Tensor Contractions with Extended BLAS Kernels on CPU and GPU

Cris Cecka

Senior Research Scientist
NVIDIA Research, Santa Clara, California

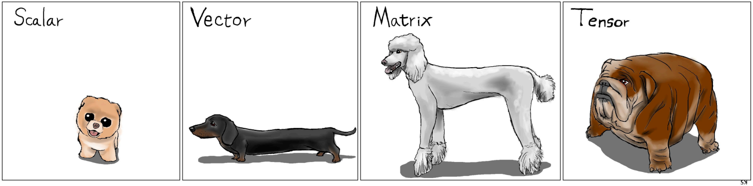
Joint work with Yang Shi, U.N. Niranjan, and Animashree Anandkumar

Electrical Engineering and Computer Science
University of California, Irvine, California

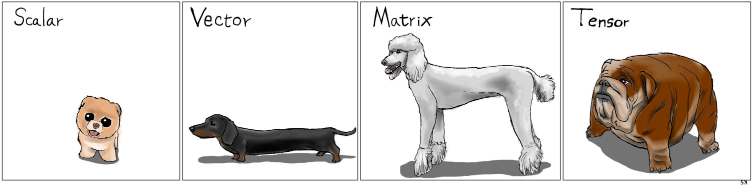
SIAM CSE 2017

July 10, 2017

Tensor Contraction-Motivation

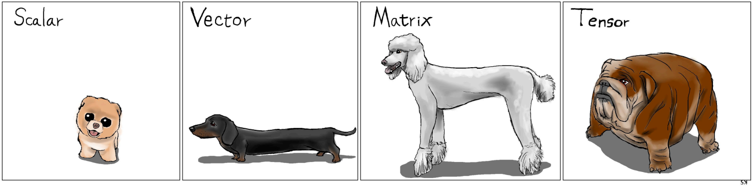


Tensor Contraction-Motivation

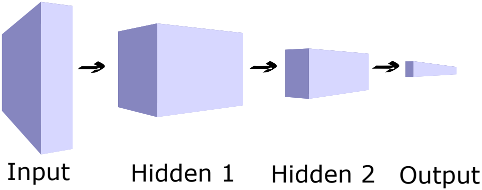


Modern data is inherently multi-dimensional

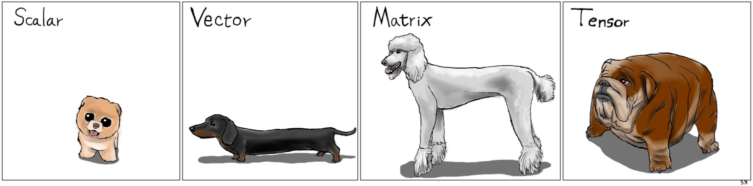
Tensor Contraction-Motivation



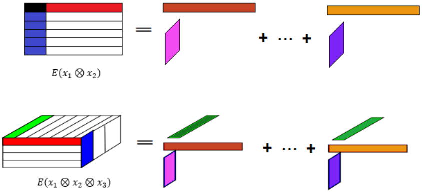
Modern data is inherently multi-dimensional



Tensor Contraction-Motivation



Modern data is inherently multi-dimensional



Tensor Contraction-Motivation

What is tensor contraction?

Tensor Contraction-Motivation

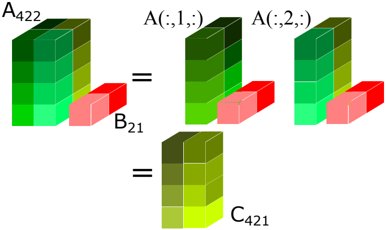
What is tensor contraction?

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

Tensor Contraction-Motivation

What is tensor contraction?

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

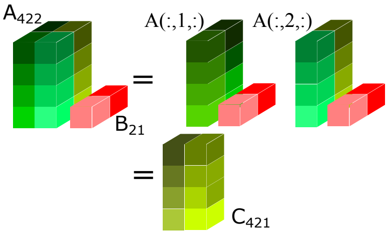


$$\text{e.g.} \quad C_{mnp} = A_{mnk} \, B_{kp}$$

Tensor Contraction-Motivation

What is tensor contraction?

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



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

Why do we need tensor contraction?

1. Core primitive of multilinear algebra.
2. BLAS Level 3: Unbounded compute intensity.

Tensor Contraction – Motivation

Lots of hot applications at the moment:

- Machine learning
- Deep learning
- e.g. Learning latent variable model with tensor decomposition:

Topic model [1]

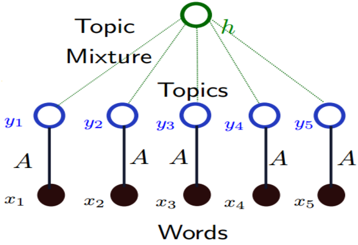
Tensor Contraction – Motivation

Lots of hot applications at the moment:

Machine learning
Deep learning
e.g. Learning latent variable model with tensor decomposition:

Topic model [1]

h: PDF of topics in a document.
A: Topic-word matrix.
$A_{ij} = \mathcal{P}(x_m = i | y_m = j)$

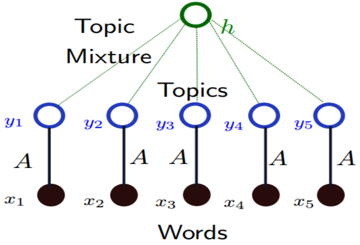


Tensor Contraction – Motivation

Lots of hot applications at the moment:

Machine learning
Deep learning
e.g. Learning latent variable model with tensor decomposition:

Topic model [1]
h: PDF of topics in a document.
A: Topic-word matrix.
$A_{ij} = \mathcal{P}(x_m = i | y_m = j)$



Form third-order tensor $M_3 = \mathbb{E}(x \otimes x \otimes x) = \sum_i h_i a_i \otimes a_i \otimes a_i$

[1] Tensor Decompositions for Learning Latent Variable Models, Anima Anandkumar, Rong Ge, Daniel Hsu et. al.

Tensor Contraction – Motivation

Distributed FFT

Tensor Contraction – Motivation

Distributed FFT

$$\begin{aligned}
T_{pib} &= S2T^{(p)}_{ijs}\, S_{pj(b+s)} && \Longrightarrow && T_{pib} = S2T^{(p)}_{i(js)}\, S_{p(js)b} \\
M_{pqb} &= S2M_{qi}\, S_{pib} && \Longrightarrow && M_{pq[b]} = S_{pi[b]}\, S2M^{T}_{qi} \\
M_{pqb'} &= M2M^{-}_{qm}\, M_{pmb^{-}} + M2M^{+}_{qm}\, M_{pmb^{+}} && \Longrightarrow && M_{pq[b']} = M_{pM[b]}\, M2M^{T}_{qM} \\
r_p &= 1_{ib}\, S_{pib} = 1_{qb}\, M_{pqb} && \Longrightarrow && r_p = 1_{(qb)}\, M_{p(qb)} \\
L_{pnb} &= M2L^{(p)}_{nms}\, M_{pm(b+s)} && \Longrightarrow && L_{pnb} = M2L^{(p)}_{n(ms)}\, M_{p(ms)b} \\
L_{pqb^{\pm}} &= L2L^{\pm}_{qm}\, L_{pmb'} && \Longrightarrow && L_{pq[b]} = L_{pM[b']}\, M2M_{qM} \\
T_{pib} &= L2T_{iq}\, L_{pqb} && \Longrightarrow && T_{pi[b]} = L_{pq[b]}\, S2M_{qi}
\end{aligned}$$

What do we have?

Tensor Contraction-Motivation

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...

Tensor Contraction-Motivation

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 Contraction-Motivation

Libraries

Explicit permutation dominates.

Tensor Contraction-Motivation

Libraries

Explicit permutation dominates.

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

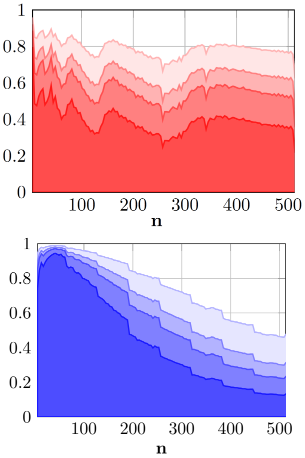
1. $A_{km} \rightarrow A_{mk}$
2. $B_{pkn} \rightarrow B_{kpn}$
3. $C_{mnp} \rightarrow C_{mpn}$
4. $\boxed{C_{m(pn)} = A_{mk} B_{k(pn)}}$
5. $C_{mpn} \rightarrow C_{mnp}$

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.

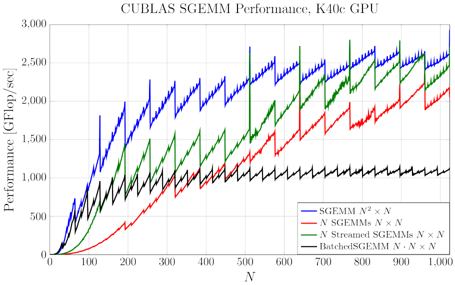
Pointer-to-Pointer BatchedGEMM

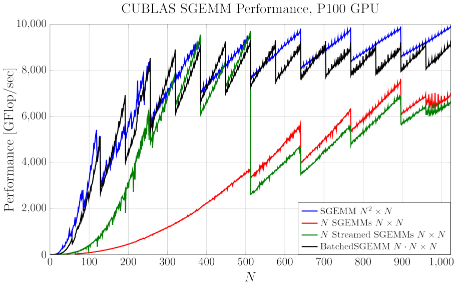
- Available in MKL 11.3β and cuBLAS 4.1

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

```
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)
```

Pointer-to-Pointer BatchedGEMM

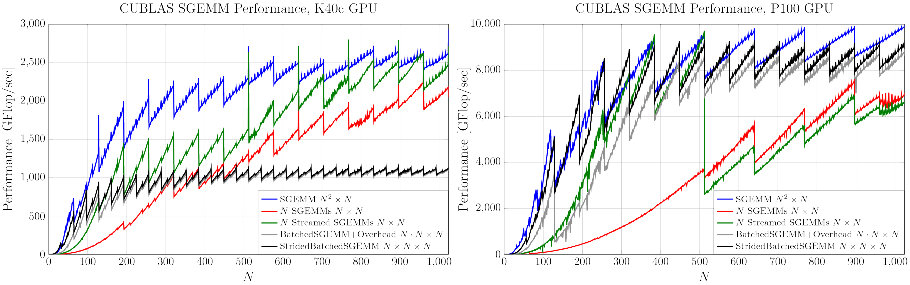




Existing Primitives

Pointer-to-Pointer BatchedGEMM

Except actually...



Solution: StridedBatchedGEMM

StridedBatchedGEMM

Exists!

... ~~Still no documentation?!?~~

Documentation as of last Tuesday!

StridedBatchedGEMM

Exists!

... ~~Still no documentation?!?~~

Documentation as of last Tuesday!

In cuBLAS 8.0:

```
$$ grep StridedBatched -A 17 /usr/local/cuda/include/cublas_api.h
2320:CUBLASAPI cublasStatus_t cublasSgemmStridedBatched (cublasHandle_t handle,
2321-                                                       cublasOperation_t transa,
2322-                                                       cublasOperation_t transb,
2323-                                                       int m,
2324-                                                       int n,
2325-                                                       int k,
2326-                                                       const float *alpha,  // host or device pointer
2327-                                                       const float *A,
2328-                                                       int lda,
2329-                                                       long long int strideA,   // purposely signed
2330-                                                       const float *B,
2331-                                                       int ldb,
2332-                                                       long long int strideB,
2333-                                                       const float *beta,   // host or device pointer
2334-                                                       float *C,
2335-                                                       int ldc,
2336-                                                       long long int strideC,
2337-                                                       int batchCount);
...
```

StridedBatchedGEMM

```
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 BatchedGEMM.
- No Pointer-to-pointer data structure or overhead.
- Performance on par with pure GEMM (P100 and beyond).

Tensor Contraction with Extended BLAS Primitives

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

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

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	$A_{kp}B_{nmk}$		

Tensor Contraction with Extended BLAS Primitives

Case	Contraction	Kernel1	Kernel2	Kernel3
1.1	$A_{mk}B_{knp}$	$C_{m(np)} = A_{mk}B_{k(np)}$	$C_{mn[p]} = A_{mk}B_{kn[p]}$	$C_{m[n]p} = A_{mk}B_{k[n]p}$
6.1	$A_{kp}B_{kmn}$	$C_{(mn)p} = B^{\top}_{k(mn)}A_{kp}$	$C_{m[n]p} = B^{\top}_{km[n]}A_{kp}$	

Example: Mappings to Level 3 BLAS routines

- Case 1.1, Kernel2: $C_{mn[p]} = A_{mk}B_{kn[p]}$

```
cublasDgemmStridedBatched(handle,
                          CUBLAS_OP_N, CUBLAS_OP_N,
                          M, N, K,
                          &alpha,
                          A, ldA1, 0,
                          B, ldB1, ldB2,
                          &beta,
                          C, ldC1, ldC2,
                          P)
```

Tensor Contraction with Extended BLAS Primitives

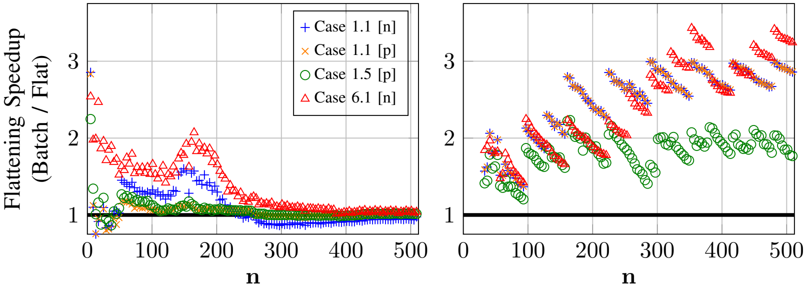
Case	Contraction	Kernel1	Kernel2	Kernel3
1.1	$A_{mk}B_{knp}$	$C_{m(np)} = A_{mk}B_{k(np)}$	$C_{mn[p]} = A_{mk}B_{kn[p]}$	$C_{m[n]p} = A_{mk}B_{k[n]p}$
6.1	$A_{kp}B_{kmn}$	$C_{(mn)p} = B^{\top}_{k(mn)}A_{kp}$	$C_{m[n]p} = B^{\top}_{km[n]}A_{kp}$	

Example: Mappings to Level 3 BLAS routines

- Case 6.1, Kernel2: $C_{m[n]p} = B^{\top}_{km[n]}A_{kp}$

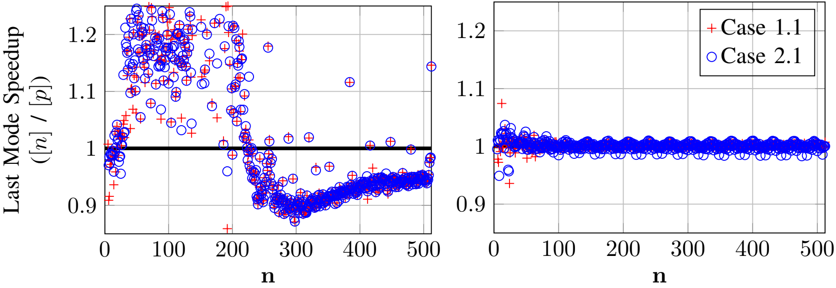
```
cublasDgemmStridedBatched(handle,
                          CUBLAS_OP_T, CUBLAS_OP_N,
                          M, P, K,
                          &alpha,
                          B, ldB1, ldB2,
                          A, ldA1, 0,
                          &beta,
                          C, ldC2, ldC1,
                          N)
```

Flatten V.S. SBGEMM



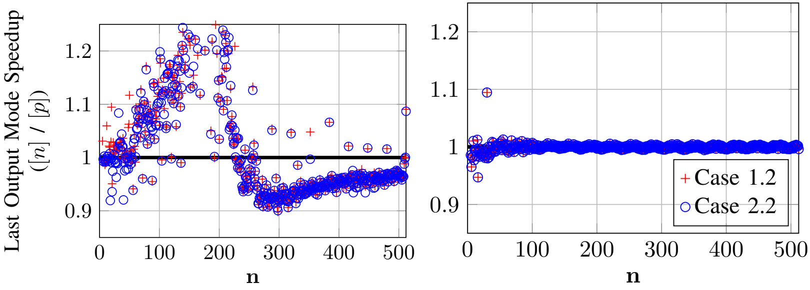
Prefer flattening to "pure" GEMM.

Batching in last mode versus middle mode



On CPU: Prefer batching in the last mode.

Mixed mode batching



On CPU: mode of the output tensor is more important than the batching mode of the input tensor.

Analysis

Exceptional Cases:
Cannot be computed by StridedBatchedGEMM.

Case	Contraction
3.4	$C_{mnp} = A_{nk}B_{pkm}$
6.4	$C_{mnp} = A_{kp}B_{nkm}$
	$C_{mnp} = A_{mkp}B_{mkn}$
	$C_{mnp} = A_{pkm}B_{nkp}$

Example of exceptional cases.

- These cases are precisely the interleaved GEMMs.
- When batching index is the major index in an argument:
  - That argument is interpreted as interleaved matrices.
  - May be one or both inputs and/or output.

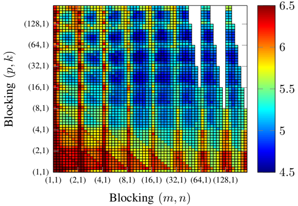
3D Tiled GEMM

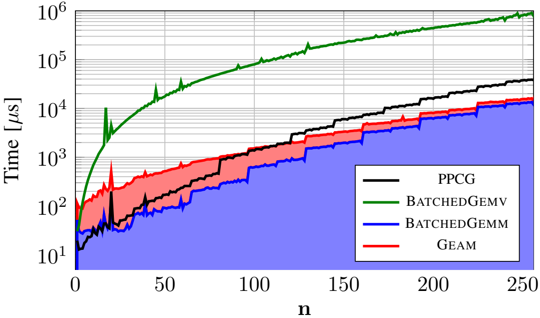
Implement GEMM with a 3D tile:

- Transpositions performed on the way to smem/reg.
- Keep canonical GEMM core.
- Considers three modes rather than two:
  - Major mode: $A_{\mathbf{m}nkpqr}$
  - Reduction mode: $A_{mn\mathbf{k}pqr}$
  - Aux (batch,row,col) mode: $A_{mnk\mathbf{p}qr}$ (Optional)
- Third tile dimension interpolates between pure GEMM and interleaved GEMM.
- Nested loop over remaining modes performs full contraction.

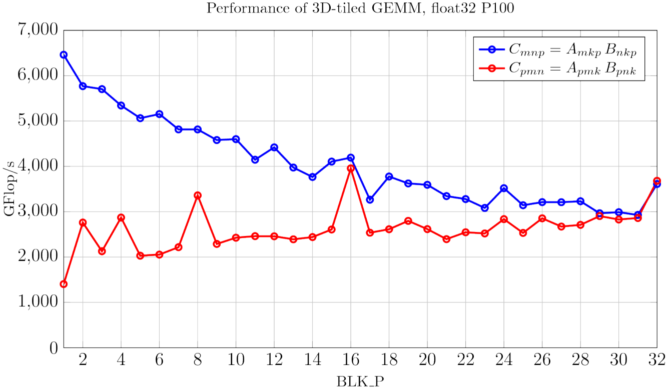
3D Tiled GEMM

Tilesize tuning with PPCG for exceptional cases:





3D Tiled GEMM



- $C_{mnp} = A_{mkp}\, B_{nkp}$: Increasing BLK_P decreases effective tile size.
- $C_{pmn} = A_{pmk}\, B_{pnk}$: Increasing BLK_P increases cache line utilization.
  - e.g. BLK_P$= 1, 2, 4, 8$
  - BLK_P $= 1$ equivalent to BLIS (strides in row and column)

3D Tiled GEMM – Interface?

Extend the StridedBatchedGEMM transpose parameters?

✓	C_{mnp}		$A_{pmk}\, B_{pkn}$	EX_N	EX_N
✓	C_{mpn}	$=$	$A_{pmk}\, B_{pnk}$	EX_N	EX_T
✗	C_{pmn}		$A_{pkm}\, B_{pkn}$	EX_T	EX_N
			$A_{pkm}\, B_{pnk}$	EX_T	EX_T

E.g. $C_{mn[p]} = A_{m[p]k}\, B_{[p]nk}$

```
cublasDgemmStridedBatched(handle,
                          CUBLAS_OP_N, CUBLAS_OP_EX_T,
                          M, N, K,
                          &alpha,
                          A, ldA2, ldA1,
                          B, ldB1, ldB2,
                          &beta,
                          C, ldC1, ldC2,
                          P);
```

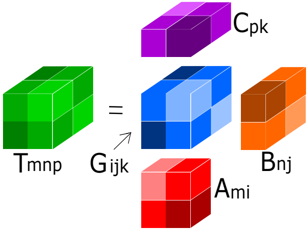
contract

```
contract(cublas::par,
         alpha,
         A, {M,P,K}, _<'m','p','k'>,
         B, {K,N,P}, _<'k','n','p'>,
         beta,
         C,          _<'m','n','p'>);
```

ROWIDX	COLIDX	BATIDX	REDIDX	Kernel	e.g.
0	0	0	0	mult	$C = A\, B$
0	0	0	1	dot	$C = A_k\, B_k$
0	0	1	0	XXX (scalar-mult)	$C_p = A_p\, B_p$
0	0	1	1	XXX (nested-dot?)	$C_p = A_{kp}\, B_{kp}$
0	1	0	0	scal	$C_n = A\, B_n$
0	1	0	1	gemv	$C_n = A_k\, B_{kn}$
0	1	1	0	dgmm (cublas?)	$C_{np} = A_p\, B_{np}$
0	1	1	1	batch_gemm (m=1?)	$C_{np} = A_{kp}\, B_{nkp}$
1	0	0	0	scal	$C_m = A_m\, B$
1	0	0	1	gemv	$C_m = A_{mk}\, B_k$
1	0	1	0	dgmm (cublas?)	$C_{mp} = A_{mp}\, B_p$
1	0	1	1	batch_gemm (n=1?)	$C_{mp} = A_{mkp}\, B_{kp}$
1	1	0	0	ger	$C_{mn} = A_m\, B_n$
1	1	0	1	gemm	$C_{mn} = A_{mk}\, B_{kn}$
1	1	1	0	batch_gemm (k=1?)	$C_{mnp} = A_{mp}\, B_{np}$
1	1	1	1	batch_gemm	$C_{mnp} = A_{mkp}\, B_{nkp}$

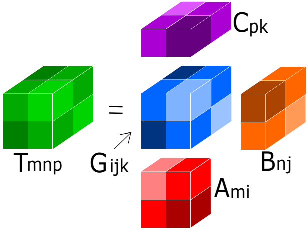
Applications: Tucker Decomposition

$$T_{mnp} = G_{ijk} A_{mi} B_{nj} C_{pk}$$



Applications: Tucker Decomposition

$$T_{mnp} = G_{ijk} A_{mi} B_{nj} C_{pk}$$

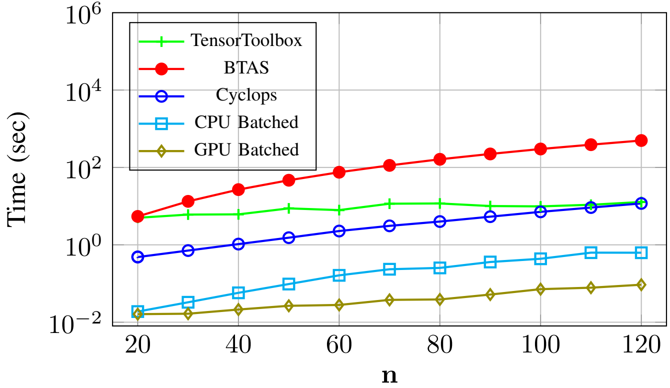


Main steps in the algorithm

- $Y_{mjk} = T_{mnp} B^t_{nj} C^t_{pk}$
- $Y_{ink} = T_{mnp} A^{t+1}_{mi} C^t_{pk}$
- $Y_{ijp} = T_{mnp} B^{t+1}_{nj} A^{t+1}_{mi}$

Applications: Tucker Decomposition

Performance on Tucker decomposition:



Applications: FFT

Low-Communication FFT for multiple GPUs.

- StridedBatchedGEMM composes 75%+ of the runtime.
  - Essential to the performance.
  - Two custom kernels are precisely interleaved GEMMs.

- 2 P100 GPUs: 1.3x over cuFFTXT.
- 8 P100 GPUs: 2.1x over cuFFTXT.

Conclusion

- StridedBatchedGEMM in cuBLAS for generalized tensor contractions.
- Avoid explicit transpositions or permutations.
- **10x**(GPU) and **2x**(CPU) speedup on small/moderate sized tensors.
- **Available in cuBLAS 8.0**

Conclusion

- StridedBatchedGEMM in cuBLAS for generalized tensor contractions.
- Avoid explicit transpositions or permutations.
- **10x**(GPU) and **2x**(CPU) speedup on small/moderate sized tensors.
- **Available in cuBLAS 8.0**
- Future work:
  - Exceptional case kernels/performance/interface??
  - Library Optimizations
    - Matrix stride zero – Persistent Matrix Strided Batched GEMM
    - Staged – RNNs: Staged Persistent Matrix Strided Batched GEMM

Thank you!
Questions?