CUTENSOR

A CUDA Library for High-Performance Tensor Primitives

Paul Springer, November 20th 2019

pspringer@nvidia.com

CONTRIBUTERS

Markus Höhnerbach

Andrew Kerr

Chenhan Yu

Timothy Costa

Miguel Ferrer Avila

Manish Gupta

Alex Fit-Florea

WHAT IS A TENSOR?

- mode-0: scalar α

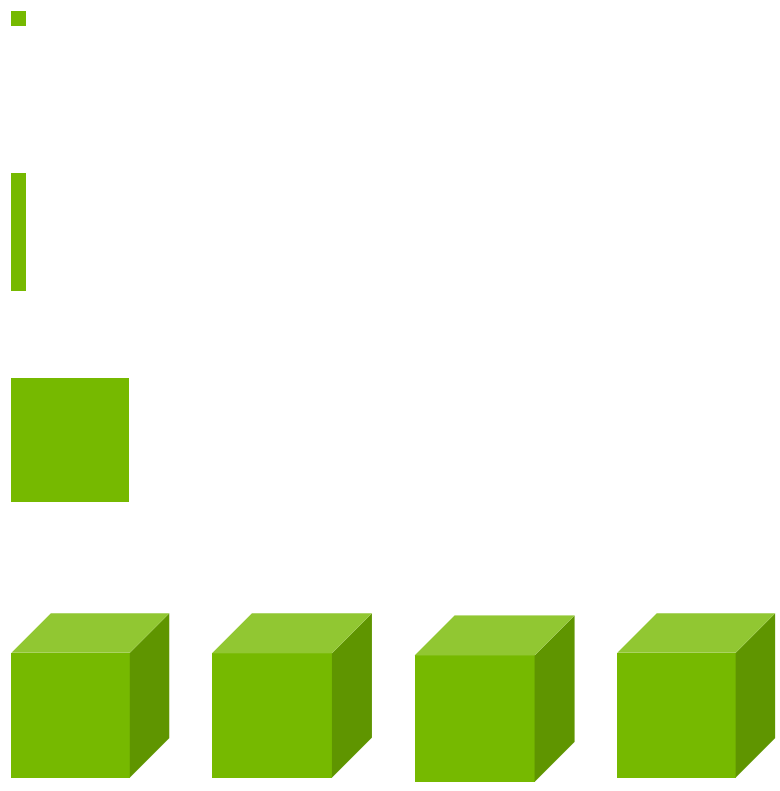
- mode-1: vector A_i

- mode-2: matrix $A_{i,j}$

- mode-n: general tensor $A_{i,j,k}$

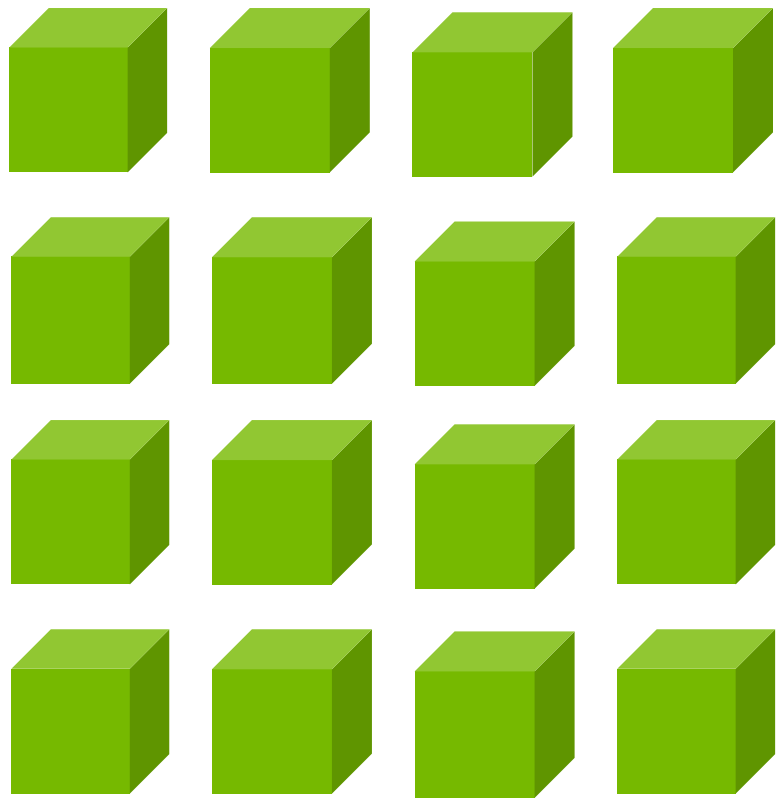
WHAT IS A TENSOR?

- 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?

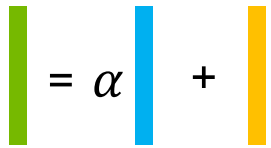
- mode-0: scalar α
- mode-1: vector A_i
- mode-2: matrix $A_{i,j}$
- mode-n: general tensor $A_{i,j,k,l,m}$



BASIC LINEAR SUBPROGRAMS

A Success Story

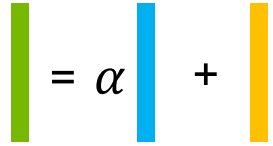
- 1969 – BLAS Level 1: Vector-Vector



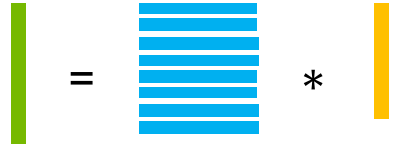
BASIC LINEAR SUBPROGRAMS

A Success Story

- 1969 – BLAS Level 1: Vector-Vector



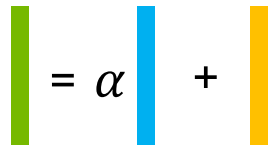
- 1972 – BLAS Level 2: Matrix-Vector



BASIC LINEAR SUBPROGRAMS

A Success Story

- 1969 – BLAS Level 1: Vector-Vector

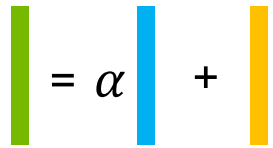


- 1972 – BLAS Level 2: Matrix-Vector

BASIC LINEAR SUBPROGRAMS

A Success Story

- 1969 – BLAS Level 1: Vector-Vector



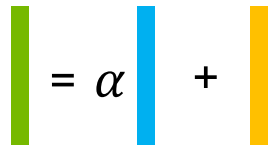
- 1972 – BLAS Level 2: Matrix-Vector

- 1980 – BLAS Level 3: Matrix-Matrix

BASIC LINEAR SUBPROGRAMS

A Success Story

- 1969 – BLAS Level 1: Vector-Vector



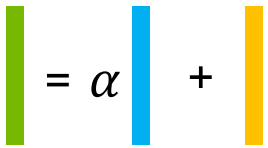
- 1972 – BLAS Level 2: Matrix-Vector

- 1980 – BLAS Level 3: Matrix-Matrix

BASIC LINEAR SUBPROGRAMS

A Success Story

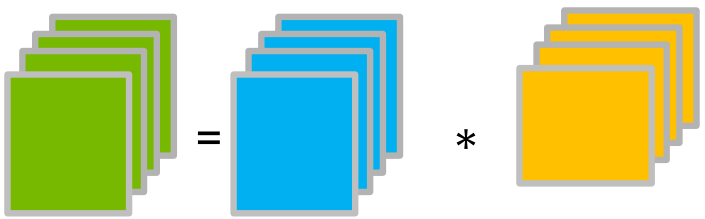
- 1969 – BLAS Level 1: Vector-Vector



- 1972 – BLAS Level 2: Matrix-Vector

- 1980 – BLAS Level 3: Matrix-Matrix

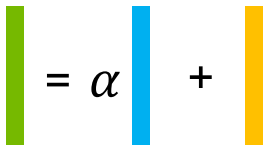
- Now? – BLAS Level 4: Tensor-Tensor



BASIC LINEAR SUBPROGRAMS

A Success Story

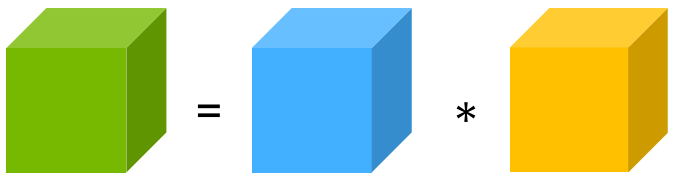
- 1969 – BLAS Level 1: Vector-Vector



- 1972 – BLAS Level 2: Matrix-Vector

- 1980 – BLAS Level 3: Matrix-Matrix

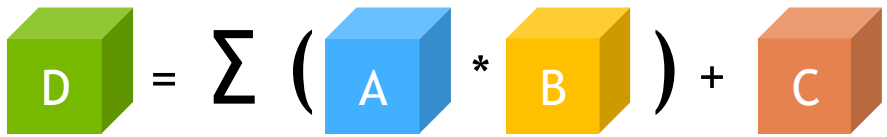
- Now? – BLAS Level 4: Tensor-Tensor



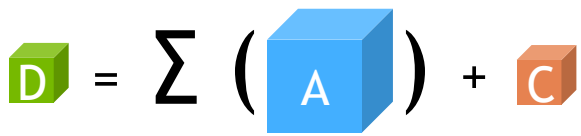
CUTENSOR

A High-Performance CUDA Library for Tensor Primitives

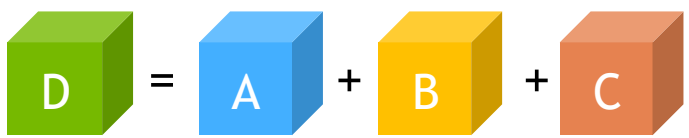
- Tensor contractions (generalization of matrix-matrix multiplication)



- Tensor reductions

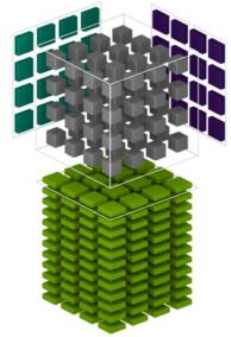


- Element-wise operations (e.g., permutations, additions)

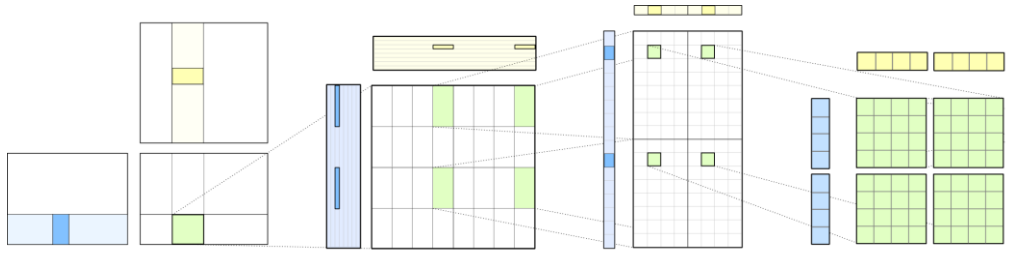


CUTENSOR

Key Features



- Transpose-free contractions
- Arbitrary data layouts
- Extensive mixed-precision support
  - Complex-times-Real
  - FP64 data + FP32 compute
  - FP32 data + FP16 compute
- Tensor-Core support
  - Built on CUTLASS 2.0
- Flexible and modern interface
  - No mallocs inside the lib
  - Tensor Cores active by default



TENSORS ARE UBIQUITOUS

Potential Use Cases: HPC & AI

PyTorch

Pyro

TensorFlow

Scikit-CUDA

NWCHEM
HIGH-PERFORMANCE COMPUTATIONAL CHEMISTRY SOFTWARE

TAL-SH

ITENSOR

julia

CuPy

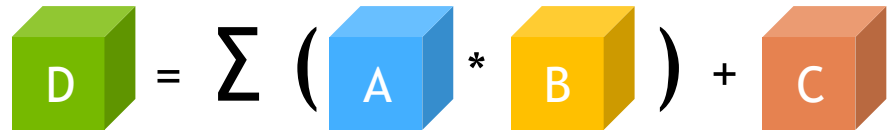
TAL-SH: https://github.com/DmitryLyakh/TAL_SH

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

Tensor Contractions

TENSOR CONTRACTIONS

Examples

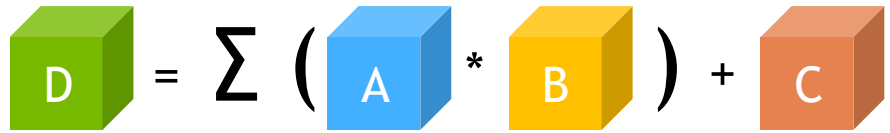


- Einstein notation (einsum)
  - Modes that appear in A and B are contracted
- Examples
  - $D_{m,n} = \alpha \sum_k A_{m,k} * B_{k,n}$ // GEMM

TF: https://www.tensorflow.org/api_docs/python/tf/einsum
pyTorch: https://pytorch.org/docs/stable/torch.html#torch.einsum

TENSOR CONTRACTIONS

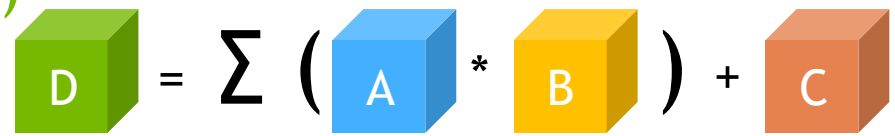
Examples



- 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
  - $D_{m_1,n_1,n_2,m_2} = \alpha\, A_{m_1,k_1,m_2,k_2} * B_{k_1,k_2,n_2,n_1}$ // Multi-mode Tensor Contraction

TENSOR CONTRACTIONS

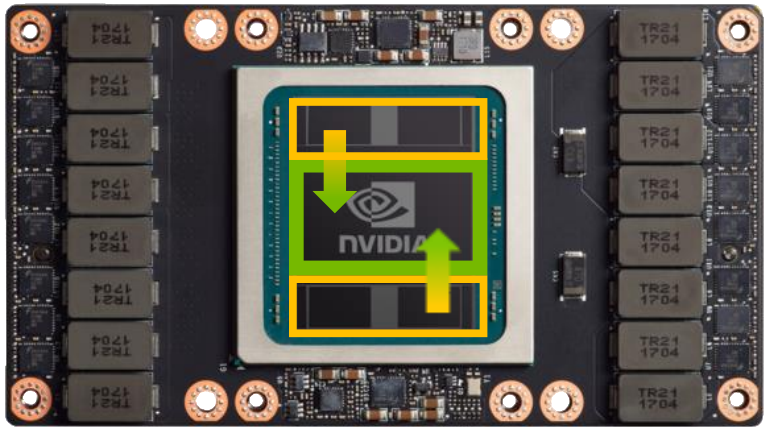
Examples (cont.)

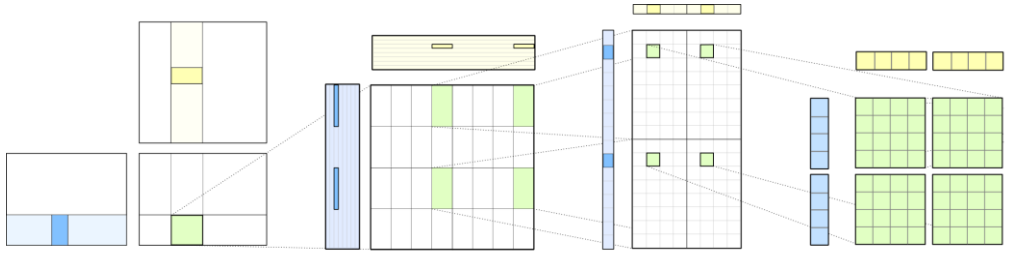


- Examples
  - $D_{m,n} = \alpha\, A_m * B_n$ // outer product
  - $D_{m_1,n,m_2} = \alpha\, A_{m_1,m_2} * B_n$ // outer product
  - $D_{m_1,m_2} = \alpha\, A_{m_1,k_1,m_2,k_2} * B_{k_1,k_2}$ // GEMV-like tensor contraction
  - $D_{m_1,n_1,l_1} = \alpha\, A_{m_1,k,l_1} * B_{k,n_1,l_1}$ // batched GEMM
  - $D_{m_1,n_1,l_1,n_2,m_2} = \alpha\, A_{m_1,k,l_1,m_2} * B_{k,n_2,n_1,l_1}$ // 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 Challenges



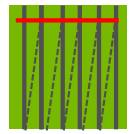
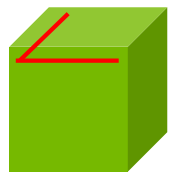


CUTLASS 2.0

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

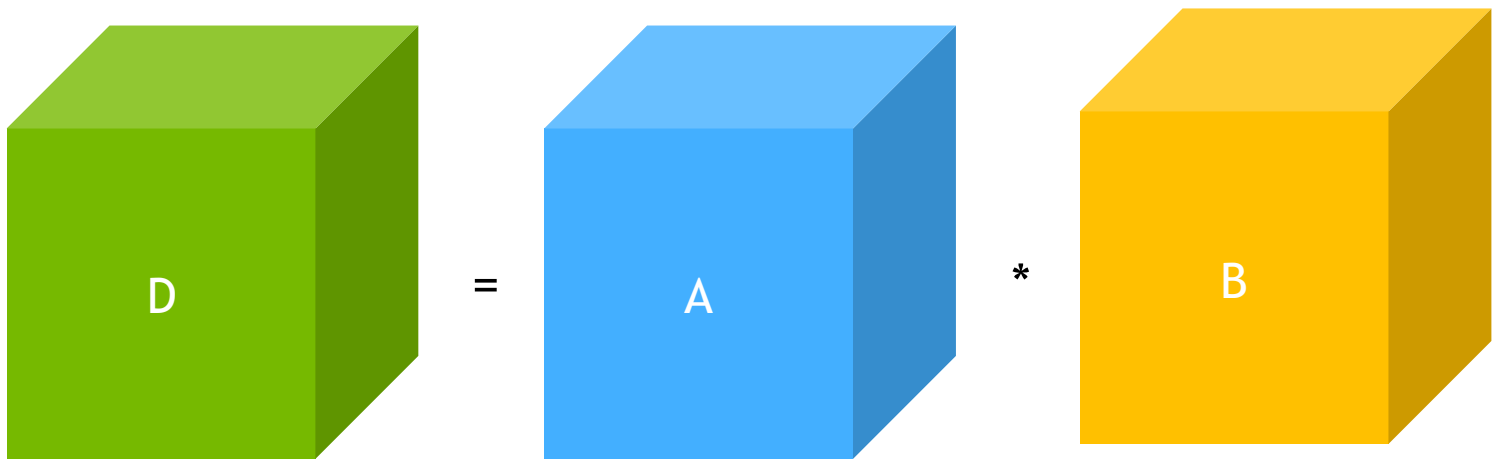
TENSOR CONTRACTIONS

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

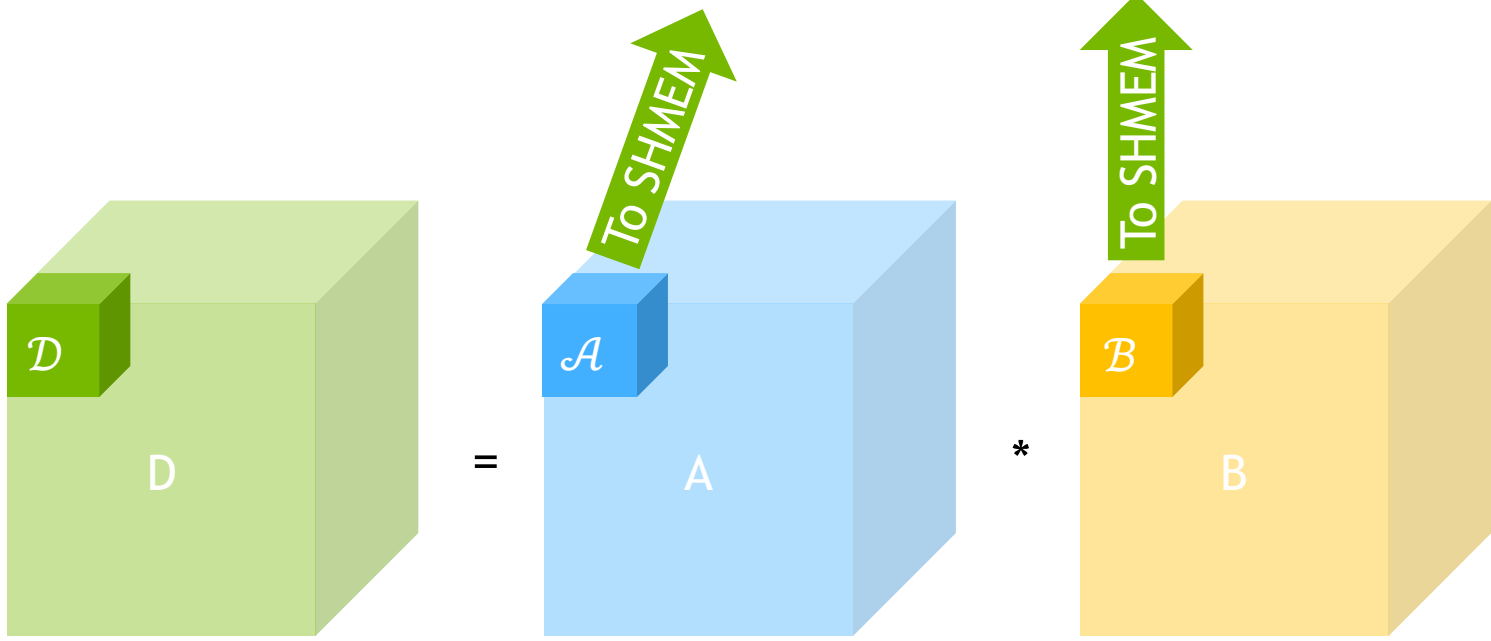
Technical insight



TENSOR CONTRACTIONS

Technical insight

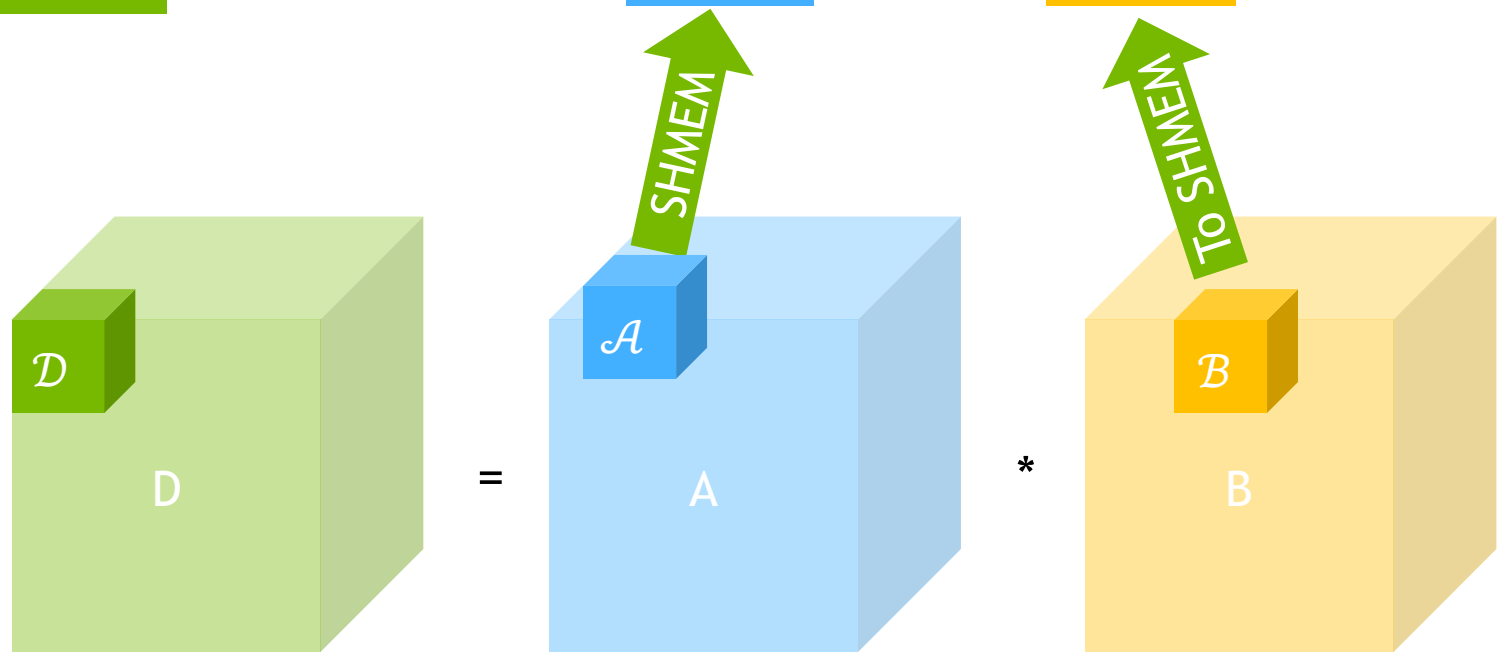
$\mathcal{D}$ = GEMM-like ($\mathcal{A}$, $\mathcal{B}$)



TENSOR CONTRACTIONS

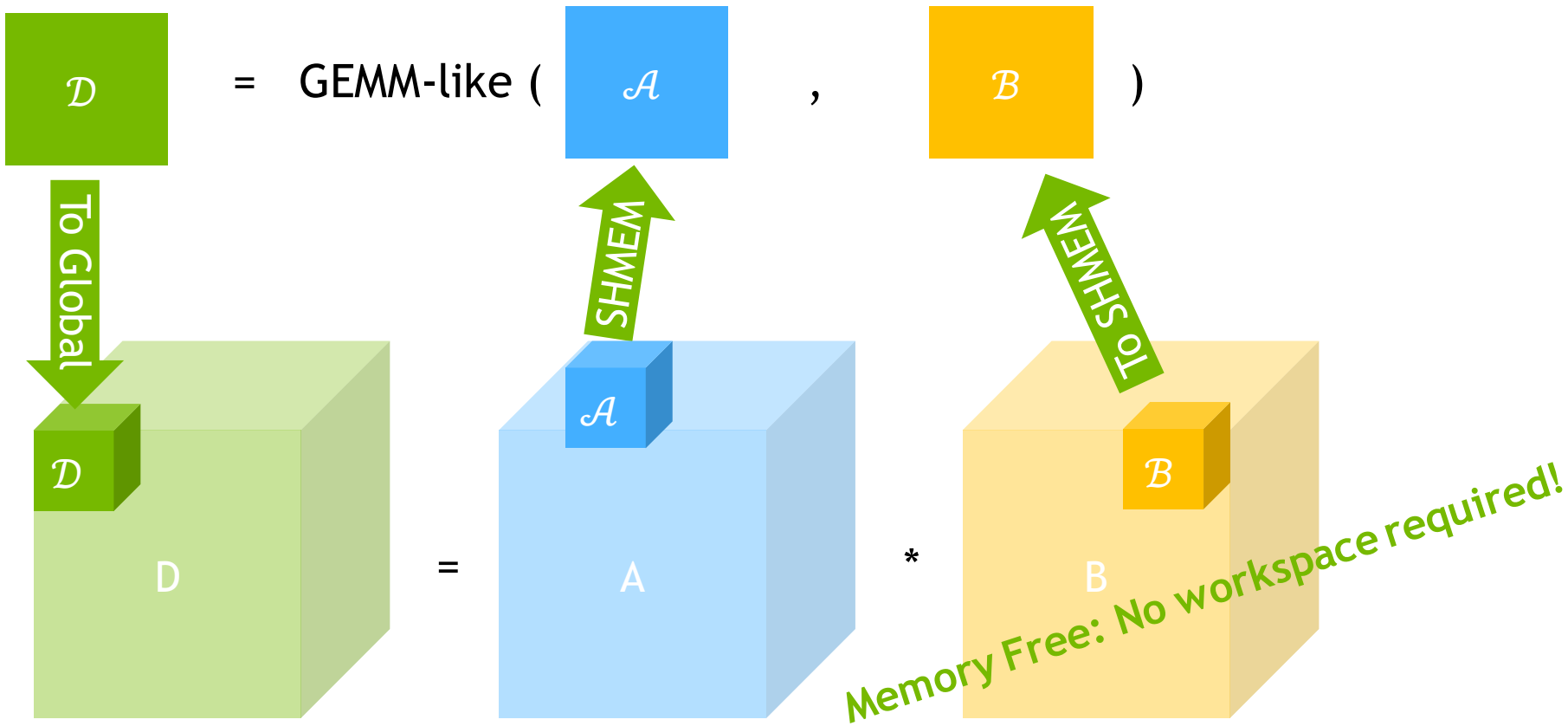
Technical insight

$\mathcal{D}$ = GEMM-like ($\mathcal{A}$, $\mathcal{B}$)



TENSOR CONTRACTIONS

Technical insight



TENSOR CONTRACTIONS

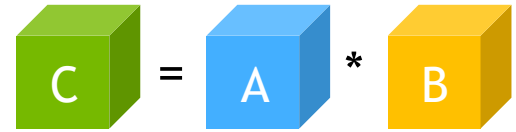
Performance Guidelines

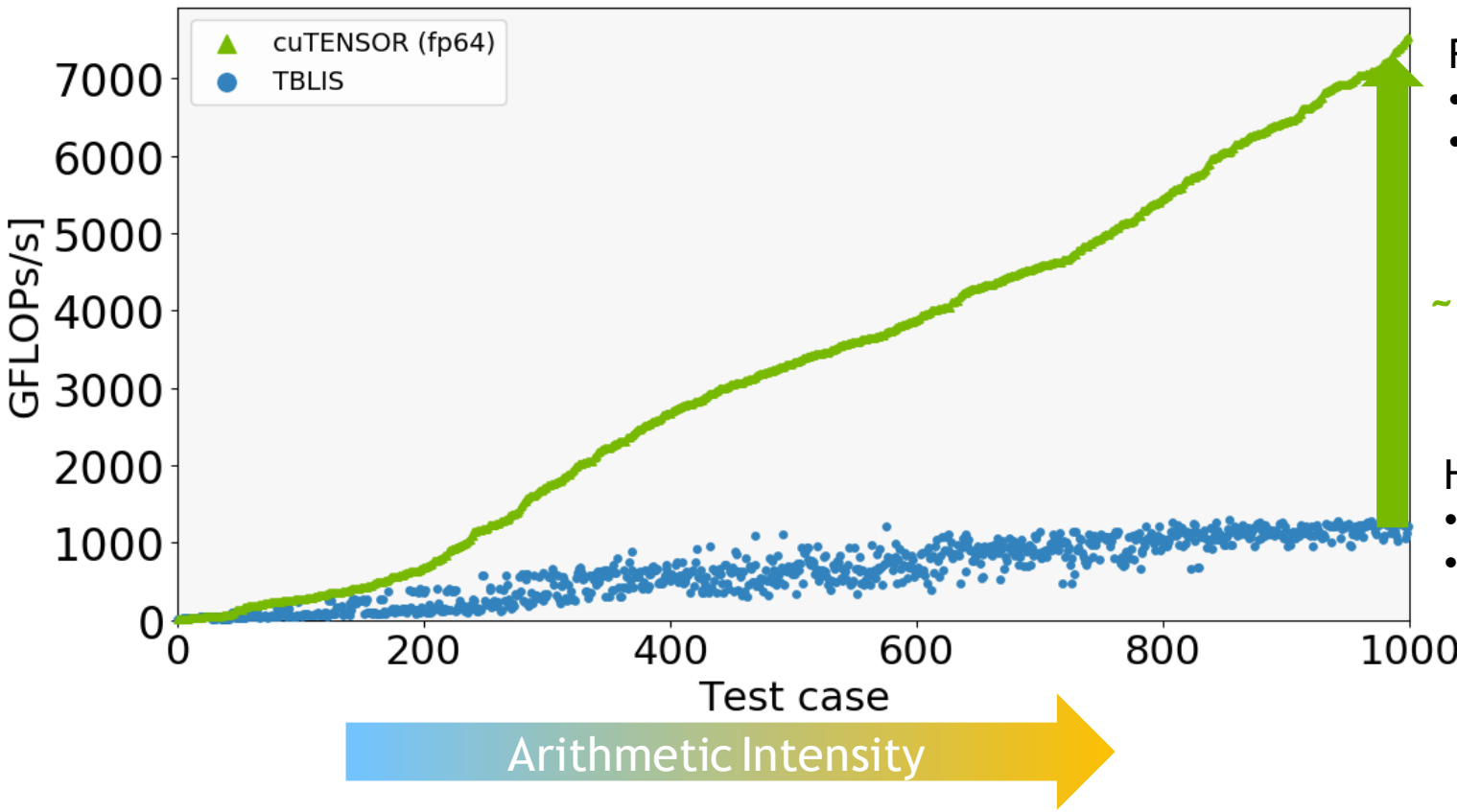
- Arrange modes similarly in all tensors
  - E.g., $C_{a,b,c} = A_{a,k,c} B_{k,b}$ is preferable to $C_{a,b,c} = A_{c,k,a} B_{k,b}$
- Keep the extent of the fastest-varying mode as large as possible
  - E.g., $C_{a,b,c} \in R^{1000 \times 100 \times 10}$ is preferable to $C_{c,b,a} \in R^{10 \times 100 \times 1000}$
- Keep batched modes as the slowest-varying modes (i.e., with the largest strides)
  - E.g., $C_{a,b,c,l} = A_{a,k,c,l} B_{k,b,l}$ is preferable to $C_{a,b,c,l} = A_{a,k,l,c} B_{l,k,b}$

*assuming generalized column-major layout

PERFORMANCE

Tensor Contractions





Random tensor contractions:

- 3D to 6D tensors
- FP64

~7x Speedup

Hardware:

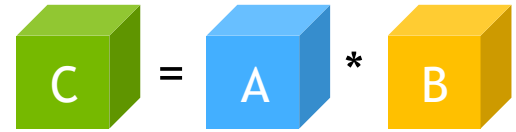
- Tesla GV100
- 2x Intel Xeon Platinum 8168

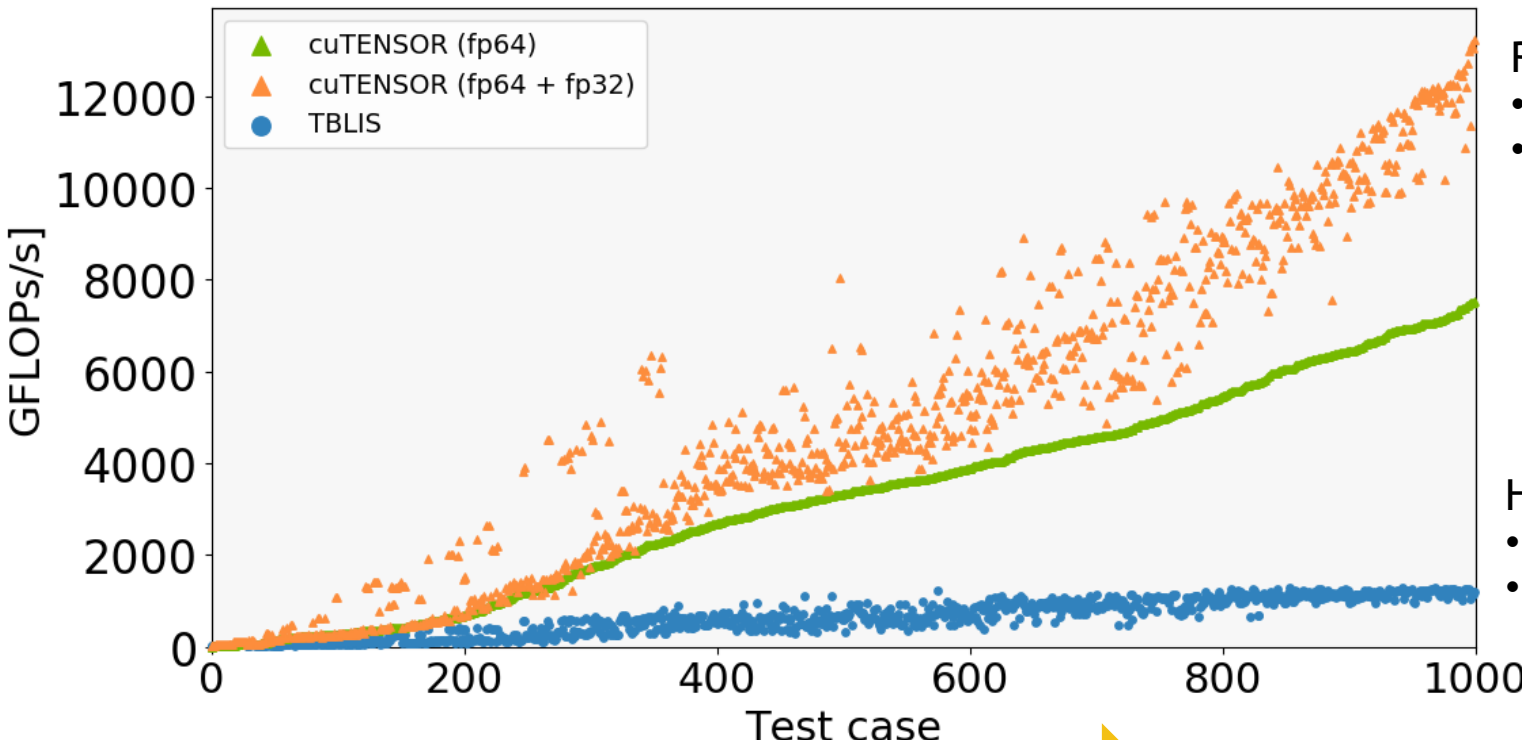
TBILS (https://github.com/devinamatthews/tblis)

Benchmark available at: http://tensornetwork.org/benchmarks/

PERFORMANCE

Tensor Contractions





Arithmetic Intensity

Random tensor contractions:

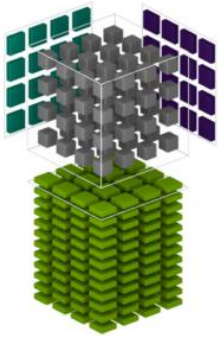
- 3D to 6D tensors
- FP64 (data) & FP32 (compute)

Hardware:

- Tesla GV100
- 2x Intel Xeon Platinum 8168

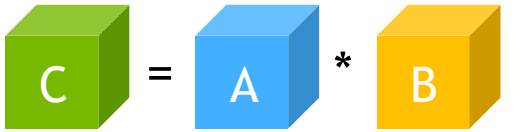
TBILS (https://github.com/devinamatthews/tblis)

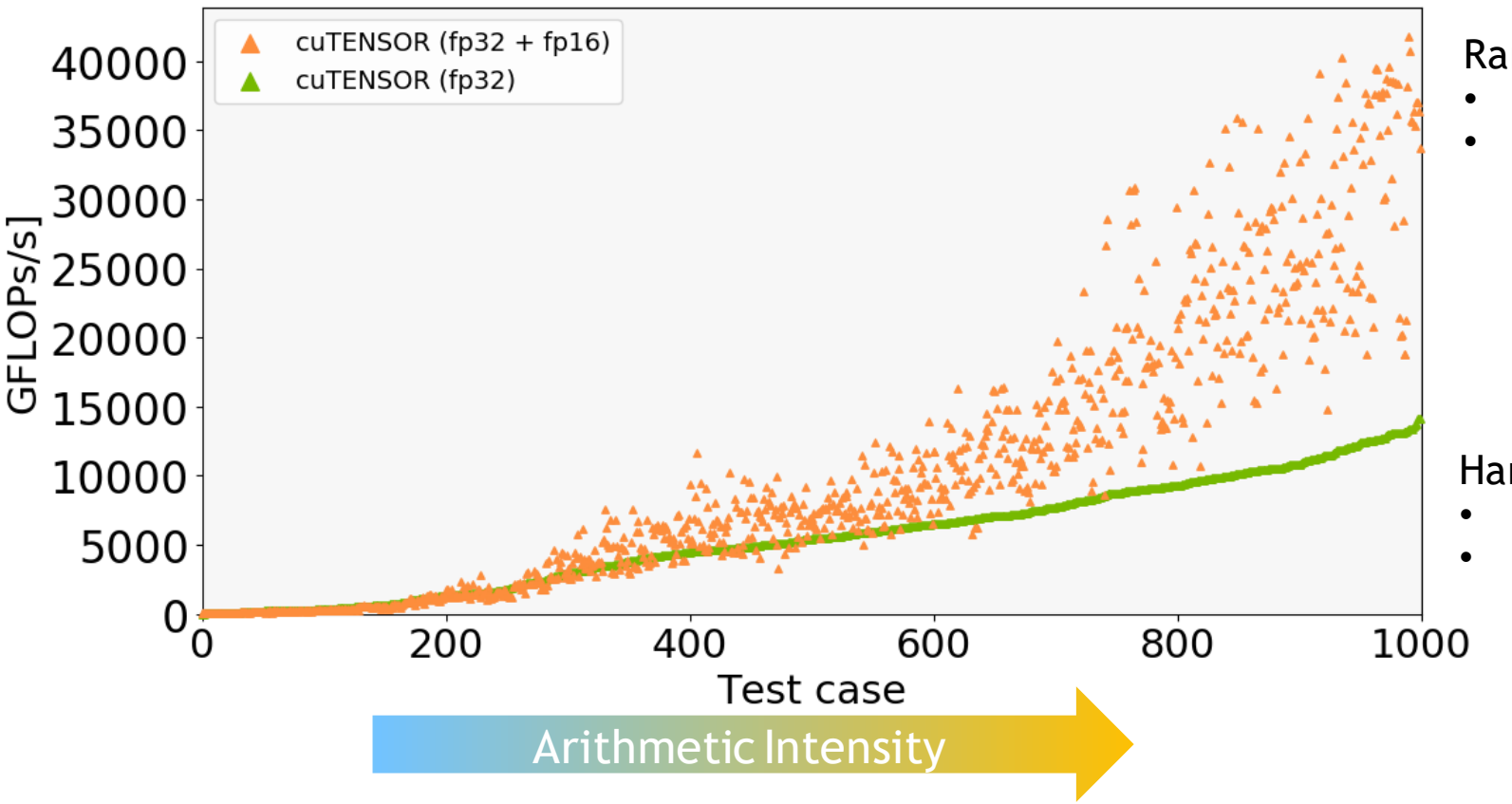
Benchmark available at: http://tensornetwork.org/benchmarks/



PERFORMANCE

Tensor Contractions





Random tensor contractions:

- 3D to 6D tensors
- FP32 (data) + Tensor Core
  - Accumulation in FP32
  - Inputs truncated to FP16

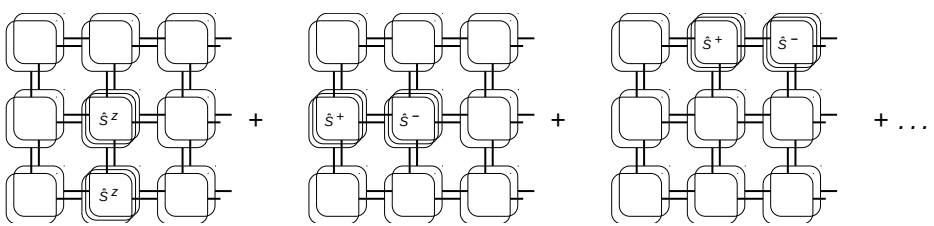
Hardware:

- Tesla GV100
- 2x Intel Xeon Platinum 8168

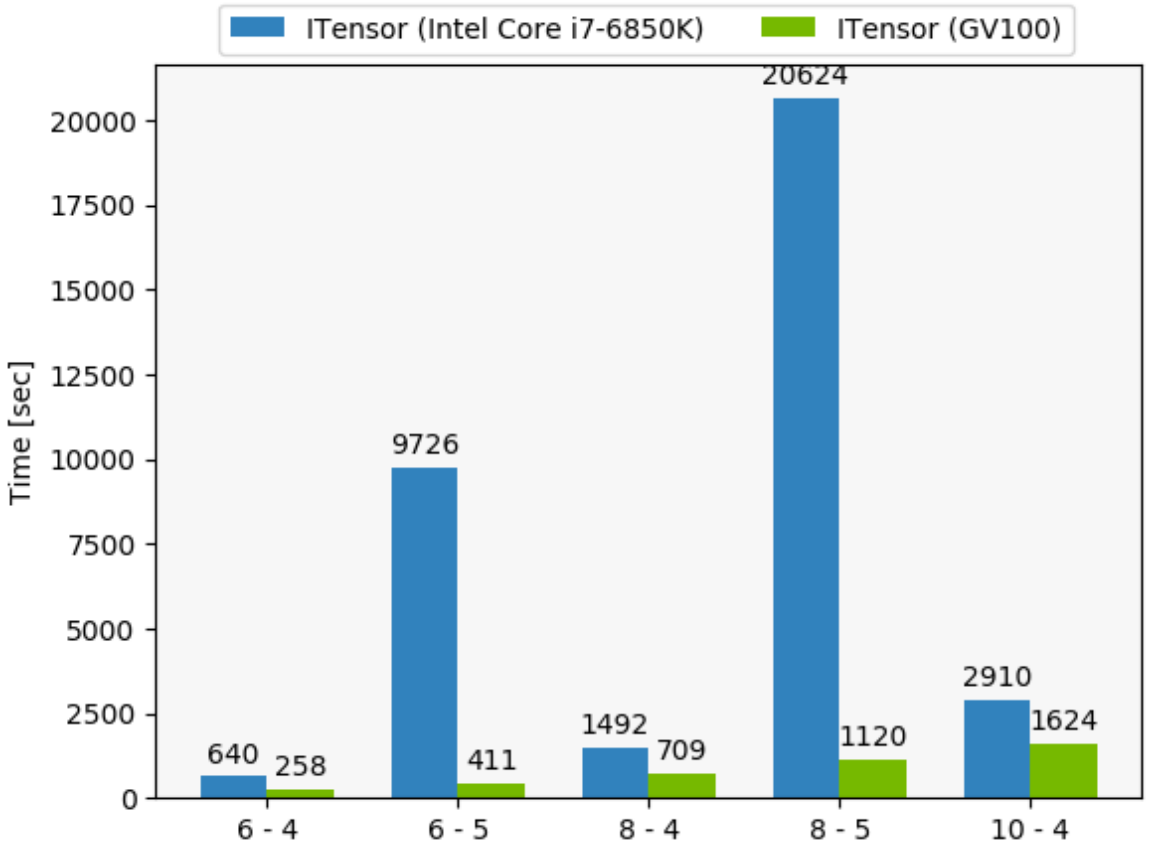
TBILS (https://github.com/devinamatthews/tblis)

Benchmark available at: http://tensornetwork.org/benchmarks/

ITENSOR – PEPS SIMULATION



Speedups: up to 23.6x



Credits:

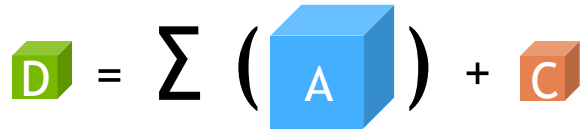
- Katharine Hyatt (Flatiron Institute) – https://github.com/itensor/ITensorsGPU.jl
- Miles Stoudenmire and Matt Fishman (Flatiron Instute) - https://github.com/ITensor/ITensors.jl
- Tim Besard (Ghent University) - https://github.com/JuliaGPU

*results are based on cuTENSOR's early-access version

Tensor Reductions

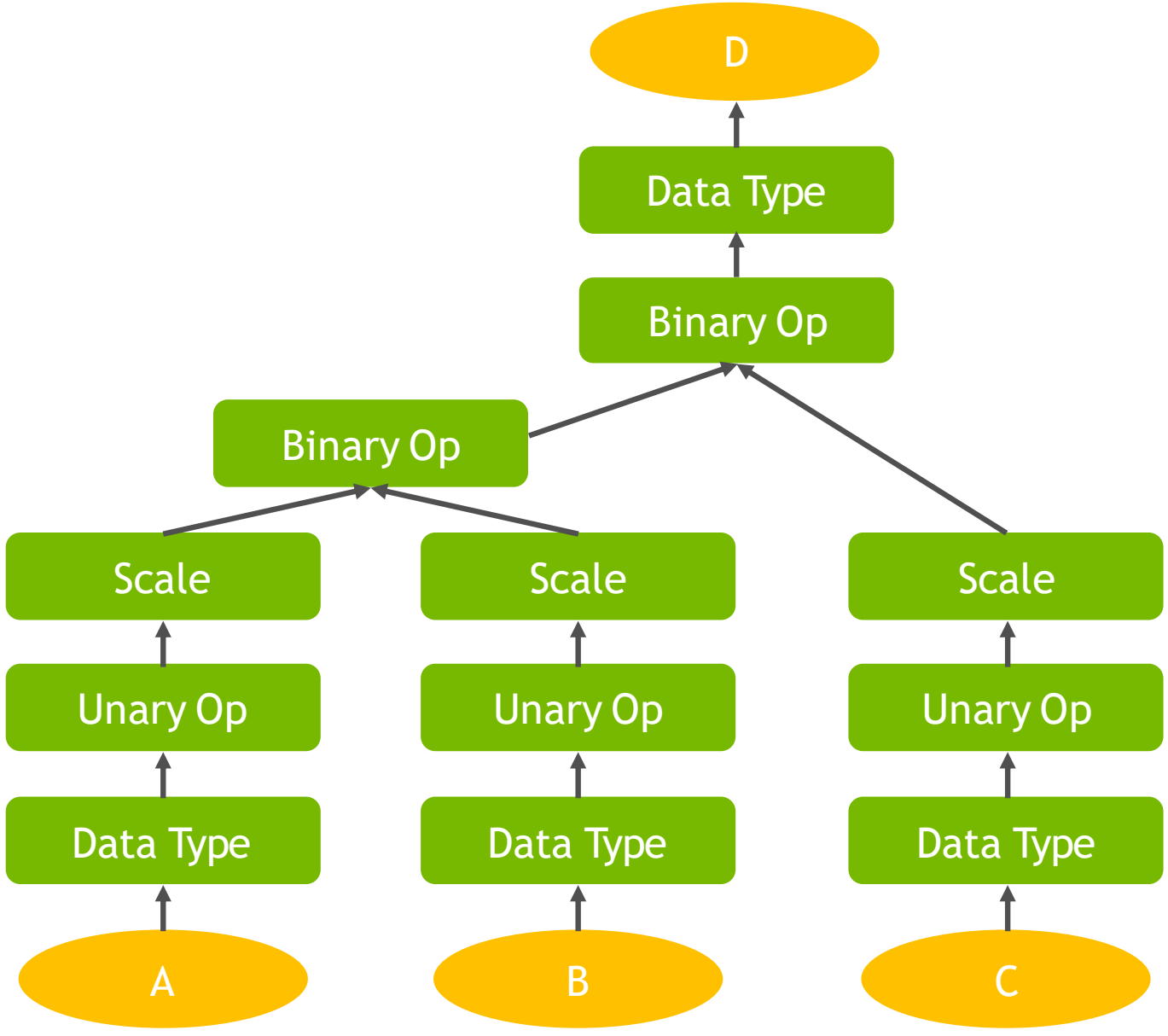
TENSOR REDUCTION

Examples



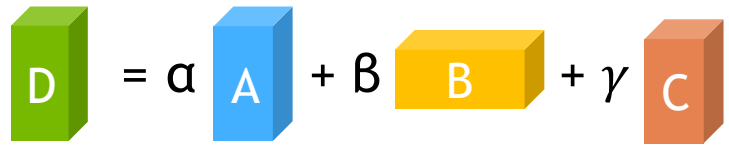
- $D_{w,h,n} = \alpha\ Reduce(A_{c,w,h,n}, ADD)$ // Reduction over mode c
- $D_{w,n} = \alpha\ Reduce(A_{c,w,h,n}, ADD)$ // Reduction over mode c and h
- $D_{w,n} = \alpha\ Reduce(RELU(A_{c,w,h,n}), ADD)$ // RELU + Reduction over mode c and h
- $D_{w,n} = \alpha\ Reduce(RELU(A_{c,w,h,n}), MAX)$ // RELU + Max-reduction over mode c and h

Element-wise Operations


D
Data Type
Binary Op
Binary Op
Scale
Scale
Scale
Unary Op
Unary Op
Unary Op
Data Type
Data Type
Data Type
A
B
C

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.

cuTENSOR’s API

CONTRACTION OPERATION

API


```
cutensorStatus_t cutensorInitContractionDescriptor ( const cutensorHandle_t *handle,
       cutensorContractionDescriptor_t *desc,
       const cutensorTensorDescriptor_t *descA, const int modeA[], const uint32_t alignmentRequirementA,
       const cutensorTensorDescriptor_t *descB, const int modeB[], const uint32_t alignmentRequirementB,
       const cutensorTensorDescriptor_t *descC, const int modeC[], const uint32_t alignmentRequirementC,
       const cutensorTensorDescriptor_t *descD, const int modeD[], const uint32_t alignmentRequirementD,
       cutensorComputeType_t typeCompute );

cutensorStatus_t cutensorInitContractionFind ( const cutensorHandle_t *handle,
       cutensorContractionFind *find, const cutensorAlgo_t algo );

cutensorStatus_t cutensorInitContractionPlan ( const cutensorHandle_t *handle,
       cutensorContractionPlan_t *plan, const cutensorContractionDescriptor_t *desc,
       const cutensorContractionFind *find, uint64_t workspaceSize );

 cutensorStatus_t cutensorContraction( const cutensorHandle_t *handle,
        const cutensorContractionPlan_toid *plan,
        const void *alpha, const void *A, const void *B,
        const void *beta,  const void *C,       void *D,
        void *workspace, uint64_t workspaceSize, cudaStream_t stream );
```

CONTRACTION OPERATION

API

- $D_{m,n,u} = \alpha A_{m,k,u} * B_{k,n}$

```
cutensorInitContractionDescriptor ( &handle, &desc,
                  descA, { 'm', 'k', 'u' }, 256,
                  descB, { 'k', 'n' },      256,
                  descC, { 'm', 'n', 'u' }, 256,
                  descD, { 'm', 'n', 'u' }, 256,
                  CUTENSOR_R_MIN_F32 );
cutensorInitContractionFind ( &handle, &find, ... );
cutensorInitContractionPlan ( &handle, &plan, &desc, &find,... );
cutensorContraction ( handle,
                  &plan,
                  alpha, A, B,
                  beta,  C, D,
                  workspace, workspaceSize, stream );
```

CONTRACTION OPERATION

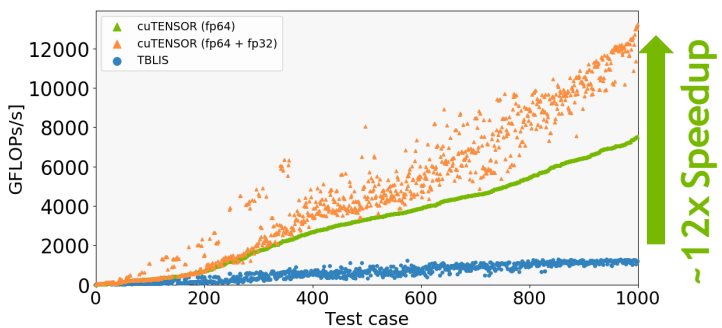
API

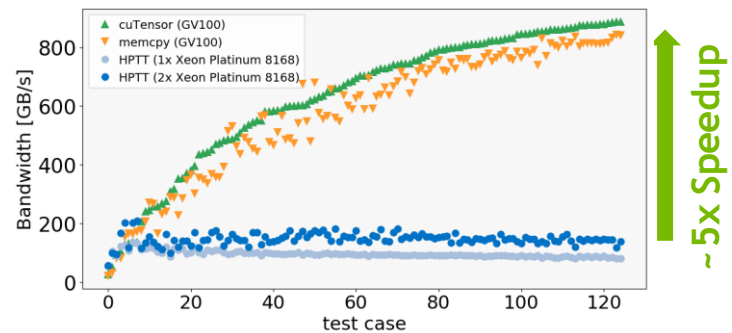
- $D_{m,n,u} = \alpha\, A_{m,k,u} * B_{k,n}$

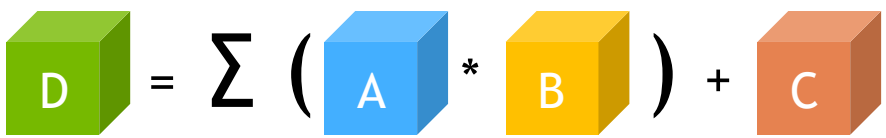
```
cutensorInitContractionDescriptor ( &handle, &desc,                  // Define Problem
                  descA, { 'm', 'k', 'u' }, 256,
                  descB, { 'k', 'n' },      256,
                  descC, { 'm', 'n', 'u' }, 256,
                  descD, { 'm', 'n', 'u' }, 256,
                  CUTENSOR_R_MIN_F32 );
cutensorInitContractionFind ( &handle, &find, ... );                 // Define search space
cutensorInitContractionPlan ( &handle, &plan, &desc, &find,... );    // Initialize plan
cutensorContraction ( handle,                                        // Execute contraction
                  &plan,
                  alpha, A, B,
                  beta,  C, D,
                  workspace, workspaceSize, stream );
```

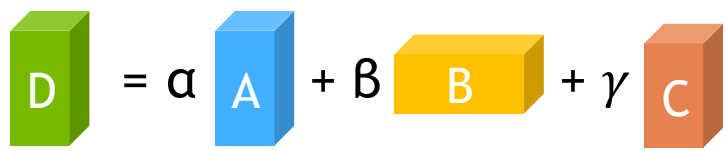
CONCLUSION

cuTENSOR: A CUDA library for high-performance tensor primitives





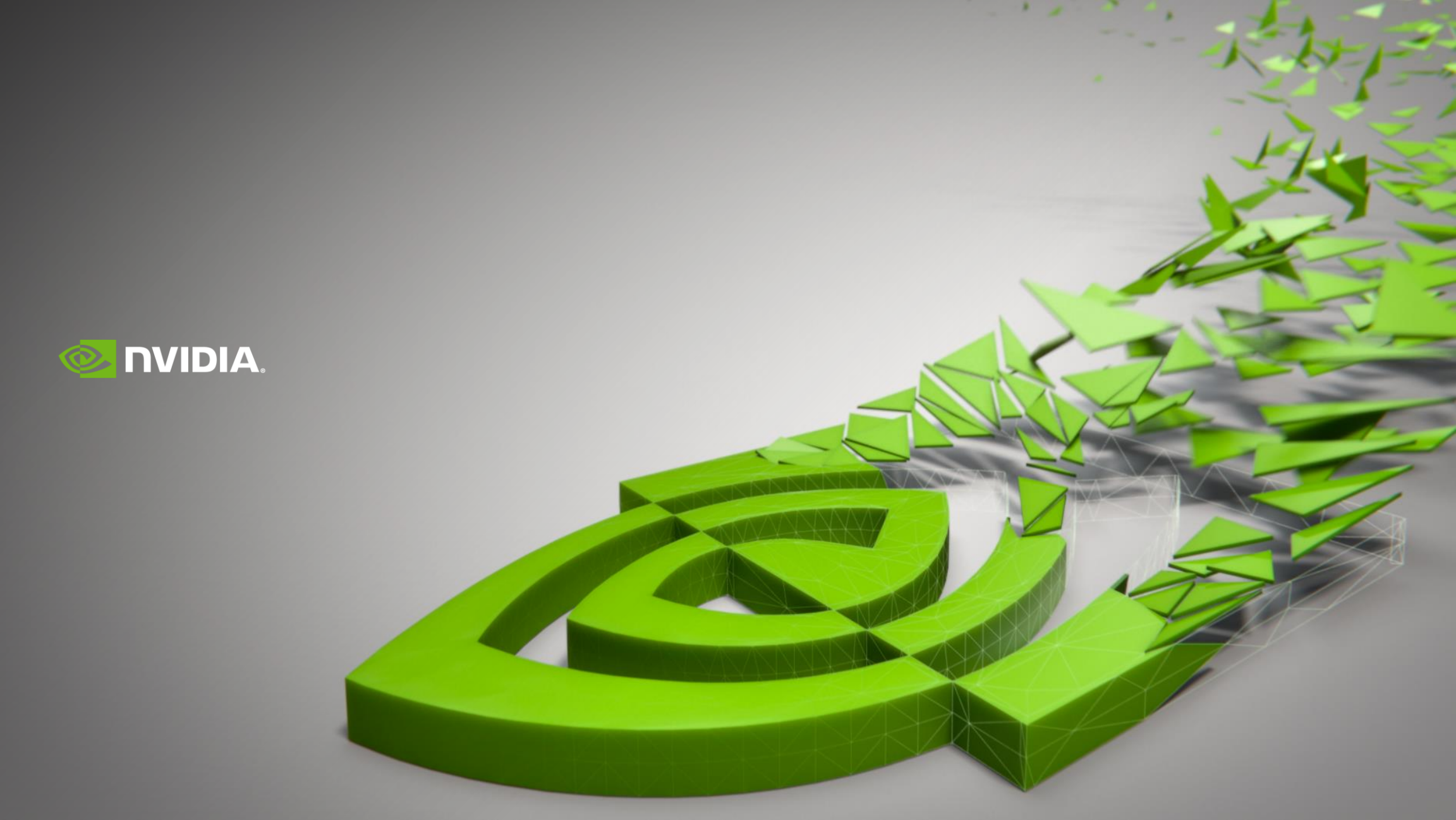




Available at: https://developer.nvidia.com/cuTENSOR

Your feedback is highly appreciated.

tcosta@nvidia.com or pspringer@nvidia.com


NVIDIA

REDUCTION OPERATION

API


```
cutensorStatus_t cutensorReduction ( const cutensorHandle_t *handle,
       const void *alpha, const void *A, const cutensorTensorDescriptor_t *descA, const int modeA[],
       const void *beta,  const void *C, const cutensorTensorDescriptor_t *descC, const int modeC[],
                                void *D, const cutensorTensorDescriptor_t *descD, const int modeD[],
       cutensorOperator_t opReduce, cutensorComputeType_t typeCompute, cudaStream_t stream );
```

REDUCTION OPERATION

API

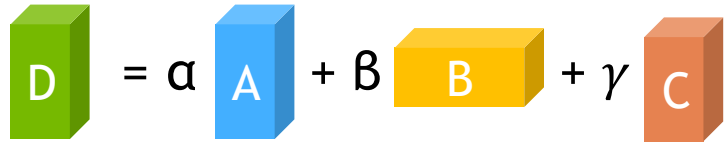

```
cutensorStatus_t cutensorReduction ( const cutensorHandle_t *handle,
       const void *alpha, const void *A, const cutensorTensorDescriptor_t *descA, const int modeA[],
       const void *beta,  const void *C, const cutensorTensorDescriptor_t *descC, const int modeC[],
                          void *D, const cutensorTensorDescriptor_t *descD, const int modeD[],
       cutensorOperator_t opReduce, cutensorComputeType_t typeCompute, cudaStream_t stream );
```

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

```
auto status = cutensorReduction ( handle,
                    alpha, A, descA, { 'c', 'w', 'h', 'n' },
                    beta,  C, descC, { 'w', 'h', 'n' },
                           D, descC, { 'w', 'h', 'n' },
                    CUTENSOR_OP_ADD, CUTENSOR_R_MIN_32F,
                    stream );
```

ELEMENT-WISE OPERATION

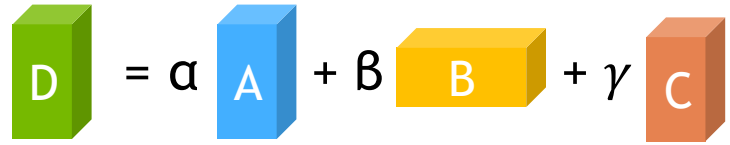
API




```
cutensorStatus_t cutensorElementwiseTrinary ( const 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



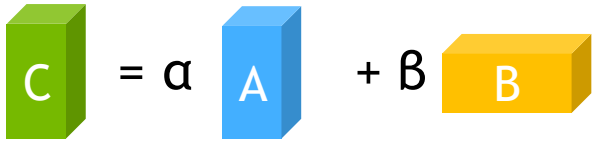

```
cutensorStatus_t cutensorElementwiseTrinary ( const cutensorHandle_t *handle,
      const void *alpha, const void *A, const cutensorTensorDescriptor_t *descA, const int modeA[],
      const void *beta,  const void *B, const cutensorTensorDescriptor_t *descB, const int modeB[],
      const void *gamma, 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 );
```

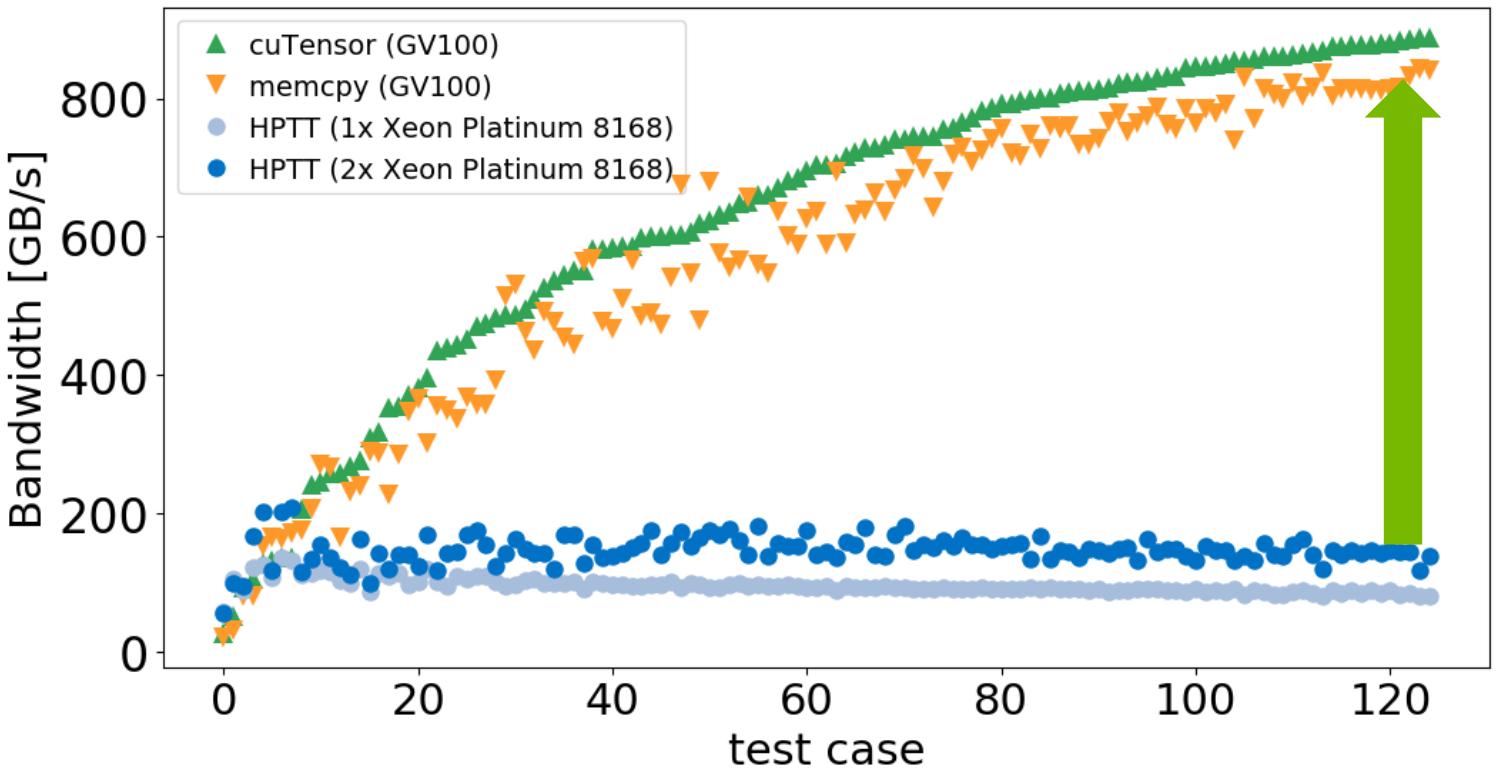
- $D_{w,h,c,n} = \min(\alpha A_{c,w,h,n}, \beta B_{c,w,h}) + \gamma C_{w,h,c,n}$

```
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, CUTENSOR_R_MIN_16F,
                  stream );
```

PERFORMANCE

Element-wise Operation





~5x over two-socket CPU

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

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