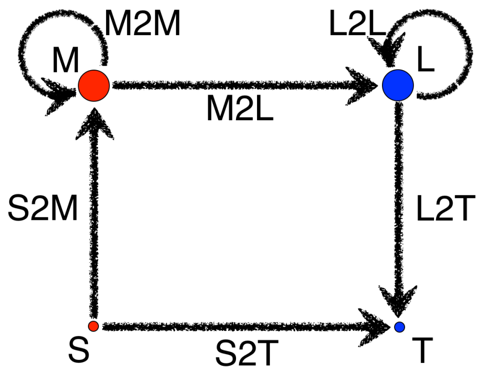


(a) K40c GPU



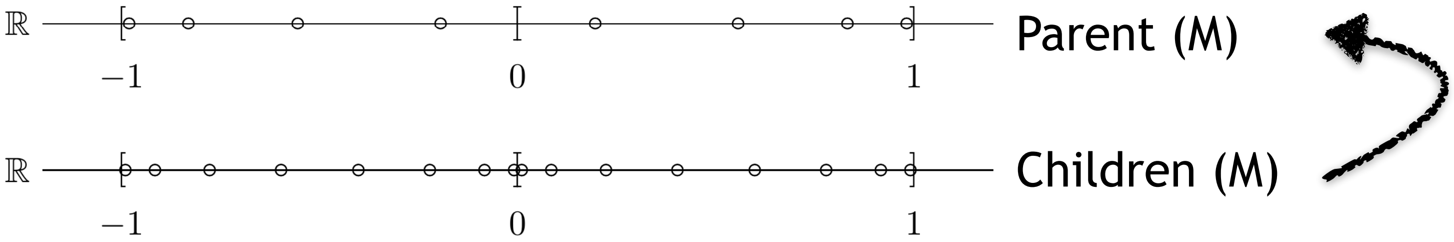
(b) P100 GPU

M2M/L2L



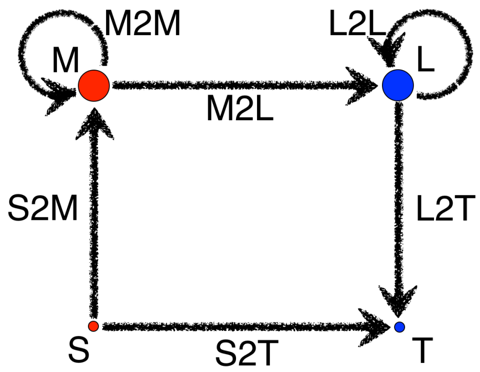
$$\mathcal{M}^{\ell}_{pqb} = M2M_{qk}\,\mathcal{M}^{\ell+1}_{pk(2b)}$$

$$\mathcal{L}^{\ell+1}_{pq(2b)} = L2L_{qk}\,\mathcal{L}^{\ell}_{pkb} + \mathcal{L}^{\ell+1}_{pq(2b)}$$



Computed with single BatchedGEMM

S2T/M2L



$$\mathcal{T}_{pib} = S2T_{p(j-i)}\, \mathcal{S}_{pjb}$$

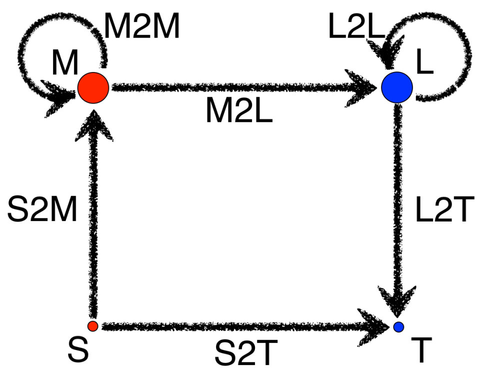
$$S2T_{pk} = \begin{cases} \cot\left(\frac{\pi}{N}(p+Pk)\right) & p>0 \\ \delta_{k0} & p=0 \end{cases}$$

$$\mathcal{L}^{\ell}_{pib} = M2L^{\ell}_{pijs}\, \mathcal{M}^{\ell}_{pj(b+s)}$$

$$M2L^{\ell}_{pijs} = \cot\left(\frac{\pi}{2^{\ell}}\left(\frac{z_j}{2} - \frac{z_i}{2} + s\right) + \frac{\pi}{N}(p+1)\right)$$

- Also Level-3 Linear Algebra computations, but no BLAS primitives.
  - CUSTOM KERNELS

INTERPOLATIVE FMM



Operator	Storage	Compute
$S2T^{(p)}_{ijs} = \cot\left(\frac{\pi}{M}(j-i) + \frac{\pi}{2^L}s + \frac{\pi}{N}p\right)$	$P(4M_L-1)$	$3P2^LM_L^2$
$S2M_{ij} = \ell_i\left(s_j\right) = \begin{cases} 1 & s_j = z_i \\ \frac{\lambda_i\,\ell(s_j)}{s_j - z_i} & \text{else} \end{cases}$	QM_L	$2PMQ$
$M2M^{\pm}_{ij} = \ell_i\left(\frac{z_j \pm 1}{2}\right)$	$2Q^2$	$4(2^L-2^B)PQ^2$
$M2L^{(p)}_{ijs} = \cot\left(\frac{\pi}{2^L}\left(\frac{z_j}{2} - \frac{z_i}{2}\right) + \frac{\pi}{2^L}s + \frac{\pi}{N}p\right)$	$4(L-B)PQ^2$	$3(2^L-2^B)PQ^2$
$L2L^{\pm}_{ij} = \ell_j\left(\frac{z_i \pm 1}{2}\right) = M2M^{\pm}_{ji}$	$2Q^2$	$4(2^L-2^B)PQ^2$
$L2T_{ij} = \ell_j\left(t_i\right) = S2M_{ji}$	QM_L	$2PMQ$

Compute on the fly!

Total Precomputation (S2M+M2M):
$Q^2M_L+Q^3$ ~ **20k Flops**

Total Overhead (S2M+M2M):
QM_L+Q^2 ~ **12k Bytes**

NVIDIA.

ALGORITHM

Algorithm 1 The FMM-FFT via tensor contractions.

$\mathcal{M}^{L}_{(p-1)qb} = S2M_{qm}\,\mathcal{S}_{pmb}$ // S2M
SendRecv $\mathcal{S}$ halo. // COMM $\mathcal{S}$
$\mathcal{T}_{pib} = S2T_{p(j-i)}\,\mathcal{S}_{pjb}$ // S2T
for $\ell = L-1, \ldots, B$ **do**
  $\mathcal{M}^{\ell}_{pqb} = M2M_{qk}\,\mathcal{M}^{\ell+1}_{pk(2b)}$ // M2M
for $\ell = L, \ldots, B-1$ **do**
  SendRecv $\mathcal{M}^{\ell}$ halo. // COMM $\mathcal{M}^{\ell}$
  $\mathcal{L}^{\ell}_{pib} = M2L^{\ell}_{pijs}\,\mathcal{M}^{\ell}_{pj(b+s)}$ // M2L ℓ
All-to-all gather $\mathcal{M}^{B}$. // COMM $\mathcal{M}^{B}$
$\mathcal{L}^{B}_{pib} = M2L_{pijs}\,\mathcal{M}^{B}_{pj(b+s)}$ // M2L B
$r_p = 1_{qb}\,\mathcal{M}^{B}_{(p-1)qb}$ // REDUCE
for $\ell = B, \ldots, L-1$ **do**
  $\mathcal{L}^{\ell+1}_{pq(2b)} = L2L_{qk}\,\mathcal{L}^{\ell}_{pkb} + \mathcal{L}^{\ell+1}_{pq(2b)}$ // L2L
$\mathcal{T}_{pmb} = L2T_{mq}\,\mathcal{L}^{L}_{(p-1)qb} + \mathcal{T}_{pmb}$ // L2T
$\mathcal{T}_{pmb} = \rho_p(\mathcal{T}_{pmb} + \imath r_p)$ // POST
$\mathbf{F}_{M,P}\,\mathcal{T}$ // 2D FFT

: Primitive – vendor optimized

`cublas<T>gemmStridedBatched`

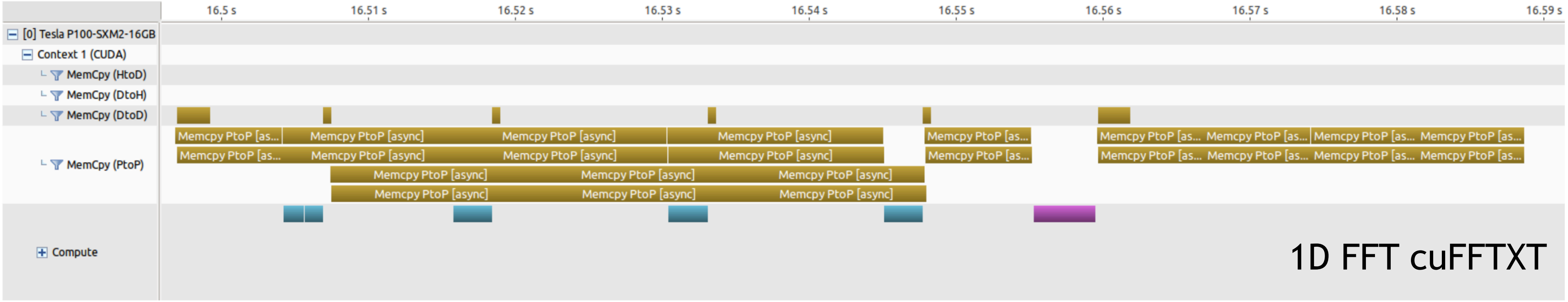
: Custom kernel

: FMM Communication

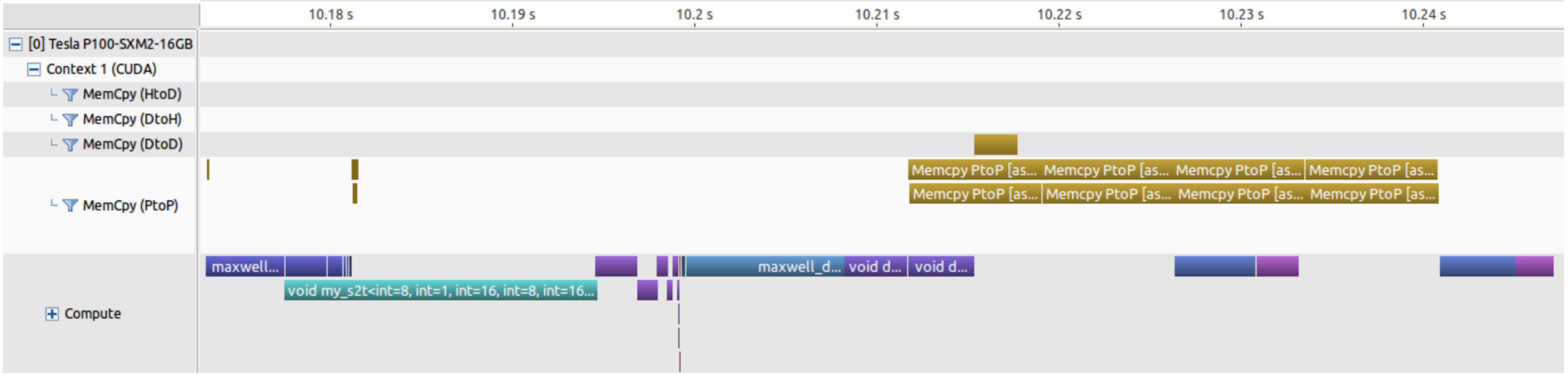
NVIDIA.

COMPARISON

Profile

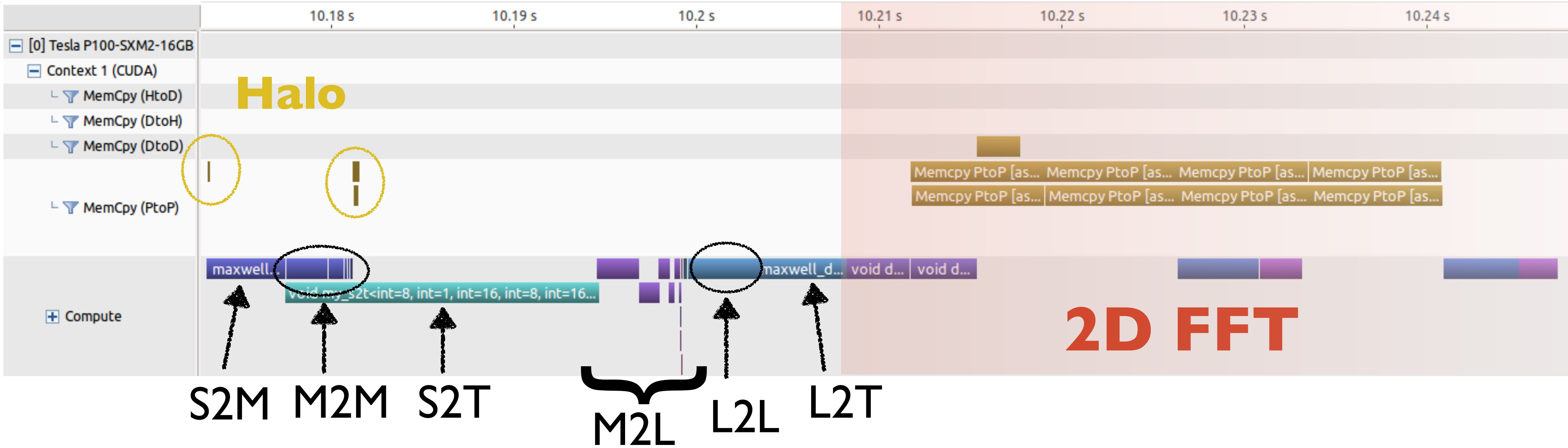


1D FFT cuFFTXT



1D FFT FMM-FFT

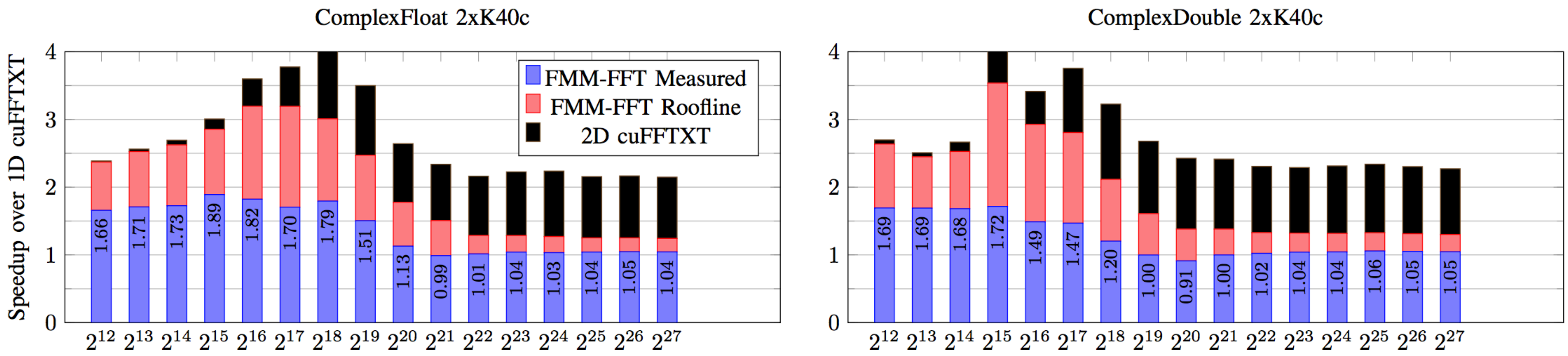
FMM-FFT PROFILE



Emph: 255 FMMs of size 524k x 524k in 32ms

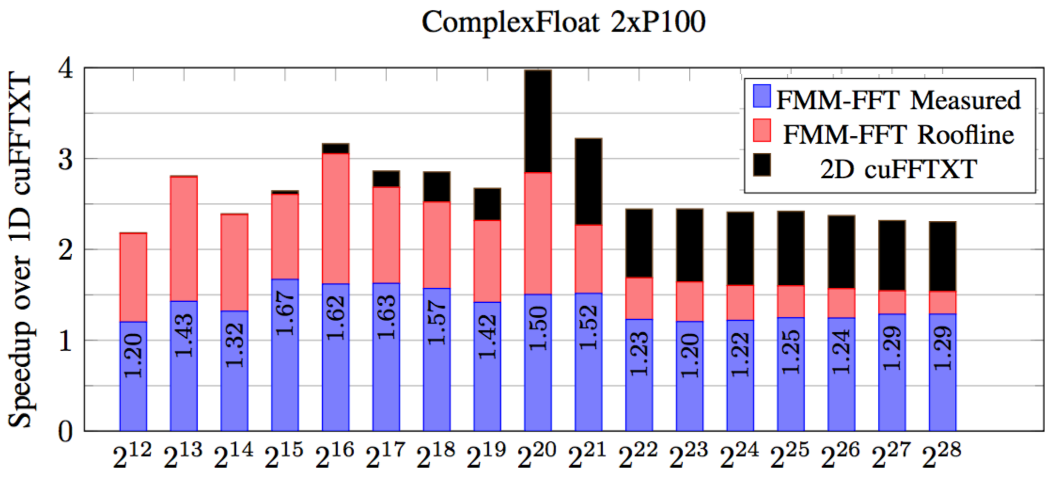
2xK40c, PCIe

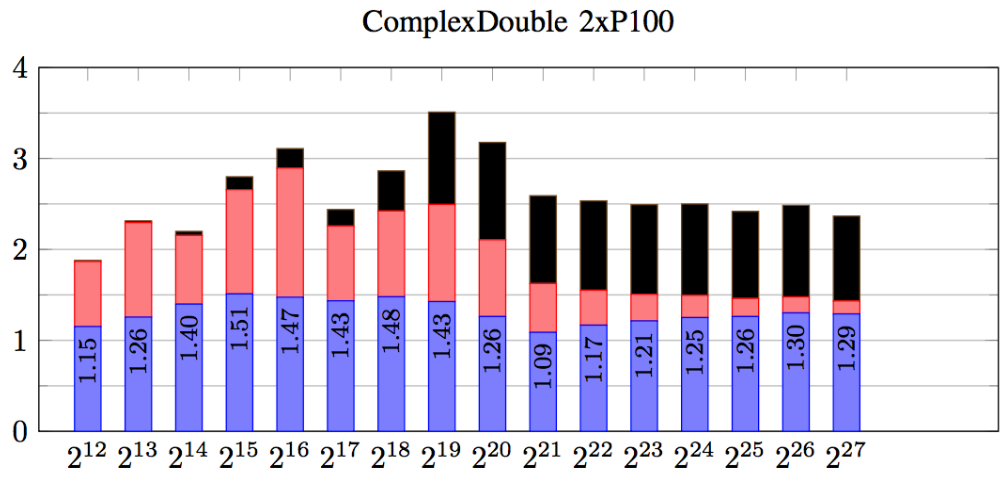
FMM-FFT Performance



2xP100, NVLINK

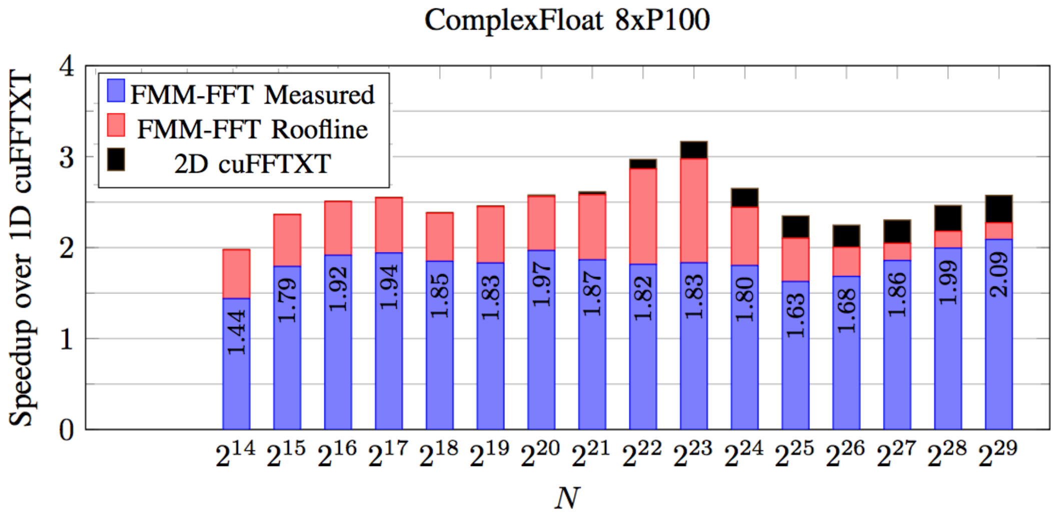
FMM-FFT Performance

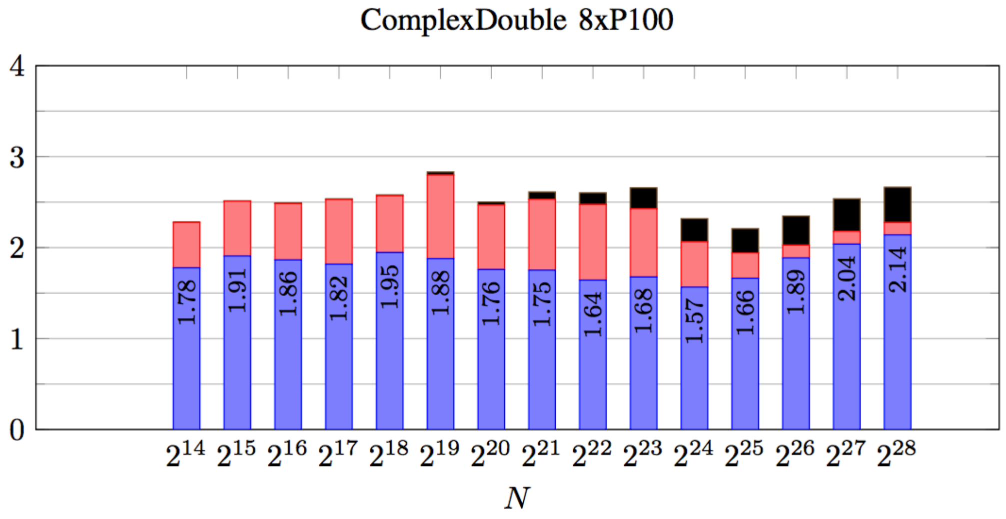




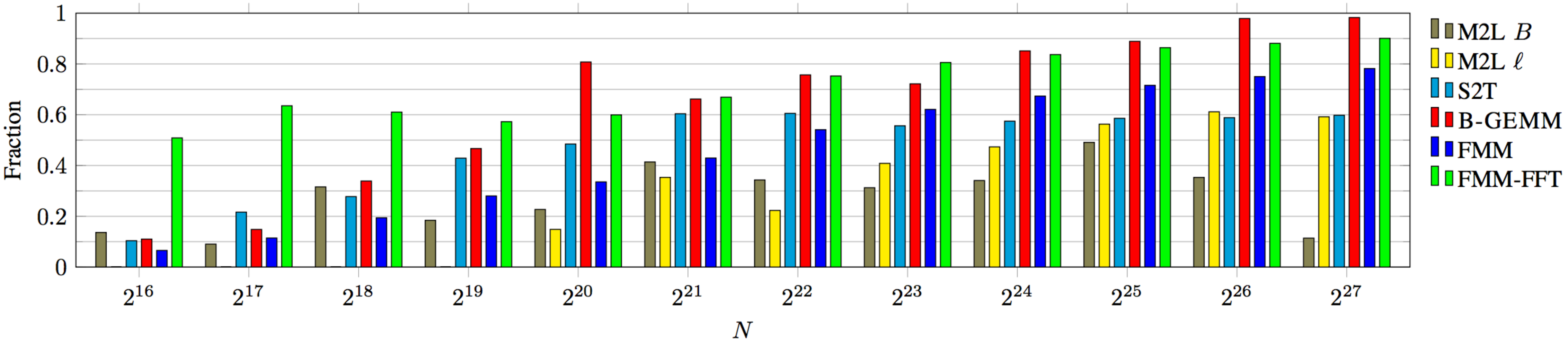
8xP100, NVLINK

FMM-FFT Performance





EFFICIENCY



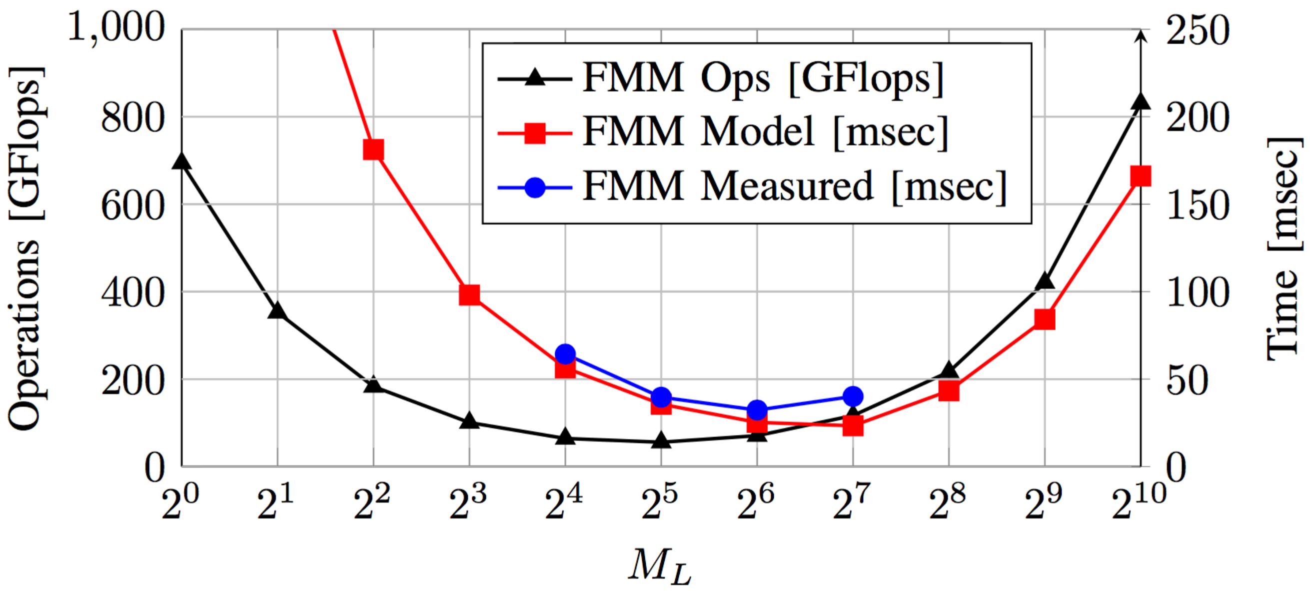
- >95% BatchedGEMM
- 60% S2T/M2L
- >90% FMM-FFT

PARAMETER DEPENDENCE — M_L

Points per leaf box per FMM

- Trade #levels for S2T comp
- Counting flops not enough
  - Want intensity increase
- Tune performance for M_L=64

ComplexDouble, 2xP100, N=2^{27}, P=256, B=3, Q=16



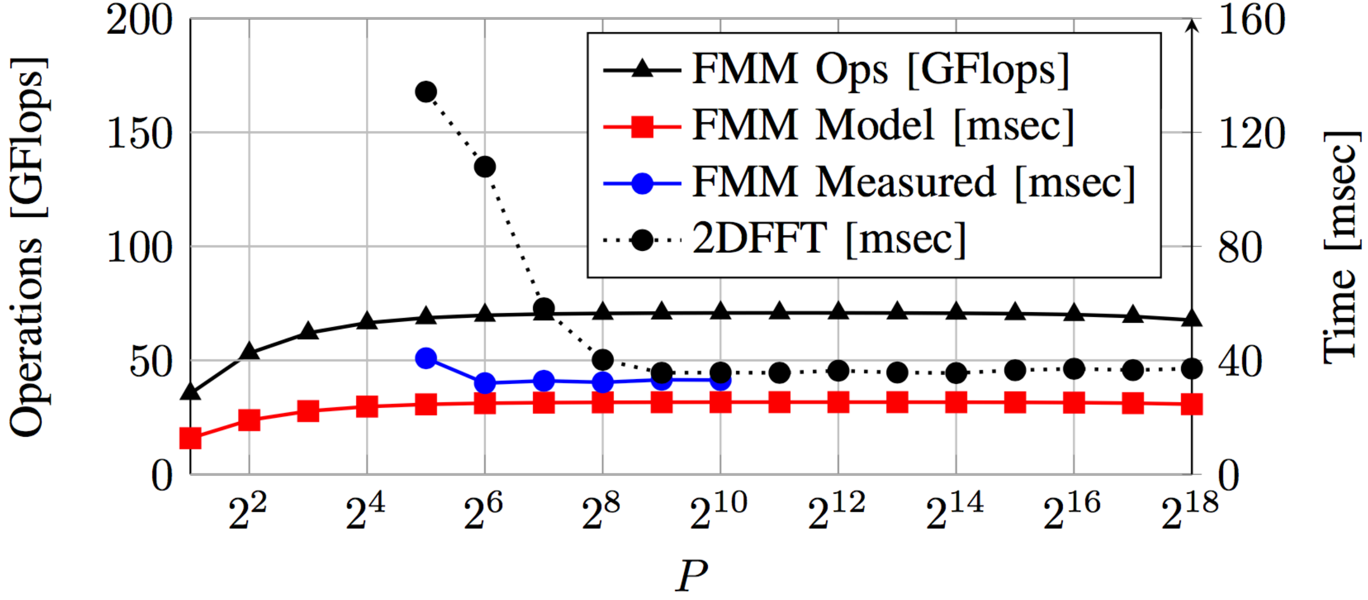
PARAMETER DEPENDENCE — P

Number of FMMs

- Flops/Intensity approx constant
  - Trade #levels for #FMMs

- Large P good
  - Fill up B-GEMM
  - More square 2D FFT

ComplexDouble, 2xP100, $N=2^{27}$, $M_L=64$, B=3, Q=16



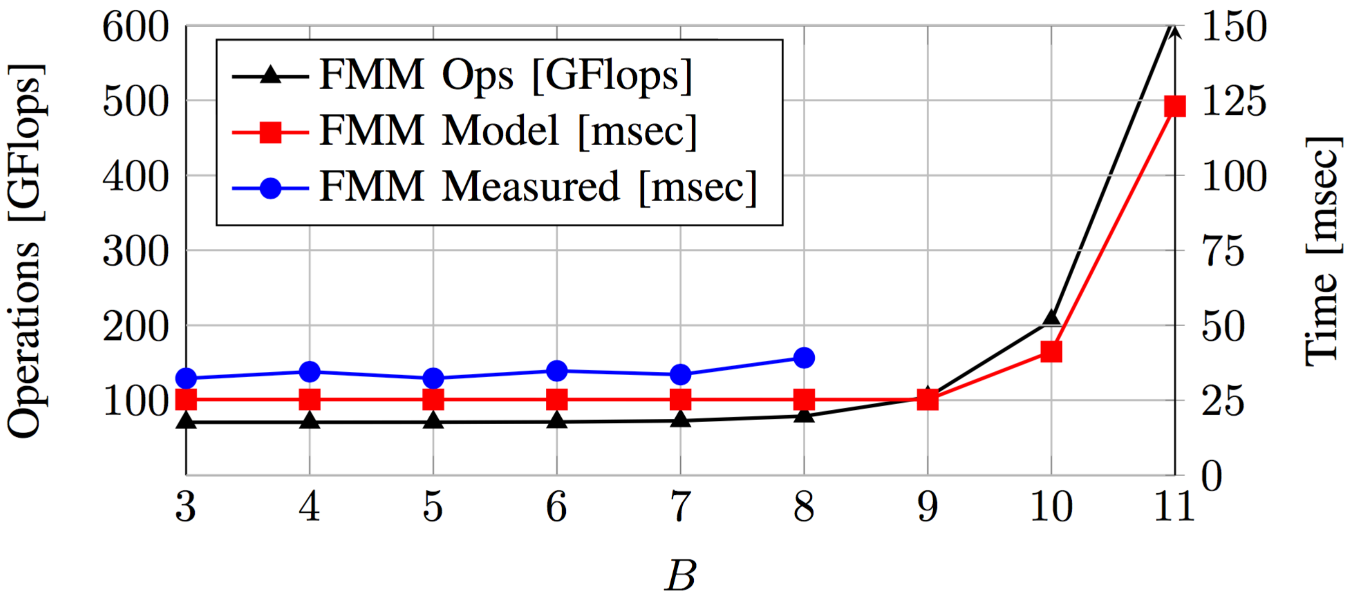
PARAMETER DEPENDENCE — B

Base level of FMM trees

- Not very significant

- Scale to 128 GPUs w/o complications

ComplexDouble, 2xP100, $N=2^{27}$, P=256, M_L=64, Q=16

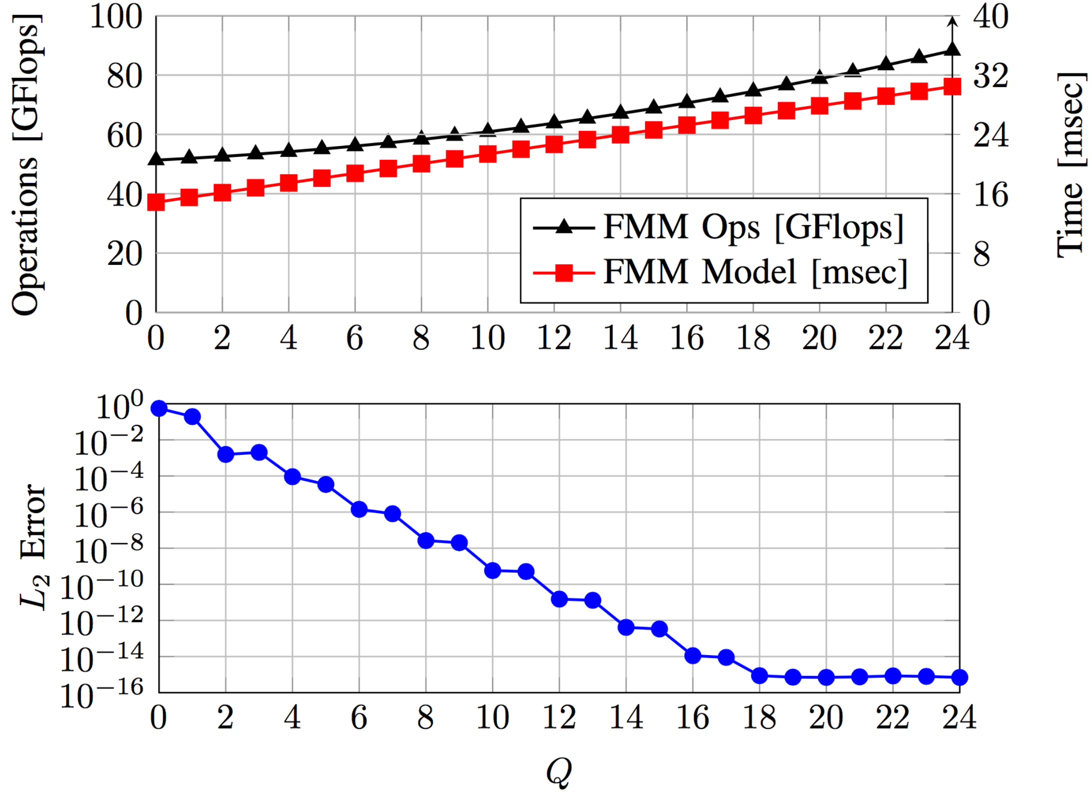


PARAMETER DEPENDENCE — Q

Quadrature order of FMM interpolation

- Accuracy tuning with performance dependence

ComplexDouble, 2xP100, N=2^{27}, P=256, M_L=64, B=3



FUTURE

- Integration into cuFFT
- Application to 2D/3D FFTs?
  - Convolutions
- NUFFT, Sparse FFT nearly out-of-the-box
- Volta predictions and measurements
  - Mixed precision (e.g. FP16 far-field) to use Tensor Core?
- Persistent Matrix Batched GEMM (cuBLAS optimization)
  - Staged Persistent Matrix Batched GEMM (cooperative groups, RNNs)

CONCLUSION

- FMM-FFT trades 2/3 communication in 1D FFT for P FMMs
  - Viable on highest comp:comm architecture available
- Detailed implementation that relies heavily on existing primitives
  - Primitives >95% efficient
  - Two custom dense kernels >60% efficient
  - Entire FMM-FFT >90% efficient
- Tunable accuracy-performance tradeoff
- Compute model accurately predicts performance

Cris Cecka, "Low-communication FMM-accelerated FFTs on GPUs" *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, Denver, CO, November 12-17, 2017

Yang Shi, U.N. Niranjan, Animashree Anandkumar, and **Cris Cecka**. "Tensor contractions with extended BLAS kernels on CPU and GPU" In *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, pages 193–202, Dec 2016