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

Workshop on Batched, Reproducible, and Reduced Precision BLAS

February 25, 2017
Tensor Contraction-Motivation

Scalar  Vector  Matrix  Tensor
Modern data is inherently multi-dimensional
Tensor Contraction - Motivation

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?

\[ C_C = A_A B_B \]

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

\[ C_{mnp} = A_{mnk} B_{kp} \]
Tensor Contraction-Motivation

What is tensor contraction?

$$C_C = A_A B_B$$

![Diagram of tensor contraction](Image)

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.
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
Lots of hot applications at the moment:

- Machine learning
- Deep learning

For example, learning latent variable model with tensor decomposition:

**Topic model**

- $h$: PDF of topics in a document.
- $A$: Topic-word matrix.

$$A_{ij} = \mathcal{P}(x_m = i | y_m = j)$$
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.
Tensor Contraction – Motivation

Distributed FFT
Distributed FFT

\[
\begin{align*}
T_{pi b} &= S2T_{ijs}^{(p)} S_{pj(b+s)} \\
M_{pq b} &= S2M_{qi} S_{pib} \\
M_{pq b'} &= M2M_{qm}^{-} M_{pmb} + M2M_{qm}^{+} M_{pmb} \\
&\quad \mapsto \quad T_{pi b} = S2T_{ijs}^{(p)} S_{p(js)b} \\
&\quad \mapsto \quad M_{pq[b]} = S_{pi[b]} S2M_{qi}^{T} \\
&\quad \mapsto \quad M_{pq[b']} = M_{pM[b]} M2M_{qM}^{T} \\
&\quad \mapsto \quad r_{pi} = 1_{(qb)} M_{p(qb)} \\
L_{pm b} &= M2L_{nm s}^{(p)} M_{pm(b+s)} \\
L_{pq b'} &= L2L_{qm}^{\pm} L_{pmb'} \\
&\quad \mapsto \quad L_{pm b} = M2L_{n(ms)}^{(p)} M_{p(ms)b} \\
&\quad \mapsto \quad L_{pq[b]} = L_{pM[b']} M2M_{qM} \\
&\quad \mapsto \quad T_{pi[b]} = L_{pq[b]} S2M_{qi}
\end{align*}
\]
What do we have?
What do we have?

Tensor computation libraries

1. Arbitrary/restricted tensor operation of any order and dimension
   - Tensor toolbox (Matlab)
   - FTensor (C++)
   - Cyclops (C++)
   - BTAS (C++)
   - All the Python...
Tensor Contraction-Motivation

What do we have?

Tensor computation libraries

1. Arbitrary/restricted tensor operation of any order and dimension
   - Tensor toolbox (Matlab)
   - FTensor (C++)
   - Cyclops (C++)
   - BTAS (C++)
   - All the Python...

Efficient computing frame

1. Static analysis solutions
   - PPCG [ISL] (polyhedral)
   - TCE (DSL)

2. Parallel and distributed primitives
   - BLAS, cuBLAS
   - 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. $C_{mpn} \rightarrow C_{mnp}$
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

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

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

CUBLAS SGEMM Performance, K40c GPU

CUBLAS SGEMM Performance, P100 GPU
Existing Primitives

Pointer-to-Pointer BatchedGEMM

Except actually...

Solution: StridedBatchedGEMM

CUBLAS SGEMM Performance, K40e GPU

CUBLAS SGEMM Performance, P100 GPU

[Graphs showing performance of different GEMM operations on K40e and P100 GPUs]
Exits!

... Still no documentation?!?

Documentation as of last Tuesday!
$$ \text{grep StridedBatched -A 17 /usr/local/cuda/include/cublas_api.h} $$

```c
CUBLASAPI cublasStatus_t cublasSgemmStridedBatched (cublasHandle_t handle,
  cublasOperation_t transa,
  cublasOperation_t transb,
  int m,
  int n,
  int k,
  const float *alpha, // host or device pointer
  const float *A,
  int lda,
  long long int strideA, // purposely signed
  const float *B,
  int ldb,
  long long int strideB,
  const float *beta, // host or device pointer
  float *C,
  int ldc,
  long long int strideC,
  int batchCount);
```

...
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

\[ C_{mnp} = A_{**} \times B_{***} \]

\[ C_{mnp} \equiv C[m + n \cdot \text{IdC1} + p \cdot \text{IdC2}] \]

<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_{mn}[p] = A_{mk} B_{k[p]n} )</td>
<td>( C_{mnp} = 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>( C_{mnp} = B_{km[p]} A_{kn} )</td>
</tr>
<tr>
<td>1.2</td>
<td>( A_{mk} B_{kpn} )</td>
<td>( C_{mnp} = A_{mk} B_{k[p]n} )</td>
<td>( C_{mnp} = A_{mk} B_{kn[p]} )</td>
<td>4.2</td>
<td>( A_{kn} B_{kpm} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kn} )</td>
<td>( C_{mnp} = B_{km[p]} A_{kn} )</td>
</tr>
<tr>
<td>1.3</td>
<td>( A_{mk} B_{npk} )</td>
<td>( C_{mnp} = A_{mk} B_{nk[p]} )</td>
<td>( C_{mnp} = A_{mk} B_{kn[p]} )</td>
<td>4.3</td>
<td>( A_{kn} B_{mkp} )</td>
<td>( C_{mn}[p] = B_{mk[p]} A_{kn} )</td>
<td>( C_{mnp} = B_{mk[p]} A_{kn} )</td>
</tr>
<tr>
<td>1.4</td>
<td>( A_{mk} B_{pkn} )</td>
<td>( C_{mnp} = A_{mk} B_{nk[p]} )</td>
<td>( C_{mnp} = A_{mk} B_{kn[p]} )</td>
<td>4.4</td>
<td>( A_{kn} B_{pkm} )</td>
<td>( C_{mn}[p] = B_{mk[p]} A_{kn} )</td>
<td>( C_{mnp} = B_{mk[p]} A_{kn} )</td>
</tr>
<tr>
<td>1.5</td>
<td>( A_{mk} B_{npk} )</td>
<td>( C_{mnp} = A_{mk} B_{nk[p]} )</td>
<td>( C_{mnp} = A_{mk} B_{kn[p]} )</td>
<td>4.5</td>
<td>( A_{kn} B_{mpk} )</td>
<td>( C_{mn}[p] = B_{m[p]k} A_{kn} )</td>
<td>( C_{mnp} = B_{m[p]k} A_{kn} )</td>
</tr>
<tr>
<td>1.6</td>
<td>( A_{mk} B_{pnm} )</td>
<td>( C_{mnp} = A_{mk} B_{nk[p]} )</td>
<td>( C_{mnp} = A_{mk} B_{kn[p]} )</td>
<td>4.6</td>
<td>( A_{kn} B_{npm} )</td>
<td>( C_{mn}[p] = B_{mk[p]} A_{kn} )</td>
<td>( C_{mnp} = B_{mk[p]} A_{kn} )</td>
</tr>
<tr>
<td>2.1</td>
<td>( A_{km} B_{knp} )</td>
<td>( C_{m(np)} = A_{km} B_{k(p)n} )</td>
<td>( C_{m(np)} = A_{km} B_{k[p]n} )</td>
<td>5.1</td>
<td>( A_{pk} B_{kmn} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{pk} )</td>
<td>( C_{m(np)} = B_{km[p]} A_{pk} )</td>
</tr>
<tr>
<td>2.2</td>
<td>( A_{km} B_{kpn} )</td>
<td>( C_{mnp} = A_{km} B_{k[p]n} )</td>
<td>( C_{mnp} = A_{km} B_{kn[p]} )</td>
<td>5.2</td>
<td>( A_{pk} B_{kmn} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{pk} )</td>
<td>( C_{mnp} = B_{km[p]} A_{pk} )</td>
</tr>
<tr>
<td>2.3</td>
<td>( A_{km} B_{npk} )</td>
<td>( C_{mnp} = A_{km} B_{nk[p]} )</td>
<td>( C_{mnp} = A_{km} B_{kn[p]} )</td>
<td>5.3</td>
<td>( A_{pk} B_{kmn} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{pk} )</td>
<td>( C_{mnp} = B_{km[p]} A_{pk} )</td>
</tr>
<tr>
<td>2.4</td>
<td>( A_{km} B_{pkn} )</td>
<td>( C_{mnp} = A_{km} B_{nk[p]} )</td>
<td>( C_{mnp} = A_{km} B_{kn[p]} )</td>
<td>5.4</td>
<td>( A_{pk} B_{kmn} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{pk} )</td>
<td>( C_{mnp} = B_{km[p]} A_{pk} )</td>
</tr>
<tr>
<td>2.5</td>
<td>( A_{km} B_{npk} )</td>
<td>( C_{m(np)} = A_{km} B_{k(p)n} )</td>
<td>( C_{m(np)} = A_{km} B_{k[p]n} )</td>
<td>5.5</td>
<td>( A_{pk} B_{kmn} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{pk} )</td>
<td>( C_{m(np)} = B_{km[p]} A_{pk} )</td>
</tr>
<tr>
<td>2.6</td>
<td>( A_{km} B_{pnm} )</td>
<td>( C_{m(np)} = A_{km} B_{k(p)n} )</td>
<td>( C_{m(np)} = A_{km} B_{k[p]n} )</td>
<td>5.6</td>
<td>( A_{pk} B_{kmn} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{pk} )</td>
<td>( C_{m(np)} = B_{km[p]} A_{pk} )</td>
</tr>
<tr>
<td>3.1</td>
<td>( A_{nk} B_{kmp} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{nk} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{nk} )</td>
<td>6.1</td>
<td>( A_{kp} B_{kmn} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kp} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kp} )</td>
</tr>
<tr>
<td>3.2</td>
<td>( A_{nk} B_{kpm} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{nk} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{nk} )</td>
<td>6.2</td>
<td>( A_{kp} B_{kmn} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kp} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kp} )</td>
</tr>
<tr>
<td>3.3</td>
<td>( A_{nk} B_{mkp} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{nk} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{nk} )</td>
<td>6.3</td>
<td>( A_{kp} B_{kmn} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kp} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kp} )</td>
</tr>
<tr>
<td>3.4</td>
<td>( A_{nk} B_{pkm} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{nk} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{nk} )</td>
<td>6.4</td>
<td>( A_{kp} B_{kmn} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kp} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kp} )</td>
</tr>
<tr>
<td>3.5</td>
<td>( A_{nk} B_{mpk} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{nk} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{nk} )</td>
<td>6.5</td>
<td>( A_{kp} B_{kmn} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kp} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kp} )</td>
</tr>
<tr>
<td>3.6</td>
<td>( A_{nk} B_{pmk} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{nk} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{nk} )</td>
<td>6.6</td>
<td>( A_{kp} B_{kmn} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kp} )</td>
<td>( C_{mn}[p] = B_{km[p]} A_{kp} )</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>$C_{(mn)p} = B_{k(mn)}^T A_{kp}$</td>
<td>$C_{m[n]p} = B_{km[n]}^T A_{kp}$</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,
      1.0,
      A, ldA1, 0,
      B, ldB1, ldB2,
      0.0,
      C, ldC1, ldC2,
      P)
```
## 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>$C_{(mn)p} = B_{k(mn)}^T A_{kp}$</td>
<td>$C_{m[n]p} = B_{km[n]}^T A_{kp}$</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,
                             1.0,
                             B, ldB1, ldB2,
                             A, lda1, 0,
                             0.0,
                             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.
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:

```
(1,1) (2,1) (4,1) (8,1) (16,1) (32,1) (64,1) (128,1)
```

Blocking $(m, n)$

Blocking $(p, k)$

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

- 0 50 100 150 200 250
- $10^1$ $10^2$ $10^3$ $10^4$ $10^5$ $10^6$

**Time [μs]**

**PPCG**

**BATCHEDGEMV**

**BATCHEDGEMM**

**GEAM**

Cris Cecka (NVIDIA)

Tensor Contractions cuBLAS

February 25, 2017
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)
Extend the StridedBatchedGEMM transpose parameters?

\[
\begin{align*}
\checkmark & \quad C_{mnp} & A_{pmk} B_{pkn} & \text{EX}_N \quad \text{EX}_N \\
\checkmark & \quad C_{mpn} & A_{pmk} B_{pnk} & \text{EX}_N \quad \text{EX}_T \\
\times & \quad C_{pmn} & A_{pkm} B_{pkn} & \text{EX}_T \quad \text{EX}_N \\
          & \quad & A_{pkm} B_{pnk} & \text{EX}_T \quad \text{EX}_T
\end{align*}
\]
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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>dot</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>XXX ( c_p = a_p b_p )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>XXX ( c_p = a_{pk} b_{pk} )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>scal</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>gemv</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>dgmm</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>batch_gemm (XXX: exceptional)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>ger</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>gemm</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>XXX (batch_gemm(K = 1)? batch_ger?)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>batch_gemm (XXX: exceptional)</td>
</tr>
</tbody>
</table>
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

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

Performance on Tucker decomposition:

![Graph showing performance comparison of different methods for Tucker decomposition.](image-url)
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 cuFFT TXT.
- 8 P100 GPUs: 2.1x over cuFFT TXT.
Conclusion

- StridedBatchedGEMM in cuBLAS for generalized tensor contractions.
- Avoid explicit transpositions or permutations.
- \textbf{10x}(GPU) and \textbf{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?