CS 179: Lecture 10

Introduction to cuBLAS

Table of contents

- Welcome to week 4
- We will be discussing how to accelerate many common Linear Algebra operations for use in lab4 through lab6.
- Based on CUDA versions of Blas –
  - https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms

Shared Memory note– caution!

- When allocating shared memory dynamically, by passing in the third parameter to the kernel call, mind how much memory you are allocating
- <<<blockNum, threadsPerBlock, sharedMemSize>>>
- If you allocate way too much memory, CUDA won’t properly error out on the kernel call and your kernel will not execute any instructions inside it.
- cuda-GDB will continue stepping correctly but not even a printf will work.

Goals for this week

- Naming, and how we use cuBLAS to accelerate linear algebra computations with already optimized implementations of Basic Linear Algebra Subroutines (BLAS).
- How we use cuBLAS to perform multiple computations in parallel.
- Learn about the cuBLAS API and why it can be difficult to read.
- Learn to use cuBLAS to write optimized cuda kernels for graphics, which we will also use later for machine learning.

What is BLAS?

- https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms
- BLAS defines a set of common linear algebra routines that we would want to apply to scalars, vectors, and matrices.
- Libraries that implement BLAS exist in almost all major languages.
- The names of these functions are opaque and can be hard to follow, so keeping a “cheat sheet” nearby is useful.
    - There are different function names for the different number types!

Example cuBLAS functions

- cublasIsamax() : “I”-s - amax. finds the smallest (first) index i in a vector that is a maximum for that vector
    - returns the least **index** of the maximum element in a vector…
- cublasSgemm() : generalized matrix matrix multiplication with single precision floats.
    - Also is how you do Vector Vector multiplication.
- cublasDtrmm() : triangular matrix matrix multiplication with double precision floats.
- See the name differences! Many different functions.

How to Read Lecture, Conventions & Symbols

- Assuming familiarity with linear algebra, but you don’t need to know too much theory.
- The symbols used in these slides will be consistent with:
  - Scalars: $\boldsymbol{\alpha}, \boldsymbol{\beta}$
  - Vectors: $\boldsymbol{\chi}, \boldsymbol{\gamma}$
  - Matrices: **A**, **B**, **C**
- BLAS (Basic Linear Algebra Subprograms) was written for FORTRAN many years ago and cuBLAS follows its conventions
  - Matrices are **indexed column major**
    - The relevant dimension is the number of columns
  - Arrays start at 1 (or 0, depending on if routine was written for FORTRAN or C)
  - We will use a macro to handle our matrix indexing as our cuBLAS arrays will be one dimensional

More, on Reading This Lecture

- There are 3 “levels” of functionality, combining scalars, vectors, and Matrices
    - Level 1: Scalar and Vector, Vector and Vector operations, vector $\boldsymbol{\gamma} \rightarrow \boldsymbol{\alpha\chi} + \boldsymbol{\gamma}$
    - Level 2: Vector and Matrix operations, vector $\boldsymbol{\gamma} \rightarrow \boldsymbol{\alpha} \mathbf{A} \boldsymbol{\chi} + \boldsymbol{\beta\gamma}$
    - Level 3: Matrix and Matrix operations, matrix $\mathbf{C} \rightarrow \boldsymbol{\alpha}\mathbf{AB} + \boldsymbol{\beta}\mathbf{C}$
- Some desired functionality like Vector Vector complex multiplication, like the kind done in lab 3, can be implemented using the generalized matrix matrix multiplication (gemm.)
    - Vectors are just 1xN matrices right?
    - No need to have even more functions for these cases!
        - Makes finding the function name you want more bothersome
        - Writing a proper “templated” C++ wrapper around cuBLAS is an interesting idea.

What is cuBLAS good for?

- Anything that uses heavy linear algebra computations (on dense matrices) can likely benefit from GPU acceleration
  - Graphics
  - Machine learning (this will be covered next week)
  - Computer vision
  - Physical simulations
  - Finance
  - etc…..
- cuBLAS excels in situations where you want to maximize your performance by batching multiple kernels using streams.
  - Like making many small matrix-matrix multiplications on dense matrices

The various cuBLAS types

- All of the functions defined in cuBLAS have four versions which correspond to the four types of numbers in CUDA C
  - 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

Sample cuBLAS function names w/ types

- cublasIsamax -> cublas "I," s, amax
  - cublas : the cuBLAS prefix since the library doesn't implement a namespaced API
  - I : stands for index. Cuda naming left over from Fortran!
  - s : this is the single precision float variant of the isamax operation
  - amax : finds a maximum
  - Returns smallest Index "i" of the earliest max element.
- cublasSgemm → cublas S gemm
  - cublas : the prefix
  - S : single precision real float
  - gemm : general matrix-matrix multiplication
- cublasHgemm
  - Same as before except half precision
- cublasDgemv → cublas D gemv
  - D : double precision real float
  - gemv : general matrix vector multiplication

How to actually use cuBLAS

- https://developer.nvidia.com/sites/default/files/akamai/cuda/files/Misc/mygpu.pdf
  - This pdf contains many various cuBLAS examples in different implementation.
  - Referencing this and the official NVIDIA docs are highly recommended!
- http://docs.nvidia.com/cuda/cublas/index.html the official NVIDIA docs.
  - As usual the actual NVIDIA documentation is terrible, but it is the canonical index of all the various functions and types as well as what they "do", the explanations leave something to be desired.
- Include the header "cublas_v2.h" and link the library with "-lcublas", this is done in the Makefile given.
- Cublas uses handles just like cuFFT in lab 3 and most of the current CUDA libraries.

  - https://en.wikipedia.org/wiki/Handle_(computing)

Using cuBLAS w C Fortran, Python matrices

- cuBLAS selected **column-first indexing**. makes annoying to use in C code.
- See which parameter should be what, in the following reference, to see which matrix should be transposed and which one should not be.
  - https://peterwittek.com/cublas-matrix-c-style.html
- To get around this type of issue, in CS179 we implement 1-D arrays for our 2-D arrays, where we specify components directly with an **indexing macro**
- For Python Interfaces, see
  - https://scikit-cuda.readthedocs.io/en/latest/

Numpy vs cuBLAS

numpy	cuBLAS (<T> is one of S, D, C, Z)
numpy.dot($\boldsymbol{\alpha}, \boldsymbol{\chi}$)	cublas<T>dot($\boldsymbol{\alpha}, \boldsymbol{\chi}$)
numpy.dot($\boldsymbol{\chi}, \boldsymbol{\gamma}$)	cublas<T>dot($\boldsymbol{\chi}, \boldsymbol{\gamma}$)
numpy.dot($\boldsymbol{\chi}$, **A**)	cublas<T>dot($\boldsymbol{\chi}$, **A**)
numpy.dot(**A, B**)	cublas<T>dot(**A, B**)
numpy.dot($\boldsymbol{\chi}$, **A**)	

The multiplication (nx1)(1xm)=(nxm) is performed by cublasDger function.

Numpy vs math vs cuBLAS

numpy	math	cuBLAS (<T> is one of S, D, C, Z, H)
numpy.multiply($\boldsymbol{\alpha}, \boldsymbol{\chi}$)	$(\lambda\mathbf{A})_{ij} = \lambda(\mathbf{A})_{ij}$	cublas<T>gemm($\boldsymbol{\alpha}, \boldsymbol{\chi}$)
numpy.multiply($\boldsymbol{\chi}, \boldsymbol{\gamma}$) (Multiply arguments element-wise)	$(A \circ B)_{i,j} = (A)_{i,j}(B)_{i,j}.$	cublas<T>gemm($\boldsymbol{\chi}, \boldsymbol{\gamma}$)
numpy.multiply($\boldsymbol{\chi}$, **A**)	**A**$\boldsymbol{\chi}$ = C	cublas<T>gemm($\boldsymbol{\chi}$, **A**)
numpy.multiply(**A, B**)	$\boldsymbol{C} \leftarrow \alpha \boldsymbol{A}\boldsymbol{B} + \beta \boldsymbol{C}$	cublas<T>gemm(**A, B**)

Array Indexing

- Our arrays are linearized into one dimension, so we will 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.

Error Checking

- Like CUDA and cuFFT, cuBLAS has a similar but slightly different status return type.
- cublasStatus_t
- We will use a similar macro to gpuErrchk and the cuFFT version to check for cuBLAS errors.

Streaming Parallelism

- Used to overlap computations when we need the results of several linear algebra operations to continue our program.
- Makes “more efficient” use of the GPU hardware than a naively, or even moderately optimized handwritten written kernel.
  - If you just need to multiply two vectors element wise and normalize (scale) you can use the GEMM operation.

Streaming Parallelism

- Use cudaStreamCreate() to create a stream for computation.
- Assign the stream to a cublas library routine using cublasSetStream()
- When you call the routine it will operate in parallel (asynchronously) with other streaming cublas calls to maximize parallelism.
- Should pass your constant scalars by reference to help maximize this benefit.

Streaming Issue. Limit of 16 Kernels!

- In general you won't be able to have more than 16 kernels running simultaneously on a GPU
  - Even though we are using a library there is still a kernel being run on the GPU, you just aren't writing it yourself.