CUTENSOR

High-Performance CUDA Tensor Primitives

Paul Springer, Chen-Han Yu, March 20th 2019

NVIDIA

pspringer@nvidia.com and chenhany@nvidia.com

ACKNOWLEDGMENTS

- Colleagues at NVIDIA
  - Albert Di
  - Alex Fit-Florea
  - Evghenii Gaburov
  - Harun Bayraktar
  - Sharan Chetlur
  - Timothy Costa
  - Zachary Zimmerman
- Collaborators outside of NVIDIA
  - Dmitry Liakh (TAL-SH)
  - Jutho Haegeman (Julia)
  - Tim Besard (Julia)

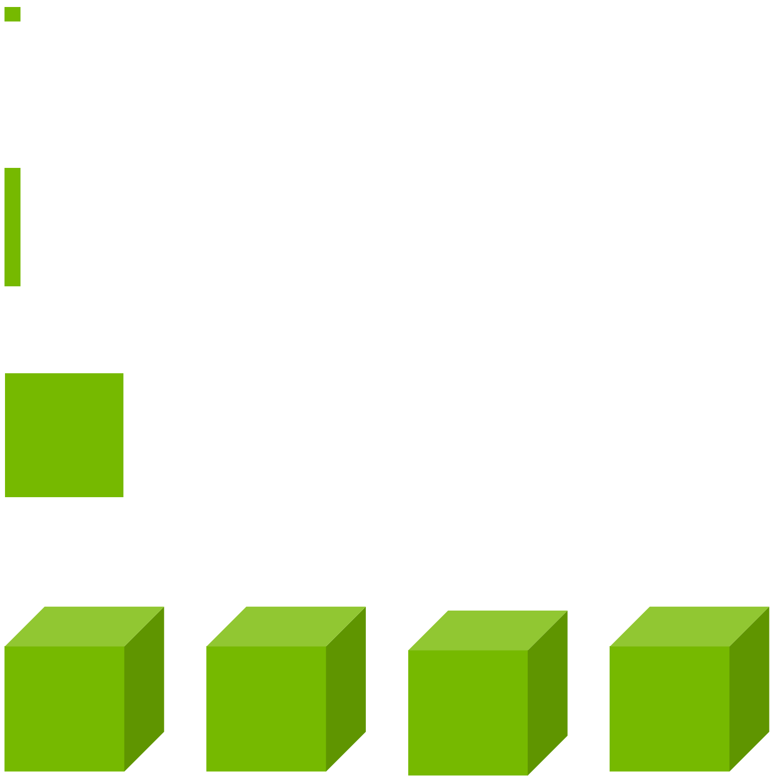
*alphabetic order

WHAT IS A TENSOR?

- mode-0: scalar α
- mode-1: vector A_i
- mode-2: matrix $A_{i,j}$
- mode-n: general tensor $A_{i,j,k}$

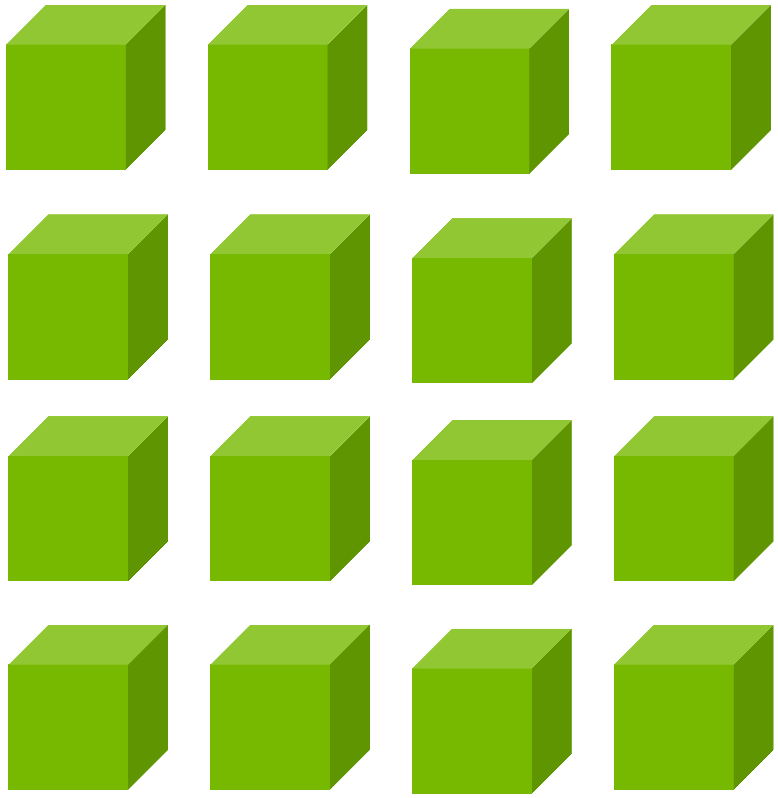
WHAT IS A TENSOR?

- mode-0: scalar α
- mode-1: vector A_i
- mode-2: matrix $A_{i,j}$
- mode-n: general tensor $A_{i,j,k,l}$



WHAT IS A TENSOR?

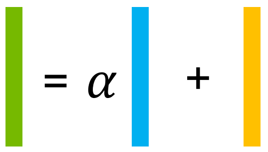
- mode-0: scalar α
- mode-1: vector A_i
- mode-2: matrix $A_{i,j}$
- mode-n: general tensor $A_{i,j,k,l,m}$



BASIC LINEAR SUBPROGRAMS

A Success Story

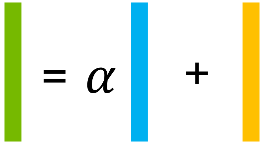
- 1969 – BLAS Level 1: Vector-Vector



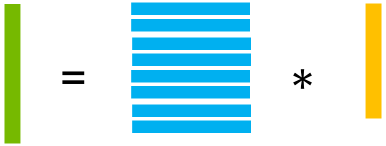
BASIC LINEAR SUBPROGRAMS

A Success Story

- 1969 – BLAS Level 1: Vector-Vector



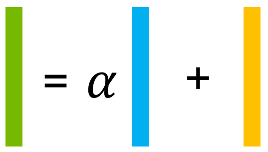
- 1972 – BLAS Level 2: Matrix-Vector



BASIC LINEAR SUBPROGRAMS

A Success Story

- 1969 – BLAS Level 1: Vector-Vector

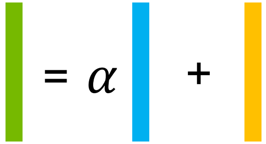


- 1972 – BLAS Level 2: Matrix-Vector

BASIC LINEAR SUBPROGRAMS

A Success Story

- 1969 – BLAS Level 1: Vector-Vector



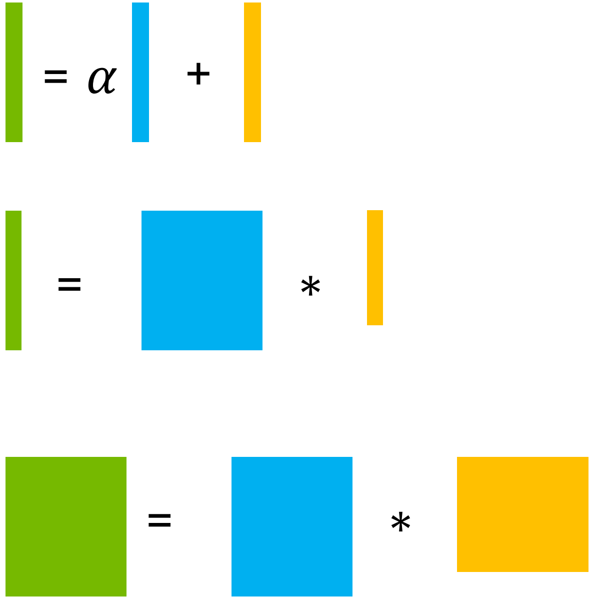
- 1972 – BLAS Level 2: Matrix-Vector

- 1980 – BLAS Level 3: Matrix-Matrix

BASIC LINEAR SUBPROGRAMS

A Success Story

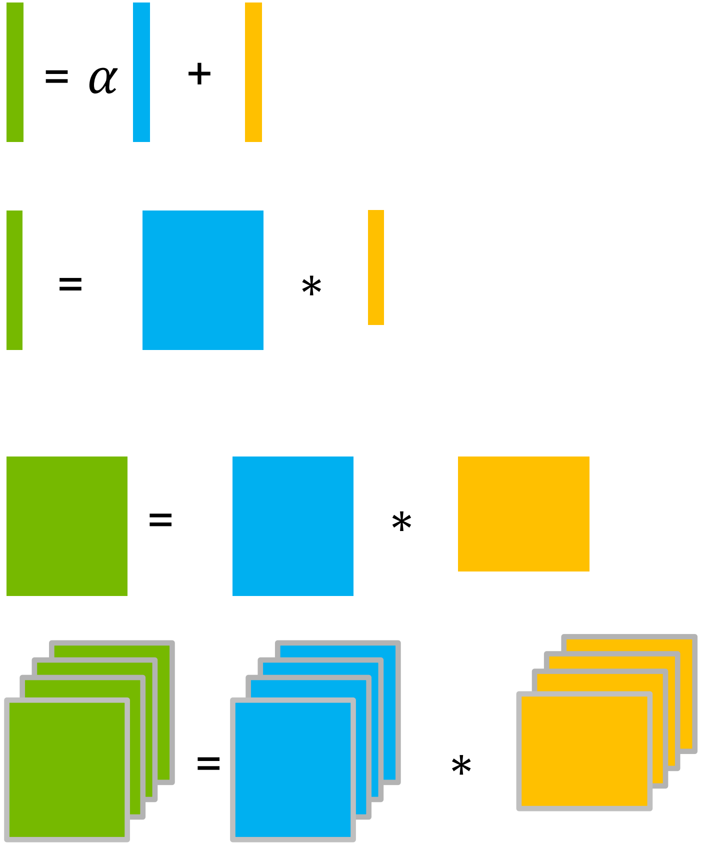
- 1969 – BLAS Level 1: Vector-Vector
- 1972 – BLAS Level 2: Matrix-Vector
- 1980 – BLAS Level 3: Matrix-Matrix



BASIC LINEAR SUBPROGRAMS

A Success Story

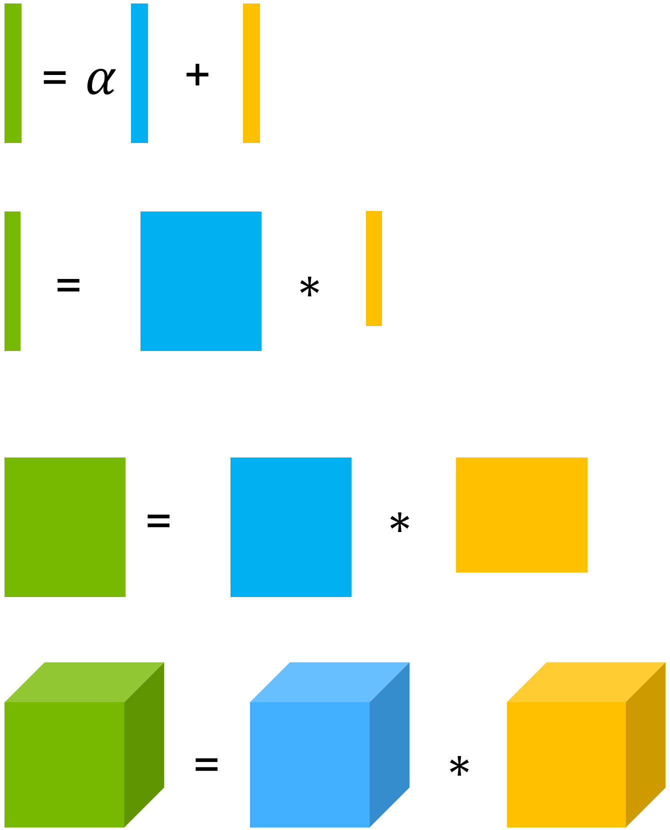
- 1969 – BLAS Level 1: Vector-Vector
- 1972 – BLAS Level 2: Matrix-Vector
- 1980 – BLAS Level 3: Matrix-Matrix
- Now? – BLAS Level 4: Tensor-Tensor



BASIC LINEAR SUBPROGRAMS

A Success Story

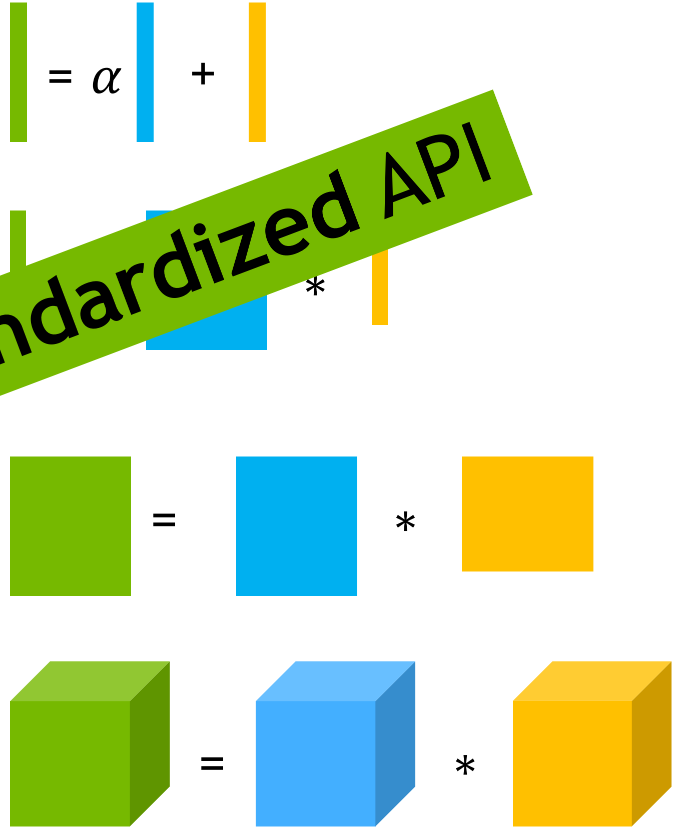
- 1969 – BLAS Level 1: Vector-Vector
- 1972 – BLAS Level 2: Matrix-Vector
- 1980 – BLAS Level 3: Matrix-Matrix
- Now? – BLAS Level 4: Tensor-Tensor



BASIC LINEAR SUBPROGRAMS

A Success Story

- 1969 – BLAS Level 1: Vector-Vector
- 1972 – BLAS Level 2: Matrix-Vector
- 1980 – BLAS Level 3: Matrix-Matrix
- Now? – BLAS Level 4: Tensor-Tensor



Key for success: Standardized API

TENSORS ARE UBIQUITOUS

Potential Use Cases

Deep Learning

PyTorch

PYRO

cuDNN

TensorFlow

TensorLy

Quantum Chemistry

NWCHEM
HIGH-PERFORMANCE COMPUTATIONAL CHEMISTRY SOFTWARE

LS-DALTON

TAL-SH

- Multi-GPU
- Out-of-Core

Condensed Matter Physics

ITENSOR

julia

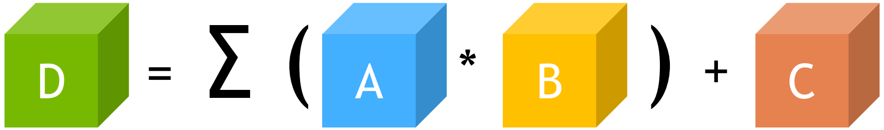
TAL-SH: https://github.com/DmitryLyakh/TAL_SH
TensorLy: http://tensorly.org

Itensor: http://itensor.org
Julia: https://github.com/Jutho/TensorOperations.jl & https://github.com/JuliaGPU/CUDAnative.jl

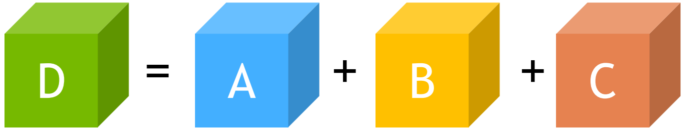
CUTENSOR

A High-Performance CUDA Library for Tensor Primitives

- Tensor Contractions (generalization of matrix-matrix multiplication)



- Element-wise operations (e.g., permutations, additions)

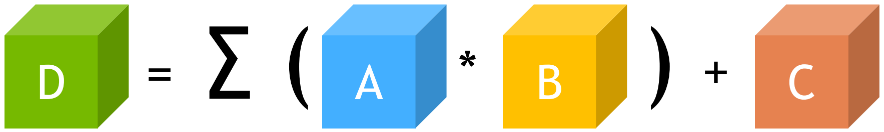


- Mixed precision support
- Generic and flexible interface

Tensor Contractions

TENSOR CONTRACTIONS

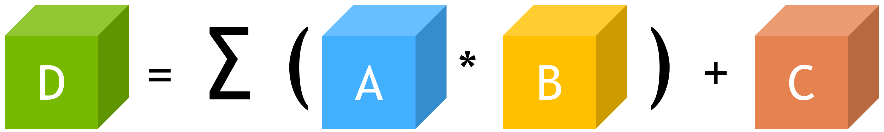
Examples



- Einstein notation (einsum)
  - Modes that appear in A and B are contracted
- Examples
  - $D_{m,n} = \alpha \sum_k A_{m,k} * B_{k,n}$ // GEMM

TENSOR CONTRACTIONS

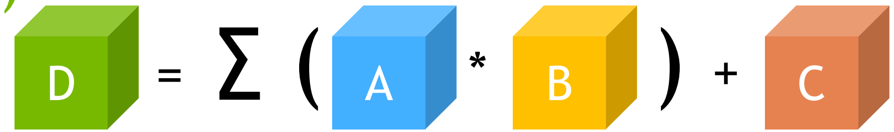
Examples



- Einstein notation (einsum)
  - Modes that appear in A and B are contracted
- Examples
  - $D_{m,n} = \alpha\, A_{m,k} * B_{k,n}$ // GEMM
  - $D_{m_1,n,m_2} = \alpha\, A_{m_1,k,m_2} * B_{k,n}$ // Tensor Contraction
  - $D_{m_1,n_1,n_2,m_2} = \alpha\, A_{m_1,k,m_2} * B_{k,n_2,n_1}$ // Tensor Contraction
  - $D_{m_1,n_1,n_2,m_2} = \alpha\, A_{m_1,k_1,m_2,k_2} * B_{k_2,k_1,n_2,n_1}$ // Multi-mode Tensor Contraction

TENSOR CONTRACTIONS

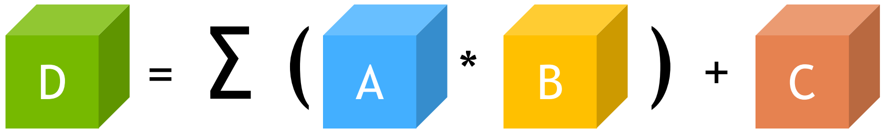
Examples (cont.)



- Examples
  - $D_{m,n} = \alpha\, A_m * B_n$ // outer product
  - $D_{m_1,n,m_2} = \alpha\, A_{m_1,m_2} * B_n$ // outer product
  - $D_{m_1,n_1,l_1} = \alpha\, A_{m_1,k,l_1} * B_{k,n_1,l_1}$ // batched GEMM
  - $D_{m_1,n_1,l_1,n_2,m_2} = \alpha\, A_{m_1,k,l_1,m_2} * B_{k,n_2,n_1,l_1}$ // single-mode batched tensor contraction
  - $D_{m_1,n_1,l_1,n_2,m_2,l_2} = \alpha\, A_{m_1,k,l_2,\, l_1,m_2} * B_{k,n_2,n_1,l_1,l_2}$ // multi-mode batched tensor contraction

TENSOR CONTRACTIONS

Key Features

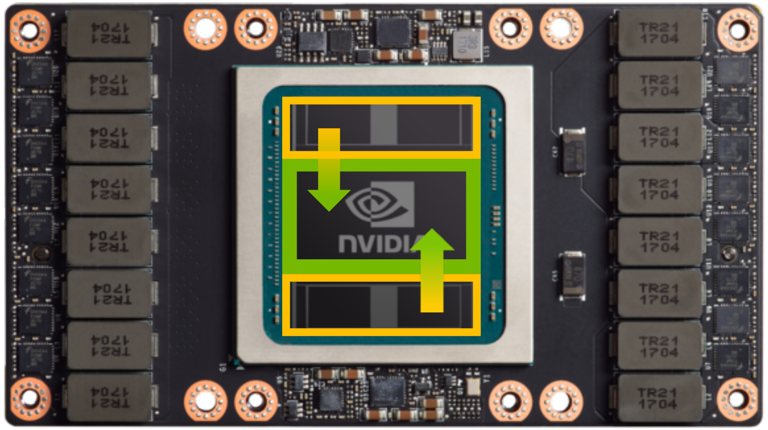


$$D_{\Pi^C(i_1,\ldots,i_n)} = \Psi_D(\alpha\Psi_A(A_{\Pi^A(i_1,\ldots,i_n)}) * \Psi_B(B_{\Pi^B(i_1,\ldots,i_n)})) + \beta\Psi_C(C_{\Pi^C(i_1,\ldots,i_n)}))$$

- Ψ are unary operators
  - E.g., Identity, RELU, CONJ, …
- Mixed-precision
- No additional work-space required
- Auto-tuning capability (similar to cublasGemmEx)
- High performance

TENSOR CONTRACTIONS

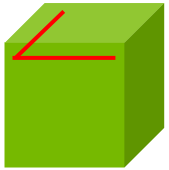
Key Challenges



- Keep the fast FPUs busy
  - Reuse data in shared memory & registers as much as possible
  - Coalesced accesses to/from global memory

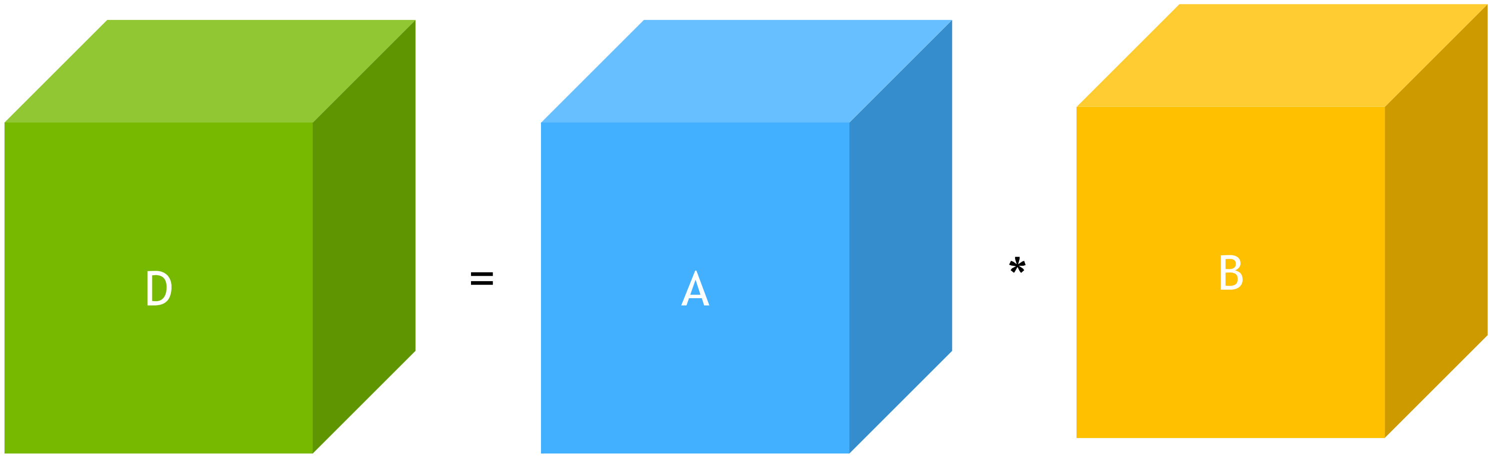
TENSOR CONTRACTIONS

Key Challenges

- Loading a scalar α ✅
- Loading a vector A_i ✅
- Loading a matrix $A_{i,j}$  (✅)
- Loading a general tensor $A_{i,j,k}$  ((✅))

TENSOR CONTRACTIONS

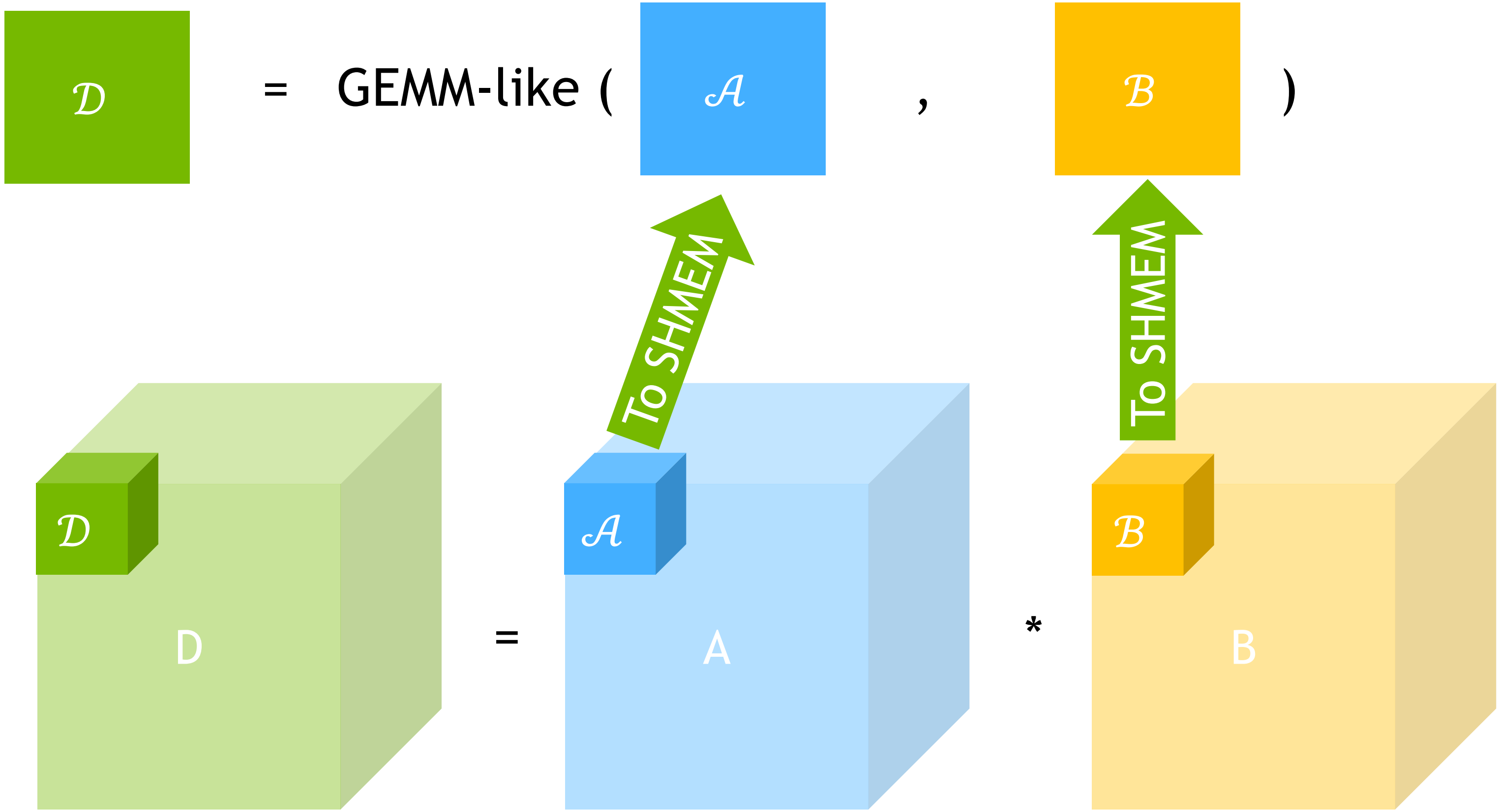
Technical insight



[1] Paul Springer and Paolo Bientinesi: “Design of a high-performance GEMM-like Tensor-Tensor Multiplication” (2016)

TENSOR CONTRACTIONS

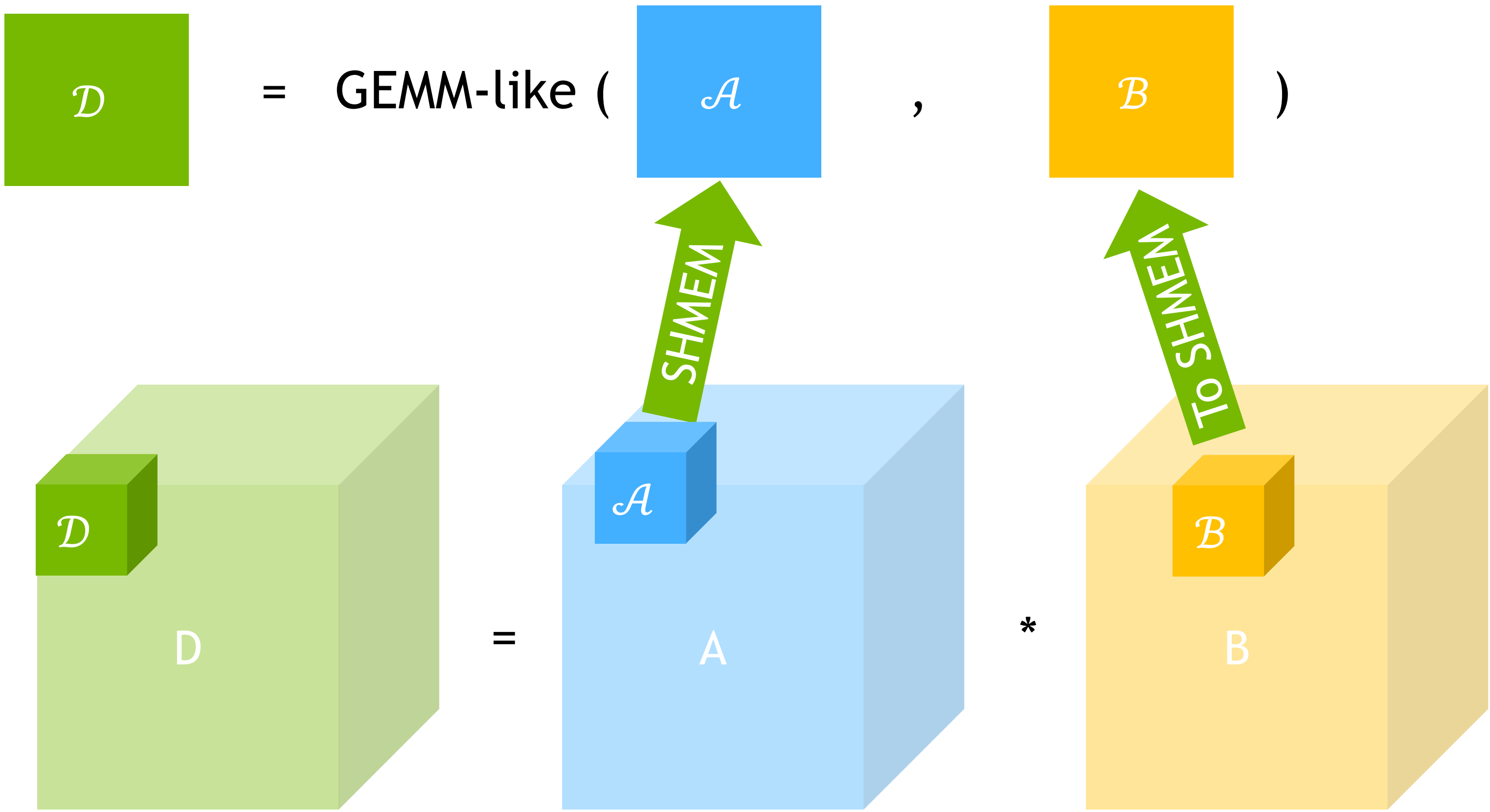
Technical insight



[1] Paul Springer and Paolo Bientinesi: “Design of a high-performance GEMM-like Tensor-Tensor Multiplication” (2016)

TENSOR CONTRACTIONS

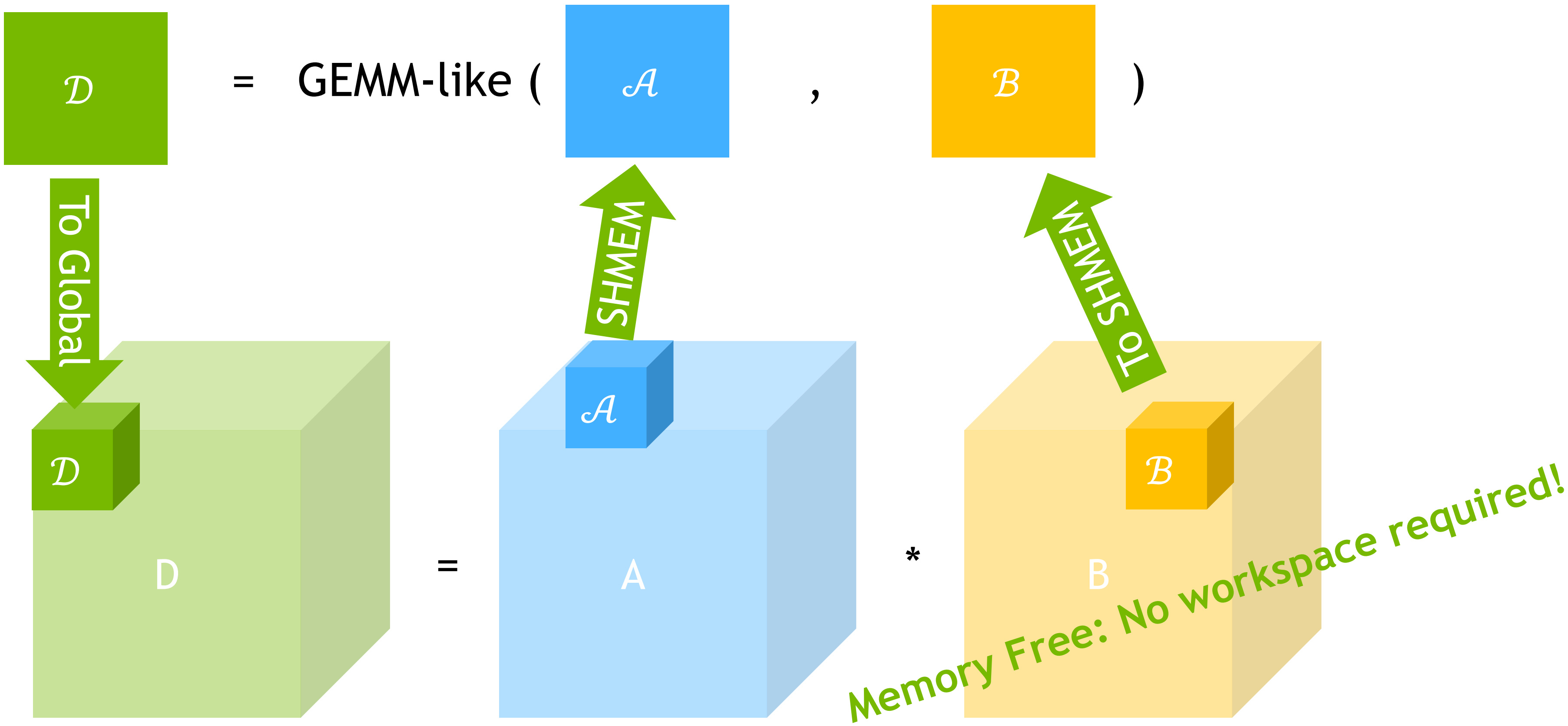
Technical insight



[1] Paul Springer and Paolo Bientinesi: "Design of a high-performance GEMM-like Tensor-Tensor Multiplication" (2016)

TENSOR CONTRACTIONS

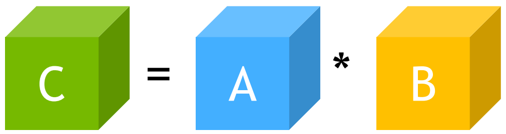
Technical insight

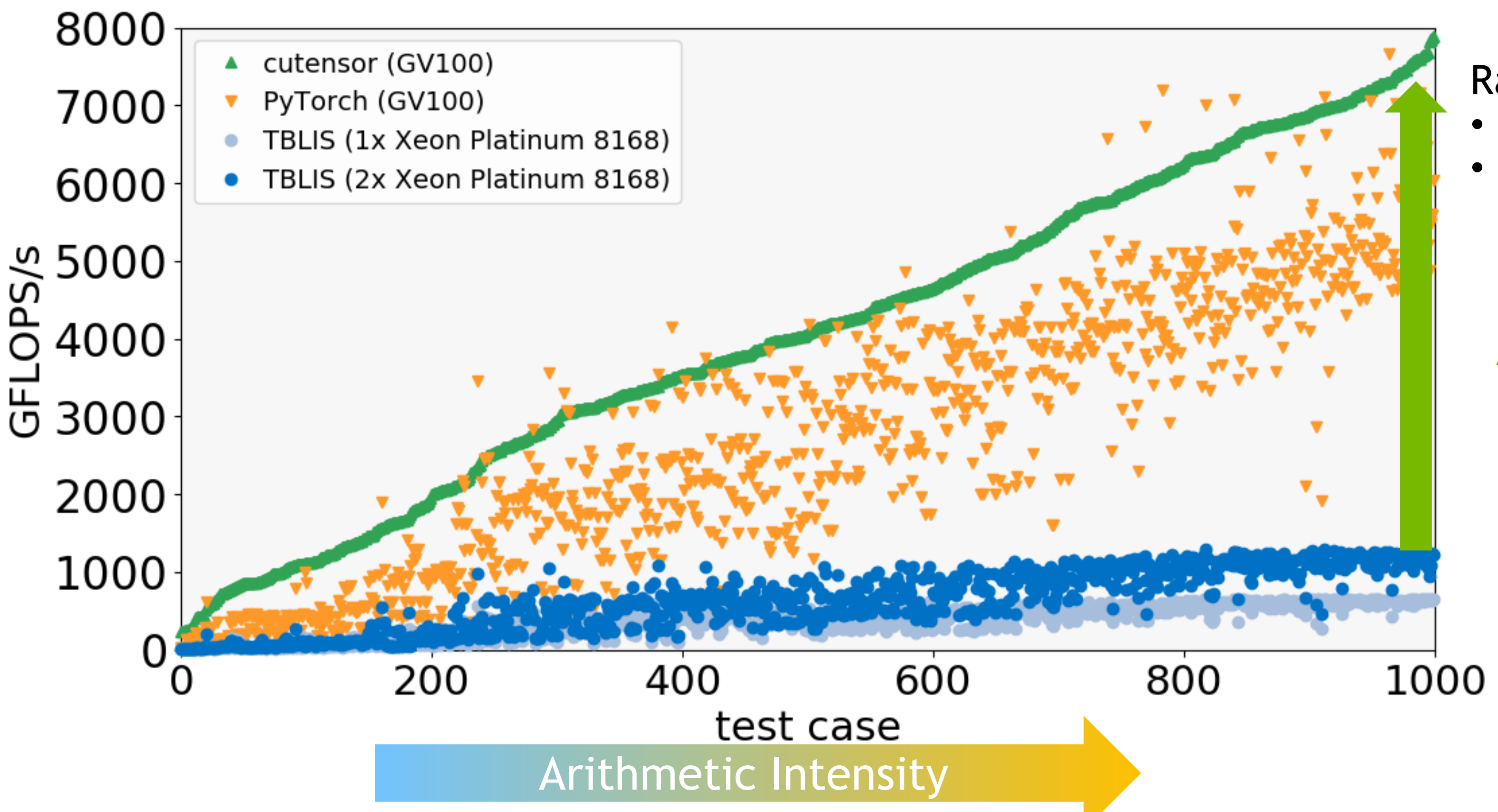


[1] Paul Springer and Paolo Bientinesi: “Design of a high-performance GEMM-like Tensor-Tensor Multiplication” (2016)

PERFORMANCE

Tensor Contractions





Random tensor contractions:

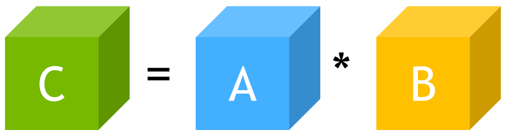
- 3D to 6D tensors
- FP64

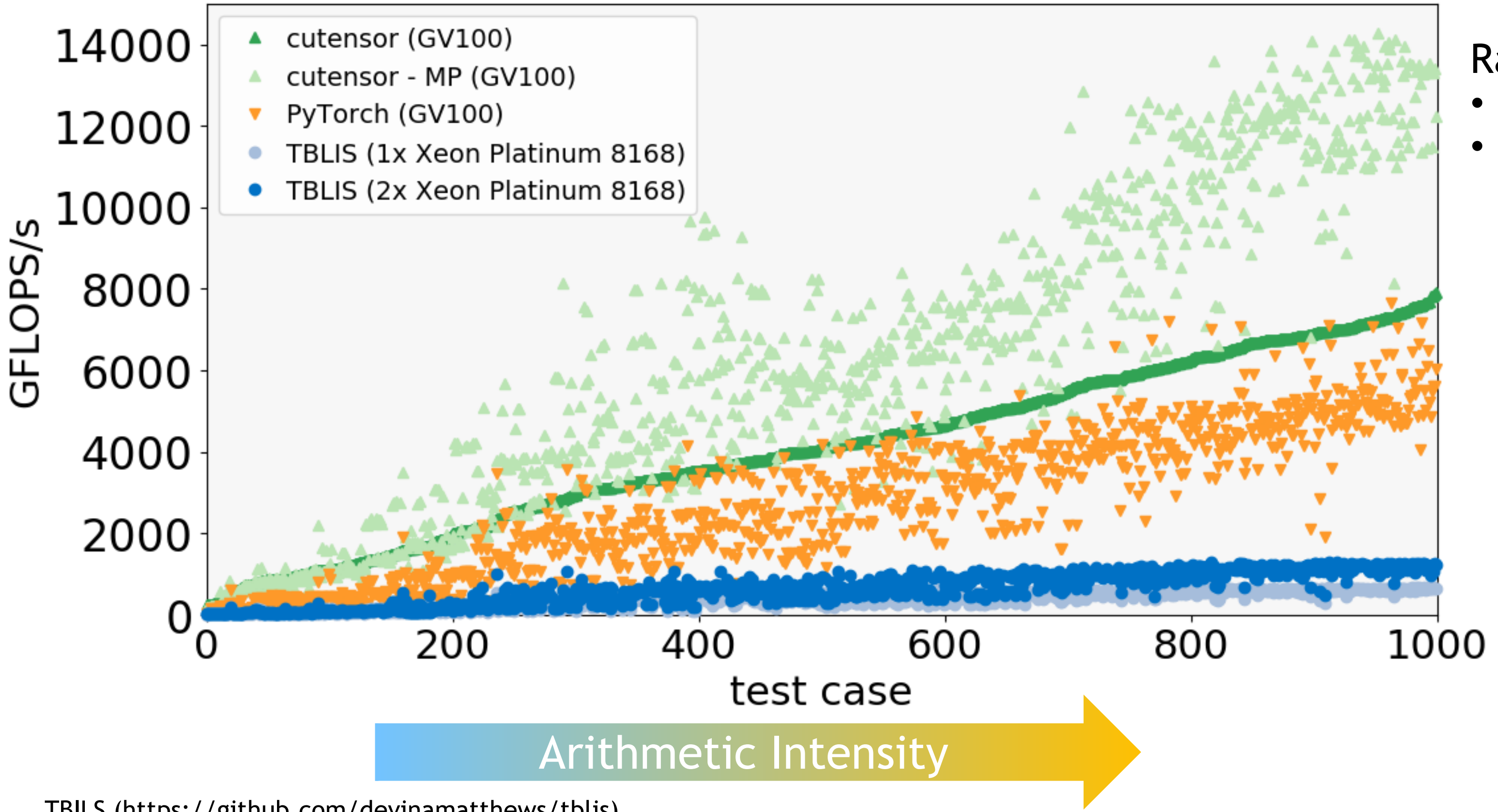
~8x over two-socket CPU

TBILS (https://github.com/devinamatthews/tblis)

PERFORMANCE

Tensor Contractions





Random tensor contractions:

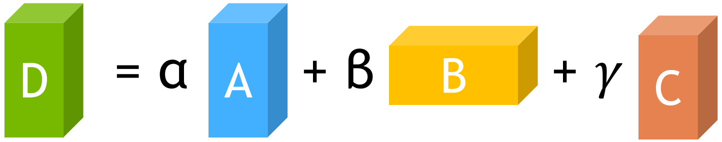
- 3D to 6D tensors
- FP64 (data) & FP32 (compute)

TBILS (https://github.com/devinamatthews/tblis)

Element-wise Operations

ELEMENT-WISE TENSOR OPERATIONS

Examples

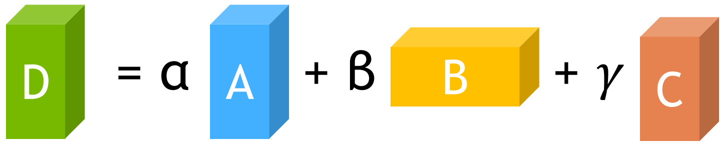


- $D_{w,h,c,n} = \alpha\, A_{c,w,h,n}$
- $D_{w,h,c,n} = \alpha\, A_{c,w,h,n} + \beta B_{c,w,h,n}$
- $D_{w,h,c,n} = \min(\alpha\, A_{c,w,h,n}\,,\ \beta\, B_{c,w,h,n})$
- $D_{w,h,c,n} = \alpha\, A_{c,w,h,n} + \beta\, B_{w,h,c,n} + \gamma\, C_{w,h,c,n}$
- $D_{w,h,c,n} = \alpha\, RELU(A_{c,w,h,n}) + \beta\, B_{w,h,c,n} + \gamma\, C_{w,h,c,n}$
- $D_{w,h,c,n} = FP32(\alpha\, RELU(A_{c,w,h,n}) + \beta\, B_{w,h,c,n} + \gamma\, C_{w,h,c,n})$

Enables users to fuse multiple element-wise calls.

ELEMENT-WISE TENSOR OPERATIONS

Key Features

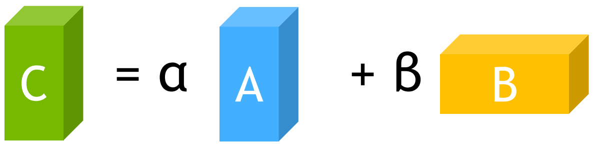


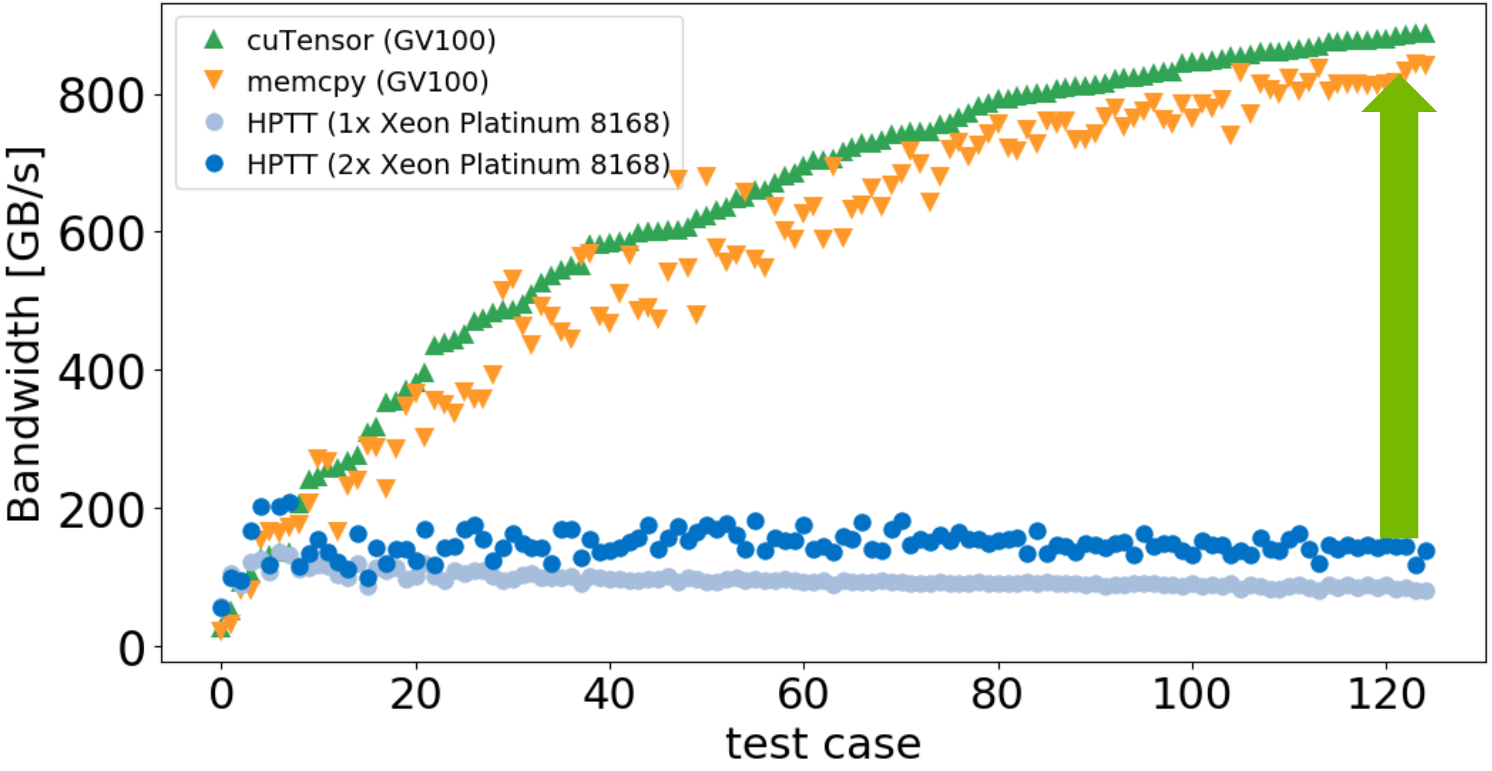
$$D_{\Pi^C(i_0,i_1,\ldots,i_n)} = \Phi_{ABC}(\Phi_{AB}(\alpha\,\Psi_A(A_{\Pi^A(i_0,i_1,\ldots,i_n)}),\ \beta\,\Psi_B(B_{\Pi^B(i_0,i_1,\ldots,i_n)})),\ \gamma\,\Psi_C(C_{\Pi^C(i_0,i_1,\ldots,i_n)}))$$

- Ψ are unary operators
  - E.g., Identity, RELU, CONJ, …
- Φ are binary operators
  - E.g., MAX, MIN, ADD, MUL, …
- Mixed-precision
- High performance

PERFORMANCE

Element-wise Operation





~5x over two-socket CPU

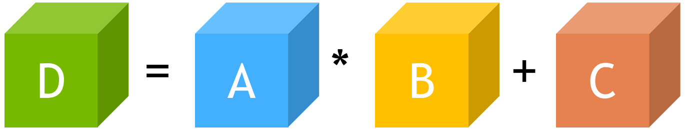
HPTT (https://github.com/springer13/hptt)

* FP32 tensor permutation (e.g., reformatting)

CUTENSOR's API

TENSOR CONTRACTIONS

API

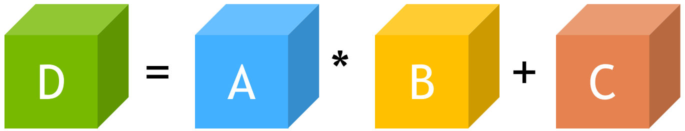



```
cutensorStatus_t cutensorContraction ( cuTensorHandle_t handle,
       const void *alpha, const void *A, const cutensorTensorDescriptor_t descA, const int modeA[],
                          const void *B, const cutensorTensorDescriptor_t descB, const int modeB[],
       const void *beta,  const void *C, const cutensorTensorDescriptor_t descC, const int modeC[],
                                void *D, const cutensorTensorDescriptor_t descD, const int modeD[],
       cutensorOperator_t opOut, cudaDataType_t typeCompute, cutensorAlgo_t algo,
       void *workspace, uint64_t workspaceSize,    // Workspace is optional and may be null
       cudaStream_t stream );
```

Devin Matthews et al. “Tensor interfaces”: https://github.com/MolSSI/tensor-interfaces/blob/master/interface.md

TENSOR CONTRACTIONS

API




```
cutensorStatus_t cutensorContraction ( cuTensorHandle_t handle,
       const void *alpha, const void *A, const cutensorTensorDescriptor_t descA, const int modeA[],
                          const void *B, const cutensorTensorDescriptor_t descB, const int modeB[],
       const void *beta,  const void *C, const cutensorTensorDescriptor_t descC, const int modeC[],
                                void *D, const cutensorTensorDescriptor_t descD, const int modeD[],
       cutensorOperator_t opOut, cudaDataType_t typeCompute, cutensorAlgo_t algo,
       void *workspace, uint64_t workspaceSize,    // Workspace is optional and may be null
       cudaStream_t stream );
```

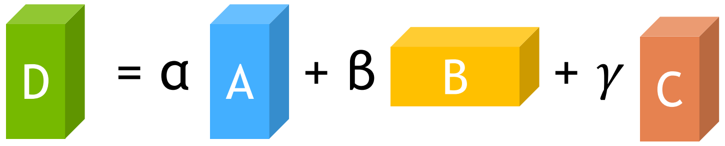
- $D_{a,b,m,n,c} = \alpha \sum_{o,p}(A_{a,o,p,b,c} * B_{o,m,p,n}) + \beta C_{a,b,m,n,c}$

```
auto status = cutensorContraction (handle,
                            alpha, A, descA, { ‘a’, ‘o’, ‘p’, ‘b’, ‘c’ },
                                   B, descB, { ‘o’, ‘m’, ‘p’, ‘n’ },
                            beta,  C, descC, { ‘a’, ‘b’, ‘m’, ‘n’, ‘c’ },
                                   D, descC, { ‘a’, ‘b’, ‘m’, ‘n’, ‘c’ },
                            CUTENSOR_OP_IDENTITY, CUDA_R_32F, CUTENSOR_ALGO_DEFAULT,
                            nullptr, 0, stream );
```

Devin Matthews et al. “Tensor interfaces”: https://github.com/MolSSI/tensor-interfaces/blob/master/interface.md

ELEMENT-WISE OPERATION

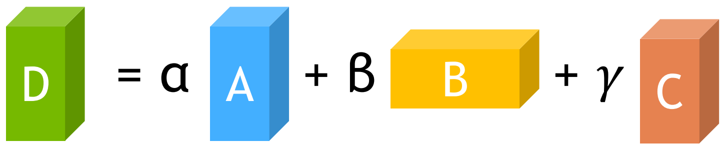
API




```
cutensorStatus_t cutensorElementwiseTrinary ( cuTensorHandle_t handle,
       const void *alpha, const void *A, const cutensorTensorDescriptor_t descA, const int modeA[],
       const void *beta,  const void *B, const cutensorTensorDescriptor_t descB, const int modeB[],
       const void *gamma, const void *C, const cutensorTensorDescriptor_t descC, const int modeC[],
                                void *D, const cutensorTensorDescriptor_t descD, const int modeD[],
       cutensorOperator_t opAB, cutensorOperator_t opABC, cudaDataType_t typeCompute,
       cudaStream_t stream );
```

ELEMENT-WISE OPERATION

API




```
cutensorStatus_t cutensorElementwiseTrinary ( cuTensorHandle_t handle,
       const void *alpha, const void *A, const cutensorTensorDescriptor_t descA, const int modeA[],
       const void *beta,  const void *B, const cutensorTensorDescriptor_t descB, const int modeB[],
       const void *gamma, const void *C, const cutensorTensorDescriptor_t descC, const int modeC[],
                               void *D, const cutensorTensorDescriptor_t descD, const int modeD[],
       cutensorOperator_t opAB, cutensorOperator_t opABC, cudaDataType_t typeCompute,
       cudaStream_t stream );
```

- $D_{w,h,c,n} = \min(\alpha A_{c,w,h,n}, \beta B_{c,w,h}) + \gamma C_{w,h,c,n}$

```
auto status = cutensorElementwiseTrinary ( handle,
                    alpha, A, descA, { ‘c’, ‘w’, ‘h’, ‘n’ },
                    beta,  B, descB, { ‘c’, ‘w’, ‘h’ },
                    gamma, C, descC, { ‘w’, ‘h’, ‘c’, ‘n’ },
                           D, descD, { ‘w’, ‘h’, ‘c’, ‘n’ },
                    CUTENSOR_OP_MIN, CUTENSOR_OP_ADD, CUDA_R_16F,
                    stream );
```

REFERENCES

[1] Devin A. Matthews “High-performance tensor contraction without Transposition” (2016)

[2] Paul Springer et al. “Design of a high-performance GEMM-like Tensor-Tensor Multiplication” (2016)

[3] Yang Shi et al. “Tensor Contractions with Extended BLAS Kernels on CPU and GPU” (2016)

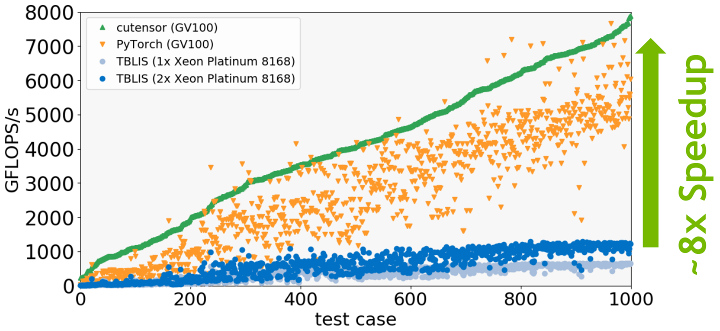
[4] Antti-Pekka Hynninen et al. “cuTT: A High-Performance Tensor Transpose Library for CUDA Compatible GPUs” (2017)

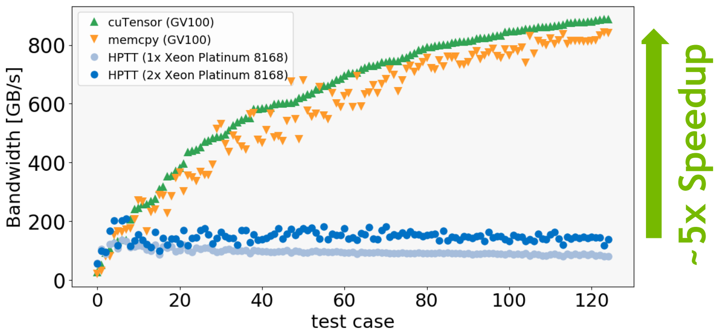
[5] Jinsung Kim et al. "Optimizing Tensor Contractions in CCSD(T) for Efficient Execution on GPUs." (2018).

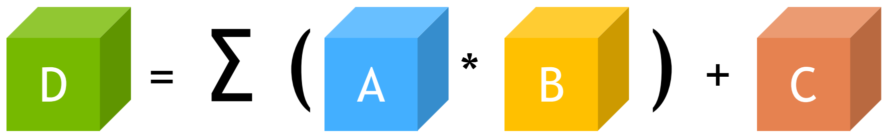
[6] Jinsung Kim et al. “A code generator for high-performance tensor contractions on GPUs” (2019)

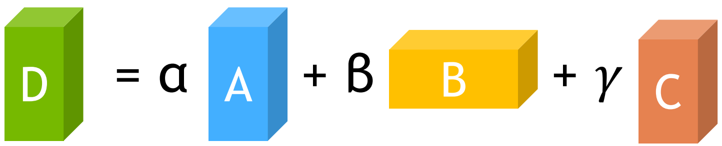
CUTENSOR

- CUDA library for high-performance CUDA tensor primitives









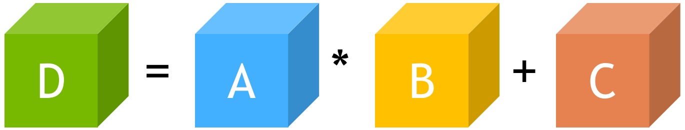
Pre-release available at:
https://developer.nvidia.com/cuTensor

Your feedback is highly appreciated.

NVIDIA

CUTENSOR

API




```
cutensorStatus_t cutensorCreateTensorDescriptor ( cutensorTensorDescriptor_t *desc,
                                                  unsigned int numModes,
                                                  const int64_t extent[],
                                                  const int64_t stride[],  // Stride is optional and may be null
                                                  cudaDataType_t dataType,
                                                  cutensorOperator_t unaryOp,
                                                  const int vectorIndex,
                                                  const int32_t vectorWidth);


cutensorStatus_t cutensorContraction (cuTensorHandle_t handle,
       const void* alpha, const void *A, const cutensorTensorDescriptor_t descA, const int modeA[],
                          const void *B, const cutensorTensorDescriptor_t descB, const int modeB[],
       const void* beta,  const void *C, const cutensorTensorDescriptor_t descC, const int modeC[],
                                void *D, const cutensorTensorDescriptor_t descD, const int modeD[],
       cutensorOperator_t opOut, cudaDataType_t typeCompute, cutensorAlgo_t algo,
       void* workspace, size_t workspaceSize,    // Workspace is optional and may be null
       cudaStream_t stream );
```

Devin Matthews et al. “Tensor interfaces”: https://github.com/MolSSI/tensor-interfaces/blob/master/interface.md