The CUBLAS and CULA libraries

Will Landau

Iowa State University

October 28, 2013

Outline

CUBLAS overview

Using CUBLAS

CULA

Outline

CUBLAS overview

Using CUBLAS

CULA

CUBLAS

- **CUBLAS**: CUda Basic Linear Algebra Subroutines, the CUDA C implementation of BLAS.
- Consider scalars α, β, vectors x, y, and matrices A, B, C.
- 3 "levels of functionality":
  - Level 1: $y \mapsto \alpha x + y$ and other vector-vector routines.
  - Level 2: $y \mapsto \alpha A x + \beta y$ and other vector-matrix routines.
  - Level 3: $C \mapsto \alpha AB + \beta C$ and other matrix-matrix routines.

Level 1 functions

- Let α be a scalar, x, y, and m be vectors, $G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$ be some 2×2 rotation matrix, and H be an arbitrary 2×2 matrix,

in R	float	double
which.max(x)	cublasIsamax()	cutlasIdamax()
which.min(x)	cublasIsamin()	cublasIdamin()
sum(abs(x))	cublasSasum()	cublasDasum()
α * x + y-> y	cublasSaxpy()	cublasDaxpy()
x -> y	cublasScopy()	cublasDcopy()
sum(x * y)	cublasSdot()	cublasDdot()
sqrt(sum(x^2))	cublasSnrm2()	cublasDnrm2()
G %*% x	cublasSrot()	cublasDrot()
H %*% x	cublasSrotm()	cublasDrotm()
α * x -> x	cublasSscal()	cublasDscal()
x -> m; y -> x; m -> y	cublasSswap()	cublasDswap()

- Like everything in CUBLAS, there are also analogous functions for cuComplex and cuDoubleComplex types.

The CUBLAS and CULA libraries
Will Landau
CUBLAS overview
Using CUBLAS
CULA

Example level 2 functions

$$\alpha \text{op}(A) \cdot x + \beta y \mapsto y$$

where

$$\text{op}(A) = \begin{cases} A & \texttt{transa == CUBLAS\_OP\_N} \\ A^T & \texttt{transa == CUBLAS\_OP\_T} \\ A^H & \texttt{transa == CUBLAS\_OP\_C} \end{cases}$$

type of matrix, A	float	double
any $m \times n$	cublasSgemv()	cublasDgemv()
banded $m \times n$	cublasSgbmv()	cublasDgbmv()
symmetric, banded	cublasSbmv()	cublasDbmv()
symmetric, packed format	cublasSspmv()	cublasDspmv()
symmetric, triangular	cublasSsymv()	cublasDsymv()

Example level 3 functions

- `cublasSgemm()` and `cublasDgemm()`: for any compatible matrices A, B, and C,

$$\alpha \cdot \text{op}(A)\text{op}(B) + \beta C \mapsto C$$

- `cublasSgemmBatched()` and `cublasDgemmBatched()`: for arrays of compatible matrices $A[]$, $B[]$, and $C[]$,

$$\alpha \cdot \text{op}(A[i])\text{op}(B[i]) + \beta C[i] \mapsto C[i]$$

- `cublasStrsm()` and `cublasDtrsm()` solve for X when A is triangular:

$$\begin{cases} \text{op}(A)X = \alpha B & \texttt{trans == CUBLAS\_SIDE\_LEFT} \\ X\text{op}(A) = \alpha B & \texttt{trans == CUBLAS\_SIDE\_RIGHT} \end{cases}$$

Outline

CUBLAS overview

Using CUBLAS

CULA

Implementation of matrices

- Matrices stored in column major order in linear arrays of memory. Array A,

1	1	2	3	5	8	13	21	34	55	89	144

encodes matrix B,

$$\begin{bmatrix} 1 & 5 & 32 \\ 1 & 8 & 55 \\ 2 & 13 & 89 \\ 3 & 21 & 144 \end{bmatrix}$$

- Index by

$$B[\text{row } i, \text{ col } j] = A[j \cdot ld + i]$$

where ld is the lead dimension of the matrix (column length for column major order matrices).

- Use a macro for indexing:

```
#define IDX2F(i, j, ld) j * ld + i
```

The CUBLAS and CULA libraries
Will Landau
CUBLAS overview
Using CUBLAS
CULA

CUBLAS context

- For CUBLAS version $\geq$ 4.0, you must create a CUBLAS context:

```
cublasHandle_t handle;
cublasCreate(&handle);

// your code

cublasDestroy(handle);
```

- Pass `handle` to every CUBLAS function in your code.
- This approach allows the user to use multiple host threads and multiple GPUs.

CUBLAS helper functions

- You don't actually need them, but you might see them:
  - `cublasSetVector()`
  - `cublasGetVector()`
  - `cublasSetMatrix()`
  - `cublasGetMatrix()`

Choosing the right header file

- 2 choices of API
  - `cublas_v2.h`: API for CUBLAS version 4.0 and above.
  - `cublas.h`: older API for programs written with CUBLAS version < 4.0.
- Additions to newer API:
  - `cublasCreate()` initializes the handle to the CUBLAS library context.
  - Scalars can be passed by reference or by value to device functions.
  - All CUBLAS functions return an error status, `cublasStatus_t`.
  - `cublasAlloc()` and `cublasFree()` are deprecated. Use `cudaMalloc()` and `cudaFree()` instead.
  - `cublasSetKernelStream()` was renamed `cublasSetStream()`.

Compiling with CUBLAS

1. Include either cublas_v2.h or cublas.h in your source
2. Link the CUBLAS library with the -lcublas flag.

- Example2.cu:

```
> nvcc -lcublas Example2.cu -o Example2.
> ./Example2
       1       7     13     19     25     31
       2       8     14     20     26     32
       3    1728    180    252    324    396
       4     160     16     22     28     34
       5     176     17     23     29     35
```

Example2.cu

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cuda_runtime.h>
#include <cublas_v2.h>
#define M 6
#define N 5
#define IDX2F(i,j,ld) ((((j)-1)*(ld))+((i)-1))

static __inline__ void modify (cublasHandle_t handle, float *m, int
    ldm, int n, int p,
  int q, float alpha, float beta){
  cublasSscal (handle, n - p+1, &alpha, &m[IDX2F(p,q,ldm)], ldm);
  cublasSscal (handle, ldm - p+1, &beta, &m[IDX2F(p,q,ldm)], 1);
}

int main (void){
  cudaError_t cudaStat;
  cublasStatus_t stat;
  cublasHandle_t handle;
  int i, j;
  float* devPtrA;
  float* a = 0;
  a = (float *)malloc (M * N * sizeof (*a));
  if (!a) {
    printf ("host memory allocation failed");
    return EXIT_FAILURE;
  }
```

Example2.cu

```
  for (j = 1; j <= N; j++) {
    for (i = 1; i <= M; i++) {
      a[IDX2F(i,j,M)] = (float)((i-1) * M + j);
    }
  }

  cudaStat = cudaMalloc ((void**)&devPtrA, M*N*sizeof(*a));
  if ( cudaStat != cudaSuccess ) {
    printf ("device memory allocation failed");
    return EXIT_FAILURE;
  }

  stat = cublasCreate(&handle);
  if ( stat != CUBLAS_STATUS_SUCCESS ) {
    printf ("CUBLAS initialization failed\n");
    return EXIT_FAILURE;
  }

  stat = cublasSetMatrix (M, N, sizeof(*a), a, M, devPtrA, M);

   if(stat != CUBLAS_STATUS_SUCCESS) {
    printf("data download failed");
    cudaFree(devPtrA);
    cublasDestroy(handle);
    return EXIT_FAILURE;
  }
```

Example2.cu

```


  modify (handle, devPtrA, M, N, 2, 3, 16.0f, 12.0f);

  stat = cublasGetMatrix (M, N, sizeof(*a), devPtrA, M, a, M);
  if( stat != CUBLAS_STATUS_SUCCESS ) {
    printf ("data upload failed");
    cudaFree (devPtrA);
    cublasDestroy ( handle );
    return EXIT_FAILURE;
  }

  cudaFree ( devPtrA );
  cublasDestroy ( handle );

  for (j = 1; j <= N; j++) {
    for (i = 1; i <= M; i++) {
      printf ("%7.0f", a[IDX2F(i,j,M)]);
    }
    printf ( "\n" );
  }
  return EXIT_SUCCESS;
}
```

Example: `ols.cu`

- I will attempt to solve the least squares problem,

$$y = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

  by computing the solution,

$$\widehat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T y$$

Example: ols.cu

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cuda_runtime.h>
#include <cublas_v2.h>
#include <cula.h>
#include <math.h>

#define I(i, j, ld) j * ld + i

#define CUDA_CALL(x) {if((x) != cudaSuccess){ \
  printf("CUDA error at %s:%d\n",__FILE__,__LINE__); \
  printf("  %s\n", cudaGetErrorString(cudaGetLastError())); \
  exit(EXIT_FAILURE);}}

float rnorm(){
  float r1 = ((float) rand()) / ((float) RAND_MAX);
  float r2 = ((float) rand()) / ((float) RAND_MAX);
  return sqrt( -2 * log(r1) ) * cos(2 * 3.1415 * r2);
}

int main(){
  int i, j;
  int n = 10;
  int p = 3;
  int* ipiv;
  float k;
  float *X, *XtX, *XtY, *beta, *Y, *dX, *dXtX, *dXtY, *dbeta, *dY;
```

Example: ols.cu

```
  float *a, *b;
  a = (float*) malloc(sizeof(*X));
  b = (float*) malloc(sizeof(*X));
  *a = 1.0;
  *b = 0.0;

  cublasHandle_t handle;
  cublasCreate(&handle);

  X = (float*) malloc(n * p * sizeof(*X));
  XtX = (float*) malloc(p * p * sizeof(*X));
  XtY = (float*) malloc(p * sizeof(*X));
  beta = (float*) malloc(p * sizeof(*X));
  Y = (float*) malloc(n * sizeof(*X));

  CUDA_CALL(cudaMalloc((void**) &ipiv, p * p * sizeof(*ipiv)));
  CUDA_CALL(cudaMalloc((void**) &dX, n * p * sizeof(*X)));
  CUDA_CALL(cudaMalloc((void**) &dXtX, p * p * sizeof(*X)));
  CUDA_CALL(cudaMalloc((void**) &dXtY, p * sizeof(*X)));
  CUDA_CALL(cudaMalloc((void**) &dbeta, p * sizeof(*X)));
  CUDA_CALL(cudaMalloc((void**) &dY, n * sizeof(*X)));
```

Example: ols.cu

```
  printf("Y\t\tX\n");
  for(i = 0; i < n; i++){
    k = (float) i;
    X[I(i, 0, n)] = 1.0;
    X[I(i, 1, n)] = k / 10.0;
    X[I(i, 2, n)] = k * k / 10.0;
    Y[i] = (k - 5.0) * (k - 2.3) / 3.0 + rnorm();

    printf("%0.2f\t\t", Y[i]);
    for(j = 0; j < p; j++){
      printf("%0.2f\t", X[I(i, j, n)]);
    }
    printf("\n");
  }
  printf("\n");

  CUDA_CALL(cudaMemcpy(dX, X, n * p * sizeof(float),
        cudaMemcpyHostToDevice));
  CUDA_CALL(cudaMemcpy(dY, Y, n * sizeof(float),
        cudaMemcpyHostToDevice));

  cublasSgemm(handle, CUBLAS_OP_T, CUBLAS_OP_N, p, p, n,
    a, dX, n, dX, n, b, dXtX, p);

  CUDA_CALL(cudaMemcpy(XtX, dXtX, p * p * sizeof(float),
        cudaMemcpyDeviceToHost));
```

Example: ols.cu

```
  printf("XtX\n");
  for(i = 0; i < p; i++){
    for(j = 0; j < p; j++){
      printf("%0.2f\t", XtX[I(i, j, p)]);
    }
    printf("\n");
  }
  printf("\n");
```

Output of code so far

```
> nvcc -I /usr/local/cula/include -L /usr/local/cula/lib64 -
    lcula_core -lcula_lapack -lcublas -lcudart ols.cu -o ols
> ./ols
Y X
3.37 1.00 0.00 0.00
1.94 1.00 0.10 0.10
0.44 1.00 0.20 0.40
-0.30 1.00 0.30 0.90
-2.08 1.00 0.40 1.60
-0.84 1.00 0.50 2.50
-0.18 1.00 0.60 3.60
3.40 1.00 0.70 4.90
5.51 1.00 0.80 6.40
7.39 1.00 0.90 8.10

XtX
10.00 4.50 28.50
4.50 2.85 20.25
28.50 20.25 153.33
```

The CUBLAS and CULA libraries
Will Landau
CUBLAS overview
Using CUBLAS
CULA

Example: `ols.cu`

- We have X^TX, but which we need to invert in order to compute our solution,

$$\widehat{\beta} = (X^TX)^{-1}X^Ty$$

- But CUBLAS can only invert triangular matrices!

Enter CULA: CUDA LAPACK

```c
  culaInitialize();

  culaDeviceSgetrf(p, p, dXtX, p, ipiv);
  culaDeviceSgetri(p, dXtX, p, ipiv);

  CUDA_CALL(cudaMemcpy(XtX, dXtX, p * p * sizeof(float),
       cudaMemcpyDeviceToHost));

  printf("XtX^(-1)\n");
  for(i = 0; i < p; i++){
    for(j = 0; j < p; j++){
      printf("%0.2f\t", XtX[I(i, j, p)]);
    }
    printf("\n");
  }
  printf("\n");

  cublasSgemm(handle, CUBLAS_OP_T, CUBLAS_OP_N, p, 1, n,
    a, dX, n, dY, n, b, dXtY, p);

  cublasSgemv(handle, CUBLAS_OP_N, p, p,
    a, dXtX, p, dXtY, 1, b, dbeta, 1);

  CUDA_CALL(cudaMemcpy(beta, dbeta, p * sizeof(float),
       cudaMemcpyDeviceToHost));

  printf("CUBLAS/CULA matrix algebra parameter estimates:\n");
  for(i = 0; i < p; i++){
    printf("beta_%i = %0.2f\n", i, beta[i]);
  }
  printf("\n");
```

CULA's culaSgels() does least squares for you

```
  culaSgels('N', n, p, 1, X, n, Y, n);

  printf("culaSgels Parameter estimates:\n");
  for(i = 0; i < p; i++){
    printf("beta_%i = %0.2f\n", i, Y[i]);
  }
  printf("\n");

  culaShutdown();
  cublasDestroy(handle);

  free(a);
  free(b);
  free(X);
  free(XtX);
  free(XtY);
  free(beta);
  free(Y);

  CUDA_CALL(cudaFree(dX));
  CUDA_CALL(cudaFree(dXtX));
  CUDA_CALL(cudaFree(dXtY));
  CUDA_CALL(cudaFree(dbeta));
  CUDA_CALL(cudaFree(dY));
}
```

Rest of the output

```
XtX^(-1)
0.62 -2.59 0.23
-2.59 16.55 -1.70
0.23 -1.70 0.19

CUBLAS/CULA matrix algebra parameter estimates:
beta_0 = 3.78
beta_1 = -25.53
beta_2 = 3.36

culaSgels Parameter estimates:
beta_0 = 3.78
beta_1 = -25.53
beta_2 = 3.36
```

Outline

CUBLAS overview

Using CUBLAS

CULA

The CUBLAS and CULA libraries
Will Landau
CUBLAS overview
Using CUBLAS
CULA

More on CULA

- `CULA`: the CUDA C implementation of LAPACK
- Features:
  - More matrix algebra routines
  - Factorizations: LU, QR, RQ, QL, SVD, and Cholesky
  - Solving systems of linear equations (matrix inversion)
  - Least squares
  - Eigenvalue solvers
- Interfaces (collections of functions)
  - **Standard**: users need not micromanage GPU memory or copy data to or from the GPU.
  - **Device**: users need explicitly to allocate GPU memory and copy to and from the GPU.
- Be careful of the standard interface functions: they're convenient, but they copy to and from the GPU with every call.

CULA naming conventions

$$\underbrace{\text{cula}}_{\text{prefix}}\underbrace{\text{S}}_{\text{type}}\underbrace{\text{ge}}_{\text{matrix}}\underbrace{\text{qrf}}_{\text{routine}}$$

- Prefix: `cula` for standard interface, `culaDevice` for device interface:
- Type: single precision (`S`), single precision complex (`C`), double precision real (`D`), or double precision complex (`Z`).
- Matrix:

`bd`	Bidiagonal
`ge`	General
`gg`	General matrices, generalized problem
`he`	Hermitian symmetric
`or`	(Real) orthogonal
`sb`	Symmetric, banded
`sy`	Symmetric
`tr`	Triangular
`un`	(Complex) unitary

CUBLAS overview
Using CUBLAS
CULA

CULA naming conventions

$$\underbrace{\text{cula}}_{\text{prefix}}\ \underbrace{\text{S}}_{\text{type}}\ \underbrace{\text{ge}}_{\text{matrix}}\ \underbrace{\text{qrf}}_{\text{routine}}$$

- Routine:

`trf`	Triangular factorization
`sv`	Factor a matrix and solve system of linear equations
`qrf`	QR factorization without pivoting
`svd`	Singular value decomposition
`ls`	Solve over- or under-determined linear system

- Consult the CULA manual for other routines.

The CUBLAS and CULA libraries
Will Landau
CUBLAS overview
Using CUBLAS
CULA

Compiling with CULA

- Include cula_lapack.h for the standard interface, cula_lapack_device.h for the device interface, or cula.h for both.
- Compile on impact1 with:

```
nvcc -I /usr/local/cula/include -L /usr/local/cula/lib64 -lcula_core -lcula_lapack -lcublas -lcudart your_source.cu -o your_binary
```

  - -I /usr/local/cula/include tells the compiler, nvcc, where to find the header files.
  - -L /usr/local/cula/lib64 tells nvcc where the CULA library is (the 64-bit version in this case).
  - -lcula_core -lcula_lapack -lcublas -lcudart links the required libraries to your binary.

Minimal working example: `mwe.cu`

```
#include <cula.h>
#include <stdlib.h>
#include <stdio.h>

int main(){

  culaStatus s;
  s = culaInitialize();

  if(s != culaNoError)
    printf("%s\n", culaGetErrorInfo());

  /* ... Your code ... */

  culaShutdown();
}
```

Minimal working example: mwe.cu

```
> nvcc -I /usr/local/cula/include -L /usr/local/cula/lib64 -lcula_core -lcula_lapack -lcublas -lcudart mwe.cu -o mwe
> ./mwe
>
```

Outline

CUBLAS overview

Using CUBLAS

CULA

Resources

- Guides:
  1. CUDA Toolkit 4.2 CUBLAS Library
  2. "CULA Programmers Guide". CULA Tools.
  3. "CULA Reference Manual". CULA Tools.
- Code from today:
  - Example2.cu
  - ols.cu
  - mwe.cu
- Other example code:
  - simpleCUBLAS.cpp
  - ae.cu
  - de.cu
  - deviceInterface.c
  - ll.cu
  - systemSolve.c

That's all for today.

- Series materials are available at http://will-landau.com/gpu.

The CUBLAS and CULA libraries

Will Landau

CUBLAS overview

Using CUBLAS

CULA