Tensor Contractions with Extended BLAS Kernels on CPU and GPU

Yang Shi, U.N. Niranjan, Animashree Anandkumar[1] Cris Cecka[2]

[1]UCIrvine [2]NVIDIA Research

Abstract

- Existing tensor contractions approaches involve explicit copy and transpose operations.
- We focus on single-index contractions involving all the possible configurations of second-order and third-order tensors and discuss extensions to more general cases.
- Our approach achieve 10x speedup on a K40c GPU and 2x speedup on dual-socket Haswell-EP CPUs, using MKL and CUBLAS respectively.
- Our kernels yields atleast an order of magnitude speedup as compared to state-of-the-art libraries for Tucker decomposition.

Conventional Tensor Contraction

The conventional approach for tensor contraction is to matricize the tensors via transpositions and copies. Libraries such as Basic Tensor Algebra Subroutines (BTAS), MATLAB Tensor Toolbox, and Cyclops Tensor Framework all perform some version of matricization, which is typically performed in four steps:

- Consider a general tensor contraction of the form $C_{\mathcal{C}} = \alpha A_{\mathcal{A}} B_{\mathcal{B}} + \beta C_{\mathcal{C}}$. Define the index sets $\mathcal{K}$, $\mathcal{I}$, $\mathcal{J}$ as
  $\mathcal{K} = \mathcal{A} \cap \mathcal{B}, \quad \mathcal{I} = \mathcal{A} \setminus (\mathcal{A} \cap \mathcal{B}), \quad \mathcal{J} = \mathcal{B} \setminus (\mathcal{A} \cap \mathcal{B})$
- Permute tensors A, B, and C into the form
  $$C_{\mathcal{I}\mathcal{J}} = \alpha\, A_{\mathcal{I}\mathcal{K}}\, B_{\mathcal{K}\mathcal{J}} + \beta\, C_{\mathcal{I}\mathcal{J}} \qquad (1)$$
- Evaluate (1) using one of four BLAS kernels:
  DOT: if $|\mathcal{K}| = |\mathcal{A}|and|\mathcal{K}| = |\mathcal{B}|$
  GER: if $|\mathcal{K}| = 0$
  GEMV: if $|\mathcal{K}| = |\mathcal{A}|xor|\mathcal{K}| = |\mathcal{B}|$
  GEMM: if else
- Permute the result $C_{\mathcal{I}\mathcal{J}}$ into the desired output $C_{\mathcal{C}}$.

This approach works for any two tensors of arbitrary order and any number of contraction indices. However, the cost of explicitly permutation outweigh the cost of the computation to be performed.

Motivating Observations

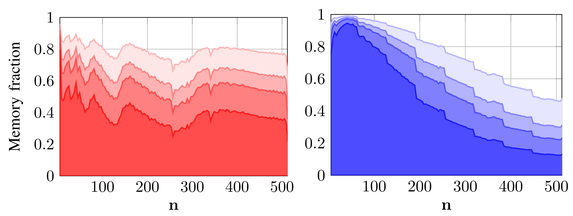

Figure 1: The fraction of time spent in copies/transpositions when computing the contraction $C_{mnp} = A_{mk}B_{pkn}$ using the conventional approach. Lines are shown with 1, 2, 3, and 6 total transpositions performed on either the input or output. (Left) CPU. (Right) GPU.

Strided Batched GEMM

C = alpha*opA(A)*opB(B) + beta*C

```
sb_gemm(op_type opA, op_type opB, int m, int
n, int k, T alpha, const T* A, int lda, int loa,
const T* B, int ldb, int lob, T beta, T* C, int
ldc, int loc, int batch_size)
{
   for (int p = 0; p < batch_size; ++p)
      gemm(opA, opB, m, n, k,alpha, A + p*loa,
lda, B + p*lob, ldb, beta, C + p*loc, ldc);
}
```

Single-mode Contractions Between a Second-order/Third-order Tensor

Case	Contraction	Kernel1	Kernel2	Kernel3	Case	Contraction	Kernel1	Kernel2
1.1	$A_{mk}B_{knp}$	$C_{m(np)} = A_{mk}B_{k(np)}$	$C_{mn[p]} = A_{mk}B_{kn[p]}$	$C_{m[n]p} = A_{mk}B_{k[n]p}$	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	$A_{kn}B_{pkm}$	$TRANS(A^{\top}_{kn}B^{\top}_{pk[m]})$	$C_{[m][n]p} = B_{pk[m]}A_{k[n]}$
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	$A_{mk}B_{pmk}$	$C_{m[n]p} = A_{mk}B^{\top}_{p[n]k}$			4.6	$A_{kn}B_{pmk}$	$TRANS(A^{\top}_{kn}B^{\top}_{p[m]k})$	$C_{[m][n]p} = B_{p[m]k}A_{k[n]}$
2.1	$A_{km}B_{knp}$	$C_{mn(p)} = A^{\top}_{km}B_{k(np)}$	$C_{mn[p]} = A^{\top}_{km}B_{kn[p]}$	$C_{m[n]p} = A^{\top}_{km}B_{k[n]p}$	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	$A_{pk}B_{nkm}$	$TRANS(B_{nk[m]}A^{\top}_{pk})$	$C_{[m]n[p]} = B_{nk[m]}A_{[p]k}$
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	$A_{pk}B_{nmk}$	$TRANS(B_{n[m]k}A^{\top}_{pk})$	$C_{[m]n[p]} = B_{n[m]k}A_{[p]k}$
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	$A_{nk}B_{pkm}$	$TRANS(A_{nk}B^{\top}_{pk[m]})$	$C_{[m][n]p} = B_{pk[m]}A_{[n]k}$		6.4	$A_{kp}B_{nkm}$	$TRANS(B_{nk[m]}A_{kp})$	$C_{[m]n[p]} = B_{nk[m]}A_{k[p]}$
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	$A_{nk}B_{pmk}$	$TRANS(A_{nk}B^{\top}_{p[m]k})$	$C_{[m][n]p} = B_{p[m]k}A_{[n]k}$		6.6	$A_{kp}B_{nmk}$	$TRANS(B_{n[m]k}A_{kp})$	$C_{[m]n[p]} = B_{n[m]k}A_{k[p]}$

Table 1: List of 36 possible single mode contraction operations between a second-order tensor and a third-order tensor and possible mappings to Level-3 BLAS routines.

SBGEMM v.s. Flattened GEMM

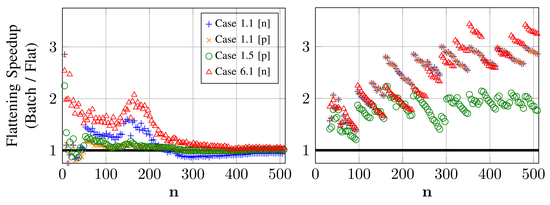

Figure 2: Performance ratio for a BATCHEDGEMM over a flattened GEMM in evaluation of Cases 1.1, 1.5, and 6.1.(Left) CPU. (Right) GPU.

Batching over different modes

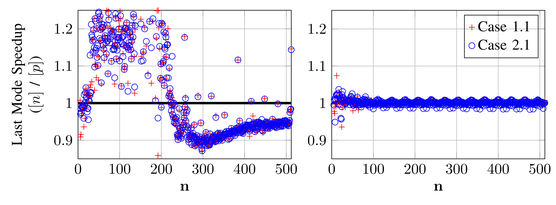

Figure 3: Speedup obtained from batching in the last mode,[p], rather than the middle mode, [n], for Cases 1.1 and 2.1.(Left) CPU. (Right) GPU.

Performance on Tucker Decomposition

In the Einstein notation, the factorization of a third-order tensor $T \in \mathbb{R}^{\mathbf{m}\times\mathbf{n}\times\mathbf{p}}$ is given by $T_{mnp} = G_{ijk}A_{mi}B_{nj}C_{pk}$, where $G \in \mathbb{R}^{\mathbf{i}\times\mathbf{j}\times\mathbf{k}}$ is the core tensor, $A \in \mathbb{R}^{\mathbf{m}\times\mathbf{i}}$, $B \in \mathbb{R}^{\mathbf{n}\times\mathbf{j}}$, $C \in \mathbb{R}^{\mathbf{p}\times\mathbf{k}}$.
A variety of problems in unsupervised learning such as topic model estimation, Gaussian mixtures model estimation, and social network learning can be solved via the tensor decomposition techniques under certain mild assumptions.

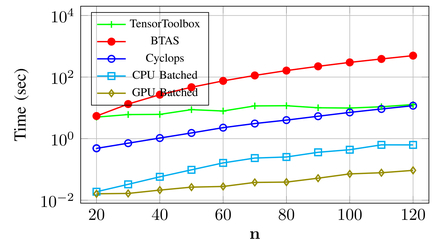

Figure 4: Performance on Tucker decompostion.

Evaluation Properties

There are a number of heuristics that may be important in constructing the most efficient evaluation strategy.

- Flatten modes whenever possible. A single large GEMM is more efficient.
- In the interest of performing the highest intensity computation within a SBGEMM, we recommend performing the largest GEMMs possible within a SBGEMM and batching in the mode with largest dimension.
- Preferring to batch in the last mode versus earlier modes can depend on the input parameters and machine.

Conclusion and Future Work

Our improvement is most significant on small and moderate sized tensors. This is important because in many applications, e.g. deep learning for training a recursive tensor network, we require evaluating a large number of tensor contractions of small sizes. Although we focused on single-node performance, these evaluations may be used as a building blocks for distributed memory implementations, which we intent to pursue as part of our future work.