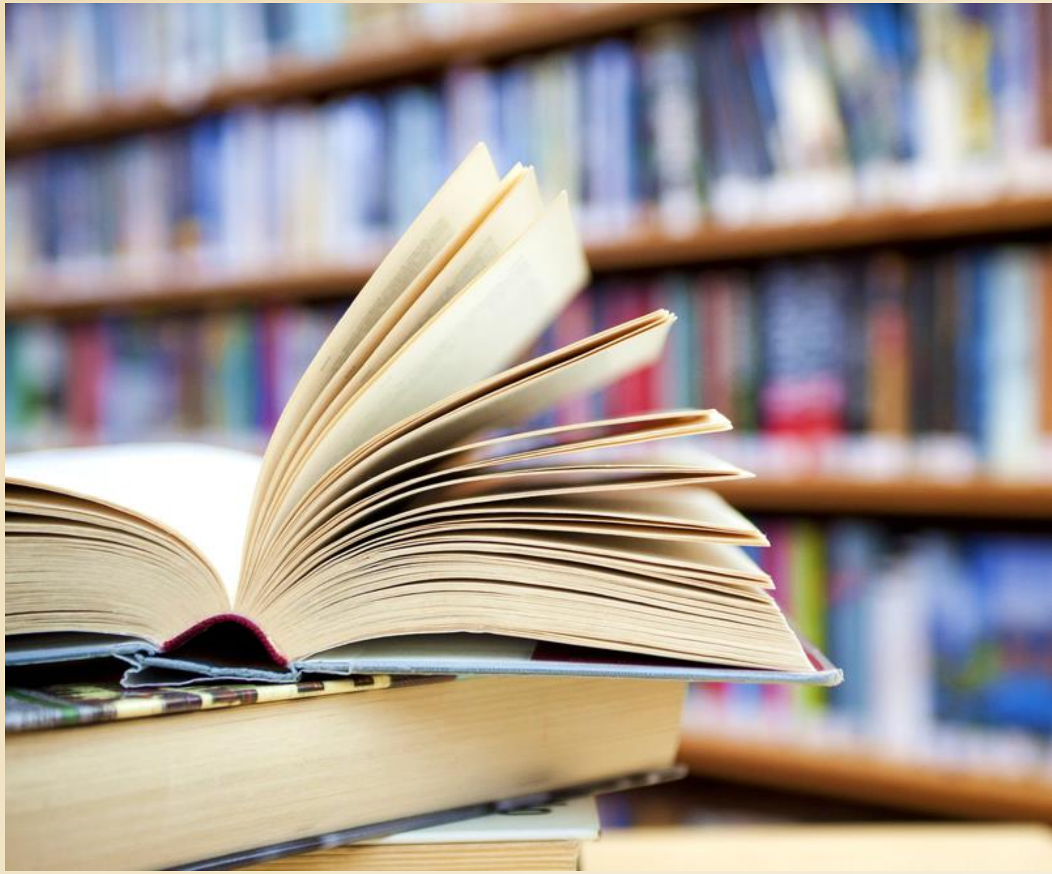
SOLUTION OF TRIDIAGONAL LINEAR SYSTEMS IN CUDA



Solving a tridiagonal system in CUDA

$$
N-2\left\{\underbrace{\begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & 0 & 0 \\ \dots & & & & & & & \\ 0 & 0 & 0 & 0 & \dots & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & \dots & 0 & -1 & 2 \end{bmatrix}}_{N-2}\right.\begin{bmatrix} u_2 \\ u_3 \\ u_4 \\ \dots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} f_2 + \dfrac{u_a}{\Delta x^2} \\ f_3 \\ f_4 \\ \dots \\ f_{N-2} \\ f_{N-1} + \dfrac{u_n}{\Delta x^2} \end{bmatrix}
$$

The solution of a tridiagonal linear system can be obtained by routines from the cuSPARSE library:

- cusparse<t>gtsv
- Cusparse<t>gtsv_nopivot
- cusparse<t>gtsv2

cusparse<t>gtsv for double precision linear systems

cusparseDgtsv(	
cusparseHandle_t handle,	cusparse handle
int m,	size of the linear system ($m \geq 3$)
int n,	Number of right-hand sides, columns of matrix B
const double *dl,	Dense array containing the lower diagonal
const double *d,	Dense array containing the main diagonal
const double *du,	Dense array containing the upper diagonal
double *B,	Dense input/output matrix of multiple right-hand sides/solutions
int ldb)	Leading dimension of B

cusparseDgtsv solves a tridiagonal linear systems $\underline{\underline{A}}\ \underline{x} = \underline{b}$ with multiple right-hand sides

cusparse<t>gtsv for double precision linear systems

```
cusparseDgtsv(
    cusparseHandle_t handle,
    int m,
    int n,
    const double *dl,
    const double *d,
    const double *du,
    double *B,
    int ldb)
```

Handle on the cuSPARSE context, which the user must initialize by calling **cusparseCreate()** prior to calling any other library function. The handle created and returned by **cusparseCreate()** must be passed to every cuSPARSE function.

cusparse<t>gtsv for double precision linear systems

```
cusparseDgtsv(
    cusparseHandle_t handle,
    int m,
    int n,
    const double *dl,
    const double *d,
    const double *du,
    double *B,
    int ldb)
```

Size of the linear system ($m \geq 3$). For the problem at hand, $m = N - 2$

cusparse<t>gtsv for double precision linear systems

```
cusparseDgtsv(
    cusparseHandle_t handle,
    int m,
    int n,
    const double *dl,
    const double *d,
    const double *du,
    double *B,
    int ldb)
```

The cusparseDgtsv solves many tridiagonal linear systems $\underline{\underline{A}}\,\underline{x} = \underline{b}$ with the same system matrix $\underline{\underline{A}}$, but multiple right-hand sides $\underline{b}$. This parameter indicates the number of solved linear systems, namely, the number of multiple right-hand sides.

cusparse<t>gtsv for double precision linear systems

```
cusparseDgtsv(
    cusparseHandle_t handle,
    int m,
    int n,
    const double *dl,
    const double *d,
    const double *du,
    double *B,
    int ldb)
```

dl is a vector of length m.

On assuming Matlab-like indexing for $\underline{\underline{A}}$, that is, indexing starting from $i = 1$,

$$dl(i) \coloneqq A(i, i-1), i = 1,2, \dots, m$$

The first element of dl is out-of-bound ($dl(1) \coloneqq A(1,0)$), so, for cusparseDgtsv to properly work, $dl(1)$ MUST be 0.

cusparse<t>gtsv for double precision linear systems

```
cusparseDgtsv(
    cusparseHandle_t handle,
    int m,
    int n,
    const double *dl,
    const double *d,
    const double *du,
    double *B,
    int ldb)
```

d is a vector of length m.

On assuming Matlab-like indexing for $\underline{\underline{A}}$, that is, indexing starting from $i = 1$,

$$d(i) := A(i,i), i = 1,2,\dots,m$$

cusparse<t>gtsv for double precision linear systems

```
cusparseDgtsv(
    cusparseHandle_t handle,
    int m,
    int n,
    const double *dl,
    const double *d,
    const double *du,
    double *B,
    int ldb)
```

du is a vector of length m.

On assuming Matlab-like indexing for $\underline{\underline{A}}$, that is, indexing starting from $i = 1$,

$$du(i) \coloneqq A(i, i+1), i = 1,2, \dots, m$$

The last element of du is out-of-bound ($du(m) \coloneqq A(m, m+1)$), so, for cusparseDgtsv to properly work, $du(m)$ MUST be 0.

cusparse<t>gtsv for double precision linear systems

```
cusparseDgtsv(
    cusparseHandle_t handle,
    int m,
    int n,
    const double *dl,
    const double *d,
    const double *du,
    double *B,
    int ldb)
```

On input, B is a matrix whose columns contain the multiple right-hand sides. On output, it is is a matrix whose columns contain the different solutions of the multiple linear systems.

cusparse<t>gtsv for double precision linear systems

```
cusparseDgtsv(
    cusparseHandle_t handle,
    int m,
    int n,
    const double *dl,
    const double *d,
    const double *du,
    double *B,
    int ldb)
```

In a column-major ordering (which is the case of all CUDA libraries), the LDA is used to define the distance in memory between elements of two consecutive columns which have the same row index.

Consider a 100x100 matrix A stored in a 100x100 array. In this case $LDA = N$.

Now suppose that one wants to work only on the submatrix $A(91:100, 1:100)$. In this case, the number of rows is 10, but $LDA = 100$. If one calls $B = A(91:100, 1:100)$, then $B(1,1)$ and $B(1,2)$ are 100 memory locations far from each other.

cusparse<t>gtsv_nopivot

cusparse<t>gtsv does perform pivoting when solving the tridiagonal linear systems, to gain accuracy and stability at the expense of computation time.

cusparse<t>gtsv_nopivot does not perform any pivoting. It achieves better performance when m is a power of 2.

The syntax of cusparse<t>gtsv_nopivot is exactly the same as of cusparse<t>gtsv, apart from the name.

cusparse<t>gtsv2

cusparse<t>gtsv requires significant amount of temporary extra storage ($\min(m, 8) \times (3 + n) \times sizeof(< t >)$). The temporary storage is dynamically managed from within the routine by cudaMalloc and cudaFree which stop concurrency.

cusparse<t>gtsv2 does not use dynamic memory allocation from inside, but uses static allocation from outside not to stop concurrency.

This function requires a buffer size returned by gtsv2_bufferSizeExt. The address of pBuffer MUST be multiple of 128 bytes. If it is not, CUSPARSE_STATUS_INVALID_VALUE is returned.

cusparse<t>gtsv2 for double precision linear systems

```
cusparseDgtsv2(
    cusparseHandle_t handle,
    int m,
    int n,
    const double *dl,
    const double *d,
    const double *du,
    double *B,
    int ldb,
    void *pBuffer)
```

The syntax is exactly the same as for cusparse<t>gtsv, apart from the pBuffer argument.

pBuffer is an array, which MUST be allocated by the user, whose size is returned by gtsv2_bufferSizeExt.

cusparse<t>gtsv2_bufferSizeExt for double precision linear systems

```
cusparseDgtsv2_bufferSizeExt(
    cusparseHandle_t handle,
    int m,
    int n,
    const double *dl,
    const double *d,
    const double *du,
    const double *B,
    int ldb,
    size_t *bufferSizeInBytes)
```

The syntax is exactly the same as for cusparse<t>gtsv2, apart from the bufferSizeInBytes argument.

Number of bytes of the buffer used in the gtsv2.

cusparse<t>gtsv2_nopivot

cusparse<t>gtsv2 does perform pivoting when solving the tridiagonal linear systems, to gain accuracy and stability at the expense of computation time.

cusparse<t>gtsv2_nopivot does not perform any pivoting. It achieves better performance when m is a power of 2.

The syntax of cusparse<t>gtsv2_nopivot is exactly the same as of cusparse<t>gtsv2, apart from the name.

Similarly as before, cusparse<t>gtsv2_nopivot requires an input buffer whose size is determined by cusparse<t>gtsv2_nopivot_bufferSizeExt.

Other useful routines to solve tridiagonal linear systems

cusparse<t>gtsvStridedBatch computes the solution of multiple tridiagonal linear systems with multiple system matrices and right-hand sides. It does not perform pivoting and uses dynamic temporary internal storage.

cusparse<t>gtsv2StridedBatch is the same as cusparse<t>gtsvStridedBatch, but it uses static external storage whose size is determined by cusparse<t>gtsv2StridedBatch_bufferSizeExt. It does not perform any pivoting. It achieves better performance when m is a power of 2.

Summary

Routine name	Tridiagonal system	Pivoting	Storage	Auxiliary routine
cusparse<t>gtsv	Multiple rhs	Yes	Internal	/
cusparse<t>gtsv_nopivot	Multiple rhs	No	Internal	/
cusparse<t>gtsv2	Multiple rhs	Yes	External	gtsv2_bufferSizeExt
cusparse<t>gtsv2_nopivot	Multiple rhs	No	External	cusparse<t>gtsv2_nopivot_bufferSizeExt
cusparse<t>gtsvStridedBatch	Multiple system matrix and rhs	No	Internal	/
cusparse<t>gtsv2StridedBatch	Multiple system matrix and rhs	No	External	cusparse<t>gtsv2StridedBatch_bufferSizeExt

Summary - Timings

Tests done for the double precision case.

Timings taken on a GTX960 card (c.c. 5.2) and reported in ms.

The number of inner points has been chosen to be a power of 2 so that the size of the linear system is a power of 2 and the performance of some of the tested routines is optimized. The overall number of discretization points should take also into account the boundary points.

The reported Err (error) is a percentage root mean square error.

Routine name	512		1024		2048		4096		8192	
	Time	Err	Time	Err	Time	Err	Time	Err	Time	Err
cusparseDgtsv	0.36	4.8e-9	0.50	4.5e-9	0.40	4.5e-9	0.49	5.7e-9	0.55	9.8e-9
cusparseDgtsv_nopivot	0.19	4.8e-9	0.19	4.7e-9	0.18	4.5e-9	0.23	4.5e-9	0.25	4.7e-9
cusparseDgtsv2	0.23	4.8e-9	0.23	4.5e-9	0.24	4.5e-9	0.29	5.7e-9	0.41	9.8e-9
cusparseDgtsv2_nopivot	0.04	4.5e-9	0.05	4.7e-9	0.05	4.5e-9	0.06	4.5e-9	0.08	0.08