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

Scalar

Vector

Matrix

Tensor
Modern data is inherently multi-dimensional
Modern data is inherently multi-dimensional
Modern data is inherently multi-dimensional

$$E(x_1 \otimes x_2) = \ldots +$$

$$E(x_1 \otimes x_2 \otimes x_3) = \ldots +$$
What is tensor contraction?
What is tensor contraction?

\[ C_C = A_A \, B_B \]
What is tensor contraction?

\[ C_C = A_A B_B \]

\[ A_{422} \] \[ A(:,1,:) \] \[ A(:,2,:) \] \[ B_{21} \] \[ = \] \[ = \] \[ C_{421} \]

e.g. \[ C_{mnp} = A_{mnk} B_{kp} \]
What is tensor contraction?

\[ C_C = A_A B_B \]

\[ A_{422} \quad A(:,1,:) \quad A(:,2,:) \]
\[ = \]
\[ B_{21} \quad C_{421} \]
\[ = \]

E.g. \[ C_{mnp} = A_{mnk} B_{kp} \]

Why do we need tensor contraction?

1. Core primitive of multilinear algebra.
2. BLAS Level 3: Unbounded compute intensity.
Lots of hot applications at the moment:

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

Topic model

1 Tensor Decompositions for Learning Latent Variable Models, Anima Anandkumar, Rong Ge, Daniel Hsu et. al.
Lots of hot applications at the moment:

- Machine learning
- Deep learning

  e.g. Learning latent variable model with tensor decomposition:

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

$h$: 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.
Distributed FFT

Cris Cecka (NVIDIA)

July 10, 2017
Distributed FFT

\[ T_{pib} = S2T_{ijs}^{(p)} S_{pj(b+s)} \]
\[ M_{pqb} = S2M_{qi} S_{pib} \]
\[ M_{pqb'} = M2M_{qm}^{-} M_{pmb} + M2M_{qm}^{+} M_{pmb}^{+} \]
\[ r_{p} = 1_{ib} S_{pib} = 1_{qb} M_{pqb} \]
\[ L_{pmb} = M2L_{nms}^{(p)} M_{pm(b+s)} \]
\[ L_{pqb} = L2L_{qm}^{b} L_{pmb}^{b} \]
\[ T_{pib} = L2T_{i}^{q} L_{pqb} \]

\[ \implies T_{pib} = S2T_{i(js)}^{(p)} S_{p(js)b} \]
\[ \implies M_{pq[b]} = S_{pi[b]} S2M_{qi}^{T} \]
\[ \implies M_{pq[b']} = M_{PM}[b] M2M_{qM}^{T} \]
\[ \implies r_{p} = 1_{(qb)} M_{p(qb)} \]
\[ \implies L_{pmb} = M2L_{n(ms)}^{(p)} M_{p(ms)b} \]
\[ \implies L_{pq[b]} = L_{pM}[b'] M2M_{qM}^{b} \]
\[ \implies T_{pi[b]} = L_{pq[b]} S2M_{qi} \]
What do we have?
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...
What do we have?

Tensor computation libraries
1. Arbitrary/restricted tensor operation of any order and dimension
   1. Tensor toolbox (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.
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. \( C_{m(pn)} = A_{mk} B_{k(pn)} \)
5. \( C_{mpn} \rightarrow C_{mnp} \)
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. \( C_{m(pn)} = A_{mk} B_{k(pn)} \)
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 Batched GEMM**
- Available in MKL 11.3β and cuBLAS 4.1

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

```c

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)
```

Cris Cecka (NVIDIA)
Existing Primitives

**Pointer-to-Pointer BatchedGEMM**

CUBLAS SGEMM Performance, K40c GPU

CUBLAS SGEMM Performance, P100 GPU

Performance (GFlop/sec)

- Blue: \( N^2 \times N \)
- Red: \( N \) SGEMMs \( N \times N \)
- Green: \( N \) Streamed SGEMMs \( N \times N \)
- Black: BatchedSGEMM \( N \cdot N \times N \)
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:

```c
$ 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);
...```

Cris Cecka (NVIDIA)
cublas\textless T\textgreater 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

\[ C_{mnp} = A_{**} \times B_{***} \]
\[ C_{mnp} \equiv C[m+n \cdot \text{IdC1} + p \cdot \text{IdC2}] \]

<table>
<thead>
<tr>
<th>Case</th>
<th>Contraction</th>
<th>Kernel1</th>
<th>Kernel2</th>
<th>Case</th>
<th>Contraction</th>
<th>Kernel1</th>
<th>Kernel2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>[ A_{mk} B_{knp} ]</td>
<td>[ C_{m(np)} = A_{mk} B_{k(np)} ]</td>
<td>[ C_{mn[p]} = A_{mk} B_{kn[p]} ]</td>
<td>4.1</td>
<td>[ A_{kn} B_{kmp} ]</td>
<td>[ C_{mn[p]} = B_{km[p]} A_{kn} ]</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>[ A_{mk} B_{kpn} ]</td>
<td>[ C_{mn[p]} = A_{mk} B_{k[p]n} ]</td>
<td>[ C_{m[n]p} = A_{mk} B_{kp[n]} ]</td>
<td>4.2</td>
<td>[ A_{kn} B_{kpm} ]</td>
<td>[ C_{mn[p]} = B_{k[p]m} A_{kn} ]</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>[ A_{mk} B_{nkp} ]</td>
<td>[ C_{mn[p]} = A_{mk} B_{nk[p]} ]</td>
<td>[ C_{m[n]p} = A_{mk} B_{kp[n]} ]</td>
<td>4.3</td>
<td>[ A_{kn} B_{mkp} ]</td>
<td>[ C_{mn[p]} = B_{mk[p]} A_{kn} ]</td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>[ A_{mk} B_{pkn} ]</td>
<td>[ C_{m[n]p} = A_{mk} B_{kp[n]} ]</td>
<td>[ C_{m[n]p} = A_{mk} B_{bp[k][n]} ]</td>
<td>4.4</td>
<td>[ A_{kn} B_{pkm} ]</td>
<td>[ C_{mn[p]} = B_{m[p]k} A_{kn} ]</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>[ A_{mk} B_{npk} ]</td>
<td>[ C_{m(np)} = A_{mk} B_{npk} ]</td>
<td>[ C_{mn[p]} = A_{mk} B_{n[p]k} ]</td>
<td>4.5</td>
<td>[ A_{kn} B_{bnk} ]</td>
<td>[ C_{mn[p]} = B_{m[p]k} A_{kn} ]</td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>[ A_{mk} B_{pmk} ]</td>
<td>[ C_{m[n]p} = A_{mk} B_{bp[n][k]} ]</td>
<td>[ C_{m[n]p} = A_{mk} B_{p[n][k]} ]</td>
<td>4.6</td>
<td>[ A_{kn} B_{bnk} ]</td>
<td>[ C_{mn[p]} = B_{m[p]k} A_{kn} ]</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>[ A_{km} B_{knp} ]</td>
<td>[ C_{m(np)} = A_{km} B_{k(np)} ]</td>
<td>[ C_{mn[p]} = A_{km} B_{kn[p]} ]</td>
<td>5.1</td>
<td>[ A_{pk} B_{bkm} ]</td>
<td>[ C_{mn[p]} = B_{km[n]} A_{pk} ]</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>[ A_{km} B_{kpn} ]</td>
<td>[ C_{mn[p]} = A_{km} B_{k[p]n} ]</td>
<td>[ C_{m[n]p} = A_{km} B_{kp[n]} ]</td>
<td>5.2</td>
<td>[ A_{pk} B_{kkm} ]</td>
<td>[ C_{mn[p]} = B_{k[p]m} A_{pk} ]</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>[ A_{km} B_{nkp} ]</td>
<td>[ C_{mn[p]} = A_{km} B_{nk[p]} ]</td>
<td>[ C_{m[n]p} = A_{km} B_{kp[n]} ]</td>
<td>5.3</td>
<td>[ A_{pk} B_{mkn} ]</td>
<td>[ C_{mn[p]} = B_{mk[n]} A_{pk} ]</td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td>[ A_{km} B_{pkn} ]</td>
<td>[ C_{m[n]p} = A_{km} B_{kp[n]} ]</td>
<td>[ C_{m[n]p} = A_{km} B_{bp[k][n]} ]</td>
<td>5.4</td>
<td>[ A_{pk} B_{kmn} ]</td>
<td>[ C_{mn[p]} = B_{m[n]k} A_{pk} ]</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>[ A_{km} B_{npk} ]</td>
<td>[ C_{m(np)} = A_{km} B_{npk} ]</td>
<td>[ C_{mn[p]} = A_{km} B_{n[p]k} ]</td>
<td>5.5</td>
<td>[ A_{pk} B_{bnk} ]</td>
<td>[ C_{mn[p]} = B_{m[n]k} A_{pk} ]</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>[ A_{km} B_{pmk} ]</td>
<td>[ C_{m[n]p} = A_{km} B_{bp[n][k]} ]</td>
<td>[ C_{m[n]p} = A_{km} B_{p[n][k]} ]</td>
<td>5.6</td>
<td>[ A_{pk} B_{bnk} ]</td>
<td>[ C_{mn[p]} = B_{m[n]k} A_{pk} ]</td>
<td></td>
</tr>
</tbody>
</table>
Tensor Contraction with Extended BLAS Primitives

<table>
<thead>
<tr>
<th>Case</th>
<th>Contraction</th>
<th>Kernel1</th>
<th>Kernel2</th>
<th>Kernel3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>$A_{mk}B_{knp}$</td>
<td>$C_{m(np)} = A_{mk}B_{k(np)}$</td>
<td>$C_{mn[p]} = A_{mk}B_{kn[p]}$</td>
<td>$C_{m[n]p} = A_{mk}B_{k[n]p}$</td>
</tr>
<tr>
<td>6.1</td>
<td>$A_{kp}B_{kmn}$</td>
<td>$C_{(mn)p} = B_{k(mn)}^{\top}A_{kp}$</td>
<td>$C_{m[n]p} = B_{km[n]}^{\top}A_{kp}$</td>
<td></td>
</tr>
</tbody>
</table>

Example: Mappings to Level 3 BLAS routines

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

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

Cris Cecka (NVIDIA)
Tensor Contractions cuBLAS
July 10, 2017 15 / 30
Tensor Contraction with Extended BLAS Primitives

<table>
<thead>
<tr>
<th>Case</th>
<th>Contraction</th>
<th>Kernel1</th>
<th>Kernel2</th>
<th>Kernel3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>$A_{mk}B_{knp}$</td>
<td>$C_{m(np)} = A_{mk}B_{k(np)}$</td>
<td>$C_{mn[p]} = A_{mk}B_{kn[p]}$</td>
<td>$C_{m[n]p} = A_{mk}B_{k[n]p}$</td>
</tr>
<tr>
<td>6.1</td>
<td>$A_{kp}B_{kmn}$</td>
<td>$C_{(mn)p} = B_{k(mn)}A_{kp}$</td>
<td>$C_{m[n]p} = B_{km[n]}A_{kp}$</td>
<td></td>
</tr>
</tbody>
</table>

Example: Mappings to Level 3 BLAS routines

- **Case 6.1, Kernel2**: $C_{m[n]p} = B_{km[n]}^T A_{kp}$

  ```c
  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.
Performance

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.
Exceptional Cases:
Cannot be computed by StridedBatchedGEMM.

<table>
<thead>
<tr>
<th>Case</th>
<th>Contraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4</td>
<td>$C_{mnp} = A_{nk}B_{pkm}$</td>
</tr>
<tr>
<td>6.4</td>
<td>$C_{mnp} = A_{kp}B_{nkm}$</td>
</tr>
<tr>
<td></td>
<td>$C_{mnp} = A_{mkp}B_{mkn}$</td>
</tr>
<tr>
<td></td>
<td>$C_{mnp} = A_{pkm}B_{nkp}$</td>
</tr>
</tbody>
</table>

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.
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: \( A_{mnkpqr} \)
  - Reduction mode: \( A_{mnkpqr} \)
  - Aux (batch,row,col) mode: \( A_{mnkpqr} \) (Optional)
- Third tile dimension interpolates between pure GEMM and interleaved GEMM.
- Nested loop over remaining modes performs full contraction.
Tile size tuning with PPCG for exceptional cases:

![Graph showing time vs. n for different blocking configurations](image)

- Time [µs]
- Blocking (m, n)
- m: 1 to 128
- n: 1 to 250

Legend:
- **PPCG**
- **BatchedGemv**
- **BatchedGemm**
- **Geam**
\( C_{mnp} = A_{mkp} B_{nkp} \): 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?

\[ \begin{align*}
&\checkmark \quad C_{mnp} & A_{pmk} B_{pkn} & \text{EX}_N \quad \text{EX}_N \\
&\checkmark \quad C_{mpn} & = & A_{pmk} B_{pnk} & \text{EX}_N \quad \text{EX}_T \\
&\times \quad C_{pmn} & A_{pkm} B_{pkn} & \text{EX}_T \quad \text{EX}_N \\
& & A_{pkm} B_{pnk} & \text{EX}_T \quad \text{EX}_T
\end{align*} \]

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

<table>
<thead>
<tr>
<th>ROWIDX</th>
<th>COLIDX</th>
<th>BATIDX</th>
<th>REDIDX</th>
<th>Kernel</th>
<th>e.g.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>mult</td>
<td>C = AB</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>dot</td>
<td>C = A_k B_k</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>XXX (scalar-mult)</td>
<td>C_p = A_p B_p</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>XXX (nested-dot?)</td>
<td>C_p = A_kp B_kp</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>scal</td>
<td>C_n = A B_n</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>gemv</td>
<td>C_n = A_k B_kn</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>dgmm (cublas?)</td>
<td>C_{np} = A_p B_{np}</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>batch_gemm (m=1?)</td>
<td>C_{np} = A_kp B_{nkp}</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>scal</td>
<td>C_m = A_m B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>gemv</td>
<td>C_m = A_{mk} B_k</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dgmm (cublas?)</td>
<td>C_{mp} = A_{mp} B_p</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>batch_gemm (n=1?)</td>
<td>C_{mp} = A_{mkp} B_{kp}</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>ger</td>
<td>C_{mn} = A_m B_n</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>gemm</td>
<td>C_{mn} = A_{mk} B_{kn}</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>batch_gemm (k=1?)</td>
<td>C_{mnp} = A_{mp} B_{np}</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>batch_gemm</td>
<td>C_{mnp} = A_{mkp} B_{nkp}</td>
</tr>
</tbody>
</table>
\[ T_{mnp} = G_{ijk} A_{mi} B_{nj} C_{pk} \]
Applications: Tucker Decomposition

\[ T_{mnp} = G_{ijk} A_{mi} B_{nj} C_{pk} \]

Main steps in the algorithm

- \( Y_{mjk} = T_{mnp} B_{nj}^t C_{pk}^t \)
- \( Y_{ink} = T_{mnp} A_{mi}^{t+1} C_{pk}^t \)
- \( Y_{ijp} = T_{mnp} B_{nj}^{t+1} A_{mi}^{t+1} \)
Applications: Tucker Decomposition

Performance on Tucker decomposition:

![Graph showing performance comparison of various tools for Tucker decomposition. The x-axis represents the value of n ranging from 20 to 120, and the y-axis represents time in seconds. The graph compares TensorToolbox, BTAS, Cyclops, CPU Batched, and GPU Batched. Each tool is represented by different marker shapes, with TensorToolbox in green, BTAS in red, Cyclops in blue, CPU Batched in cyan, and GPU Batched in yellow. The graph shows how the time increases as the value of n increases for each tool.]
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?