Project #3:
Matrix-free LSQR

Presented by Yuval Freund and Dominik Marino

Table of content

- Recap: Working plan
- What we have done
- LSQR implementation
- Why cuBLAS?
- Testing
- Problems
- Our next steps

Working Plan

Phase 0: Understanding of LSQR method

Phase 1: Programm and test of the LSQR method with a small matrix

Phase 2: Try and modify LSQR method for a matrix **A** that can not fully stored in the memory

What we have done:

- ***Solving the LSQR problem (Yuval):***
  - Look up for useful libraries
  - Implementing the LSQR problem in CUDA
  - Solving compilation & linking issues, debugging and verifying correctness

- ***Testing (Dominik):***
  - Find suitable and different test matrices and vectors
  - Implement the given LSMR library for any input data
  - Compare results from the CUDA implementing and the LSMR library

LSQR implementation

The algorithm in the article

Algorithm LSQR

(1) (Initialize.)

$$\beta_1 u_1 = b, \quad \alpha_1 v_1 = A^T u_1, \quad w_1 = v_1, \quad x_0 = 0,$$
$$\bar{\phi}_1 = \beta_1, \quad \bar{\rho}_1 = \alpha_1.$$

(2) For $i = 1, 2, 3, \ldots$ repeat steps 3–6.

(3) (Continue the bidiagonalization.)

(a) $\beta_{i+1} u_{i+1} = A v_i - \alpha_i u_i$

(b) $\alpha_{i+1} v_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i$.

(4) (Construct and apply next orthogonal transformation.)

(a) $\rho_i = (\bar{\rho}_i^2 + \beta_{i+1}^2)^{1/2}$

(b) $c_i = \bar{\rho}_i / \rho_i$

(c) $s_i = \beta_{i+1} / \rho_i$

(d) $\theta_{i+1} = s_i \alpha_{i+1}$

(e) $\bar{\rho}_{i+1} = -c_i \alpha_{i+1}$

(f) $\phi_i = c_i \bar{\phi}_i$

(g) $\bar{\phi}_{i+1} = s_i \bar{\phi}_i$.

ACM Transactions on Mathematical Software, Vol 8, No 1, March 1982

(5) (Update x, w.)

(a) $x_i = x_{i-1} + (\phi_i / \rho_i) w_i$

(b) $w_{i+1} = v_{i+1} - (\theta_{i+1} / \rho_i) w_i$.

(6) (Test for convergence.)

Pseudo-Code

```
//INIT
beta = norm(b);
u = b / beta;
v = At*u;
alpha = norm(v);
v = v/alpha;
w = v;
x = 0;
phi_hat = beta;
rho_hat = alpha;
while(not converged){
    //bidiagnolization
    u = A * v - alpha * u;
    beta = norm(u);
    u = u / beta;
    v = At * u - beta * v;
    alpha = norm(v);
    v = v / alpha;
    // orthogonal transformation
    rho = sqrt(rho_hat^2 + beta^2);
    c = rho_hat / rho;
    s = beta / rho;
    theta = s * alpha;
    rho_hat = -c * alpha;
    phi = c * phi_hat;
    phi_hat = s * phi_hat;
    //update next vectors
    x = x + (phi / rho) * w;
    w = v - (theta / rho) * w;
    residual = norm(A*x - b);
    checkForConvergnce();
}
```

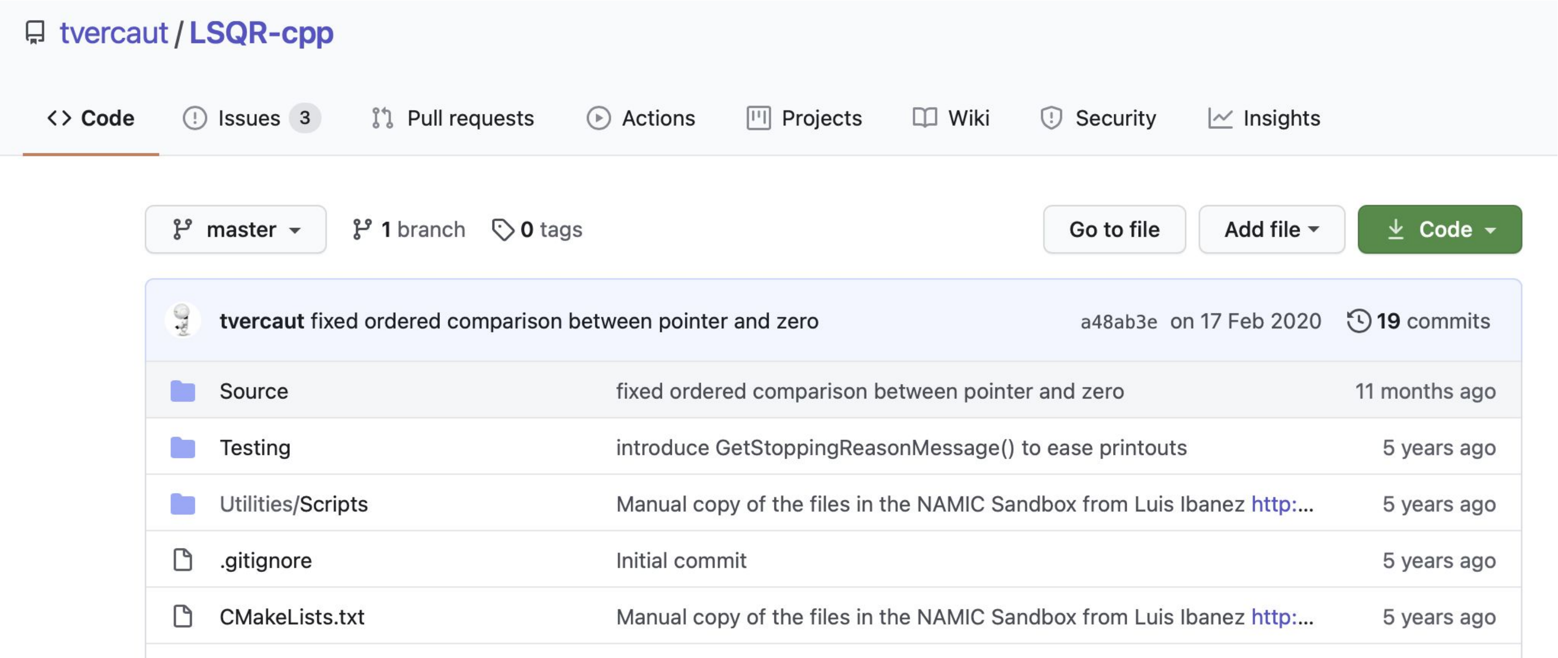
Implementation in cuBLAS

```
//Ax - b (result in tempVector)
tempDouble = -1.0;
tempDouble2 = 1.0;
status = cublasDgemv (handle, CUBL
cuBLASCheck(status,__LINE__);
status = cublasDnrm2(handle, tempV
cuBLASCheck(status,__LINE__);
improvment = prev_err-curr_err;
printf("line: %d size of error: %.
if(improvment<ebs) counter++; else
if(counter>1000) break;
prev_err = curr_err;
```

Why cuBLAS?

- Wide range of linear algebra functions (All the required Matrix & Vector operations)
- Tested and trusted library from NVIDIA
- Most of the functions have a cuBLAS equivalent (e.g. cublasDaxpy -> cusparseDaxpyi)
- Allows debugging while still working in dense format
- Requires no special data type or data structure to work with it
- Has both float and double precision function

Testing



https://github.com/tvercaut/LSQR-cpp

Testing

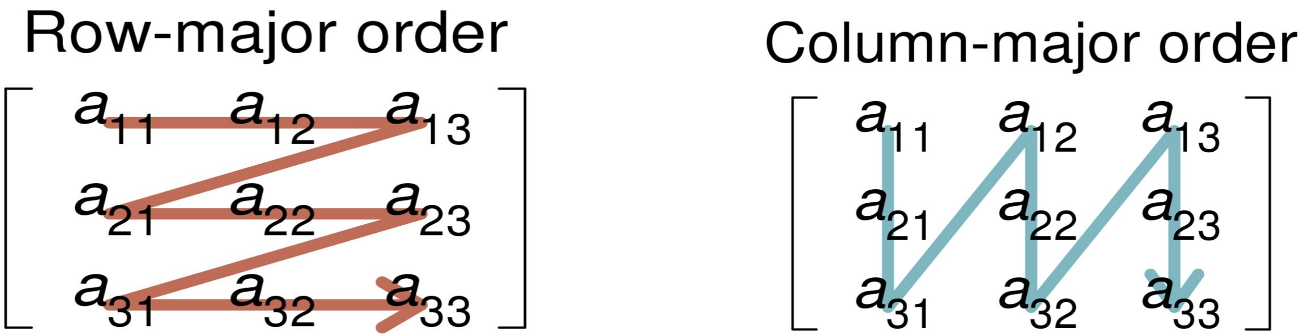
- *We are checking:*
  - The results
  - Number of iteration
  - Stopping reason
  - Norm of final value of residuals
  - Norm of final solution

```
std::cout << "Stopped because " << solver.GetStoppingReason() << ": " << solver.GetStoppingReasonMessage() << std::endl;
std::cout << "Used " << solver.GetNumberOfIterationsPerformed() << " Iterations" << std::endl;
std::cout << "Estimate of final value of norm of residuals = " << solver.GetFinalEstimateOfNormOfResiduals() << std::endl;
std::cout << "Estimate of norm of final solution = " << solver.GetFinalEstimateOfNormOfX() << std::endl;
```

- *How do we test?*
  - Predefined matrices and vectors in multiple sizes

Problems

- Different format that are used in different CUDA libraries



1.1. Data layout

For maximum compatibility with existing Fortran environments, the cuBLAS library uses column-major storage, and 1-based indexing. Since C and C++ use row-major storage, applications written in these languages can not use the native array semantics for two-dimensional arrays. Instead, macros or inline functions should be defined to implement matrices on top of one-dimensional arrays. For Fortran code ported to C in mechanical fashion, one may chose to retain 1-based indexing to avoid the need to transform loops. In this case, the array index of a matrix element in row “i” and column “j” can be computed via the following macro

```
#define IDX2F(i,j,ld) ((((j)-1)*(ld))+((i)-1))
```

https://en.wikipedia.org/wiki/Row-_and_column-major_order
https://docs.nvidia.com/cuda/cublas/index.html#data-layout

Problems

- Stopping conditions
- Float / double
- Linking problem
- It works for small matrix like 3x3 but not for a matrix 1000x1000
- Converging problems

Our next steps

- Write the implementation with the cuSPARSE library
- Build a parser for sparse matrices, to read from and write to files.
- Increase parallelization and memory usage of the algorithm to improve performance
- Test and verify our results

Do you have any questions?