Tensor Transposition and Contraction on GPUs*

Jinsung Kim¹  Aravind Sukumaran Rajam¹  Ajay Panyala²
Sriram Krishnamoorthy², P. Sadayappan¹

¹The Ohio State University
²Pacific Northwest National Laboratory

*Funded by US Department of Energy and US National Science Foundation
Tensor Contraction

\[ C_{ijkl} = \sum_{mn} A_{imkn} \cdot B_{jnlm} \]

for (i=0; i<N; i++)
for (j=0; j<N; j++)
for (k=0; k<N; k++)
for (l=0; l<N; l++)
for (m=0; m<N; m++)
for (n=0; n<N; n++)
    \( C[i][j][k][l] += A[i][m][k][n] \cdot B[j][n][l][m]; \)

- Tensor contraction is high-dimension analog of matrix-matrix product
- Each loop index appears in exactly two tensors
  - “Contraction index” appears only in input (rhs) tensors: \{m, n\}
  - “External index”: appears in output and one input tensor: \{i, k\} \{j, l\}
TTGT: Tensor Contraction Using Matrix Multiplication

\[ C_{ijkl} = \sum_{mn} A_{imkn} \cdot B_{jnlm} \]

for (i=0; i<N; i++)
for (j=0; j<N; j++)
for (k=0; k<N; k++)
for (l=0; l<N; l++)
for (m=0; m<N; m++)
for (n=0; n<N; n++)
\[ C[i][j][k][l] += A[i][m][k][n] \cdot B[j][n][l][m]; \]

TTGT: Transpose-Transpose-GEMM-Transpose

Index Permutation

T1[i][k][m][n]
\[ T1[ik][mn] \]

T2[m][n][j][l]
\[ T2[mn][jl] \]

Library Mat-Mult

T3[i][k][j][l]  T3[ik][jl]
TTLG: Tensor Transposition Library for GPUs

• Efficient in-GPU transposition of tensors of arbitrary size/shape
  \[ B_{j_N, \ldots, j_2, j_1, j_0} \leftarrow A_{\pi(j_N, \ldots, j_2, j_1, j_0)} \]

• In addition to comp. chemistry (e.g., NWChemEx), tensor contractions also needed in other domains, e.g., finite-elements, machine learning

• Library internally selects variant from multiple implemented schemes
  • Fastest-varying indices on input/output tensor match/non-match
  • Tensor extents along different dimensions

• Internal cost model predicts best variant for each invocation instance
  • Regression-based cost model built at library installation time on GPU platform

• Queryable performance prediction model for any given transposition
  • Regression-based model built at library installation time
  • Useful for building tensor-contraction library using index permutation library: TTGT (Transpose, Transpose, GEMM, Transpose)
Matrix Transposition on GPU

```
\text{Out}[i0][i1] = \text{In}[i1][i0]
```

- 2-D Shared Memory buffer with padding (avoid bank conflicts)
  - Coalesced read: Sbuf[is][0:31] = In[i][j:j+31]
  - Coalesced write: Out[j][i:i+31] = Sbuf[0:31][js]
Generalizing Matrix-Transposition to Tensor-Transposition

- \( B_{o_N, \ldots, o_2, o_1, o_0} \leftarrow A_{i_N, \ldots, i_2, i_1, i_0} \)

- If \( o_0 \) is different than \( i_0 \), the matrix-transpose scheme can be generalized for tensor-transpose
  - Contiguous distinct indices can be combined in input/output: form fused indices with larger extent
  - Ex: \( \text{Out}[i_0][i_1][i_2] = \text{In}[i_2][i_1][i_0] \)
  - \( \text{Out}[9][15][128] = \text{In}[128][15][9] \)
  - \( \text{Out}[9][15][128] = \text{In}[128][135] \)

- But different schemes needed if:
  - \( o_0 = i_0 \): scheme is inapplicable
  - \( o_0 \neq i_0 \), but extents \(< < 32\): inefficient
Taxonomy of Transposition Schemes

Four schemes:
- FVI-Match-Small
- FVI-Match-Large
- Orthogonal-Distinct
- Orthogonal-Arbitrary

1. Permutation, Sizes
2. Index Fusion
3. Permutation(0) = 0?
   - Yes
     - size(0)*size(1) ≥ 32 and size(ρ[0])*size(ρ[1]) ≥ 32
   - No
     - size(0) < 32?
       - Yes
         - FVI-Match-Small: 32x33 shared-mem buffer; dim-blocking
       - No
         - size(ρ[0])*size(ρ[1]) ≥ 32
1. Common indices in slices?
   - Yes
     - Orthogonal-Distinct: 32x33 shared-mem buffer; dim-blocking
   - No
     - Orthogonal-Arbitrary: Variable sized shared-mem buffer; dim-blocking
2. FVI-Match-Large: Direct transpose using registers
3. FVI-Match-Small/Orthogonal-Arbitrary: Variable sized shared-mem buffer; dim-blocking
4. FVI-Match-Small: 32x33 shared-mem buffer; dim-blocking
Experimental Evaluation: Tensor Transposition

- Related work compared
  - cuTT (CUDA Tensor Transpose) [Hynninen and Lyakh, Arxiv 2017] cuTT: A High-Performance Tensor Transpose Library for CUDA Compatible GPUs
  - TTC (Tensor Transposition Compiler) [Springer, Hammond and Bientinesi, Arxiv 2016] TTC: A high-performance Compiler for Tensor Transpositions

- Test cases
  - 57 TTC benchmarks
  - All 6! = 720 permutations of a 6D tensor

- Platform
  - NVIDIA Tesla K40c (15 Kepler SMs, 192 cores/MP, 12 GB Global Memory, 745 MHz, 15 MB L2 cache, ECC off)
Experimental Evaluation: TTC Benchmarks

- Related work
  - cuTT has two modes:
    1. Measure: Executes several choices on GPU and selects best (repeated-use)
    2. Heuristic: Quick parameter determination (single-use)
  - TTC: generates customized code for each benchmark

- TTLG outperforms both cuTT-heuristic and cuTT-measure.
- TTC was found to be slower than the library based approaches.

Jyothi Vedurada
May 23, 2018 21 / 22
Experimental Evaluation: Permutations of 6D Tensor (720)

- 16^6 sized tensor; all 720 possible permutations
- Different scaled ranks: if adjacent dimensions are same in output, merged, e.g., \([5 4 3 2 1 0] \Rightarrow [0 2 5 4 1 3] \Rightarrow [4' 3 2 1 0] \Rightarrow [0 2 4' 1 3]\)

Repeated-Use Scenario

- TTLG
- cuTT Heuristic
- cuTT Measure
- TTC
- Scaled Rank

Single-Use Scenario

- TTLG
- cuTT Heuristic
- cuTT Measure
- TTC
- Scaled Rank
Experimental Evaluation: Tensor Contraction

◆ Related work compared
  - TAL_SH (Lyakh, Oak Ridge): TTGT using cuTT and cuBLAS
  - TTLG + cuBLAS
  - GETT (Springer & Bientinesi): Direct tensor contraction [Multicore CPU]
  - TBLIS (Matthews): Direct tensor contraction [Multicore CPU]

◆ Test cases
  - 24 TCCG benchmarks
Experimental Evaluation: Tensor Contraction

Figure 12: Comparison of Contraction Results

Figure 13: GFLOPS of contraction schemes for different problem sizes.
Code Generator for Direct Tensor Contractions on GPUs

◆ Schema for Direct Contraction

- Generalization of register-tiled shared-memory buffered 2D thread-block strategy used for efficient dense matrix multiplication on GPU
- Customized GPU kernel is generated for each tensor contraction, based on dimensionality of input/output tensors and matching of indices
Matrix Multiplication Schema

\[ C[i][j] += A[i][k] \times B[k][j] \]

- A 2D thread-block computes a 2D slice \( (T_i \times T_j) \) of \( C \) using a \( T_i \times N_k \) slice of \( A \) and a \( N_k \times T_j \) slice of \( B \).
- Registers are used to hold the \( T_i \times T_j \) slice of \( C \).
- A \( T_i \times T_k \) slice of \( A \) and a \( T_k \times T_j \) slice of \( B \) are loaded into Shared Memory (1).
Matrix Multiplication Schema

\[ C[i][j] += A[i][k] \times B[k][j] \]

- A Ti\times Tk slice of A and a Tk\times Tj slice of B are loaded into Shared Memory (1)
- A column-slice of A and row-slice of B are loaded from shared memory to registers (2)
- Outer-product contribution added to slice of C (3)
- Slice of C is written out to global memory (4)
Generalization for Arbitrary Tensor Contractions

\[ t_3[k, j, i, a, b, c] = t_2[i, j, a, d] \times v_2[d, k, c, b] \]

- Global Memory (GMEM)
- Shared Memory (SMEM)
- Registers (REG)

- From GMEM to SMEM
- From SMEM to REG
- From REG to GMEM
Performance data for 24 TCCG benchmarks

- Direct contraction is often faster but not always
Summary

- TTLG GPU tensor transposition (index permutation) library
  - Developed four schemas to cover all possible scenarios for arbitrary dimensional tensor and arbitrary permutation
  - Internal cost model dynamically selects best schema and slicing parameters to use for any transpose call
  - Comparable/better performance than other GPU transpose alternatives
  - Tensor contraction library can be layered over index-permutation library

- Direct (transpose-free) tensor contraction on GPUs
  - Generalization of register-tiled 2D schema for efficient matrix multiplication
  - Generally but not always faster than TTGT approach
  - Advantage over TTGT: no space overhead for storing intermediate tensors
  - Future work: develop model to automatically choose between direct and TTGT schemes for tensor contractions