

Introduction to cuBLAS

Goran Frehse
ENSTA Paris

Disclaimer:

The following slides are excerpts with slight adaptations from the excellent course by Al Barr, Aadyot Bhatnagar, Tyler Port, Bobby Abrahamson

<http://courses.cms.caltech.edu/cs179/>

BLAS and cuBLAS

- https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms
 - BLAS defines a set of common functions for to scalars, vectors, and matrices.
 - Libraries that implement it exist in almost all major languages.
 - different functions for different number types
-
- **cuBLAS = BLAS function library for CUDA**

BLAS Levels

- Level 1: Scalar and Vector, Vector and Vector operations, $\gamma \rightarrow \alpha\chi + \gamma$
 - Level 2: Vector and Matrix operations, $\gamma \rightarrow \alpha\mathbf{A}\chi + \beta\gamma$
 - Level 3: Matrix and Matrix operations, $\mathbf{C} \rightarrow \alpha\mathbf{A}\mathbf{B} + \beta\mathbf{C}$
-
- Scalars: α, β
 - Vectors: χ, γ
 - Matrices: $\mathbf{A}, \mathbf{B}, \mathbf{C}$

cuBLAS Level 1

<https://docs.nvidia.com/cuda/cublas>

▽ 2.5. cuBLAS Level-1 Function Reference

2.5.1. `cublasI<t>amax()`

2.5.2. `cublasI<t>amin()`

2.5.3. `cublas<t>asum()`

2.5.4. `cublas<t>axpy()`

2.5.5. `cublas<t>copy()`

2.5.6. `cublas<t>dot()`

2.5.7. `cublas<t>nrm2()`

2.5.8. `cublas<t>rot()`

2.5.9. `cublas<t>rotg()`

2.5.10. `cublas<t>rotm()`

2.5.11. `cublas<t>rotmg()`

2.5.12. `cublas<t>scal()`

2.5.13. `cublas<t>swap()`

cuBLAS Level 2

<https://docs.nvidia.com/cuda/cublas>

▽ 2.6. cuBLAS Level-2 Function Reference

2.6.1. `cublas<t>gbmv()`

2.6.2. `cublas<t>gemv()`

2.6.3. `cublas<t>ger()`

2.6.4. `cublas<t>sbmv()`

2.6.5. `cublas<t>spmv()`

2.6.6. `cublas<t>spr()`

2.6.7. `cublas<t>spr2()`

2.6.8. `cublas<t>symv()`

2.6.9. `cublas<t>syr()`

2.6.10. `cublas<t>syr2()`

2.6.11. `cublas<t>tbmv()`

2.6.12. `cublas<t>tbsv()`

2.6.13. `cublas<t>tpmv()`

2.6.14. `cublas<t>tpsv()`

2.6.15. `cublas<t>trmv()`

2.6.16. `cublas<t>trsv()`

2.6.17. `cublas<t>hemv()`

2.6.18. `cublas<t>hbmv()`

2.6.19. `cublas<t>hpmv()`

2.6.20. `cublas<t>her()`

2.6.21. `cublas<t>her2()`

2.6.22. `cublas<t>hpr()`

2.6.23. `cublas<t>hpr2()`

cuBLAS Level 3

<https://docs.nvidia.com/cuda/cublas>

▽ 2.7. cuBLAS Level-3 Function Reference

2.7.1. `cublas<t>gemm()`

2.7.2. `cublas<t>gemm3m()`

2.7.3. `cublas<t>gemmBatched()`

2.7.4. `cublas<t>gemmStridedBatched()`

2.7.5. `cublas<t>symm()`

2.7.6. `cublas<t>syrk()`

2.7.7. `cublas<t>syr2k()`

2.7.8. `cublas<t>syrkx()`

2.7.9. `cublas<t>trmm()`

2.7.10. `cublas<t>trsm()`

2.7.11. `cublas<t>trsmBatched()`

2.7.12. `cublas<t>hemm()`

2.7.13. `cublas<t>herk()`

2.7.14. `cublas<t>her2k()`

2.7.15. `cublas<t>herkx()`

The various cuBLAS types

- Every function exists in different versions for different number types
 - S, s : single precision (32 bit) real float
 - D, d : double precision (64 bit) real float
 - C, c : single precision (32 bit) complex float (implemented as a float2*)
 - Z, z : double precision (64 bit) complex float
 - H, h : half precision (16 bit) real float

* float2 is a struct of two floats

cuBLAS function types

- `cublasIsamax` → `cublas I s amax`
 - I : stands for index.
 - s : this is the single precision float variant of the isamax operation
 - amax : finds a maximum
- `cublasSgemm` → `cublas S gemm`
 - S : single precision real float
 - gemm : general matrix-matrix multiplication
- `cublasHgemm`
 - H : half precision real float
 - gemm : general matrix-matrix multiplication
- `cublasDgemv` → `cublas D gemv`
 - D : double precision real float
 - gemv : general matrix vector multiplication

Using cuBLAS

- <https://developer.nvidia.com/sites/default/files/akamai/cuda/files/Misc/mygpu.pdf>
 - cuBLAS examples in different implementations
- <http://docs.nvidia.com/cuda/cublas/index.html> the official NVIDIA docs.
- Include the header “cublas_v2.h” and link the library with “-lcublas”
- must create a handle before using cuBLAS functions:

```
cublasHandle_t handle;
stat = cublasCreate(&handle);
    if (stat != CUBLAS_STATUS_SUCCESS) {
        printf ("CUBLAS initialization failed\n");
        return EXIT_FAILURE;
    }
... use cuBLAS functions ...
cublasDestroy(handle);
```

Example: Matrix-Matrix Multiplication

```
cublasSgemm(h, transpA, transpB, m, n, k, &alpha, &A, lda, &B, ldb, &beta, &C, ldc)
```

implements $C = \alpha \text{op}(A) \text{op}(B) + \beta C$

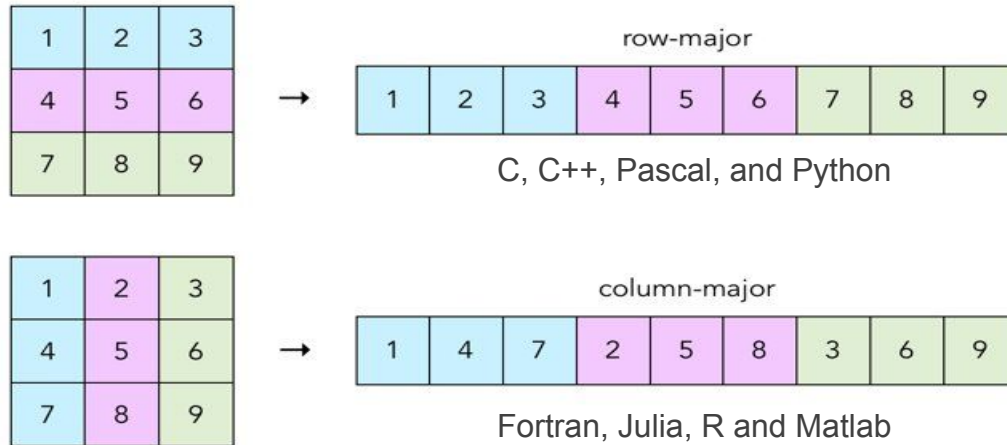
where $\text{op}(A)$ is A if $\text{transpA} = \text{CUBLAS_OP_N}$
 $\text{transpose}(A)$ if $\text{transpA} = \text{CUBLAS_OP_T}$

...similar for $\text{op}(B)$...

$\text{lda}, \text{ldb}, \text{ldc}$ = number of rows of A, B, C

m, n, k are according to the dimensions of $\text{op}(A) : m \times k$, $\text{op}(B) : k \times n$, $C : m \times n$

CuBLAS - Column Major



For efficiency, traverse matrices the way they are stored.

Array Indexing

- use an indexing macro:

```
#define IDX2C(i,j,ld) (((j)*(ld))+i)
```

Where “i” is the row, “j” is the column, and “ld” is the leading dimension.

In column major storage “ld” is the **number of rows**.