CFD Lab Project: GPU Accelerated CFD Solver fluG

by Group F

Vaishali Ravishankar
Technical University of Munich

Rajat Arunachala Chandavar
Technical University of Munich

Sumukha Shridhar
Technical University of Munich

July 21, 2022

1 Abstract

FluG is a general purpose CFD solver in C++ which is accelerated by GPU and implemented using the CUDA API. The 2D, incompressible, Navier Stokes equation has been explicitly solved using Finite Difference method on a Cartesian Grid. The solver's results are validated using existing serial results from previous worksheets. The solver's performance is evaluated using NVIDIA GPU profiling tools. The primary goal for the project is to investigate the performance of GPU acceleration on the existing Fluidchen solver. The project has been run and tested on the SCCS GPU cluster which uses NVIDIA GeForce RTX 3080.

2 Implementation

The skeleton code from the second worksheet of the CFD Lab repository has been used for the current implementation of GPU accelerated code with the CUDA interface. A new header file CUDASolver.cuh containing the CUDASolver class and a corresponding CUDASolver.cu was added. These files contain the bulk of our implementation as they contain various functions and GPU kernels used for offloading tasks onto the GPU. Overall algorithm for the solver can be seen in figure 1.
The link to the project repository is **FluG**.

3 Pressure Solver: Red-Black SOR Scheme

The traditional SOR method cannot be parallelized on GPU due to data dependencies and hence it is implemented with the help of the Red-Black SOR (RB-SOR) Scheme which works for parallel

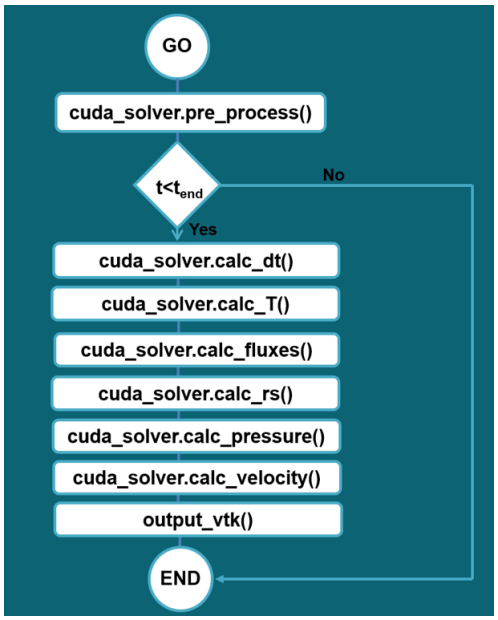

Figure 1: Algorithm implemented on GPU

executions. In this method, the red and black cells are present alternatively like a checkerboard pattern. The computations in the cells of the same colour can be handled independently of others as there is no more data dependency. Initially the Red cells get updated after which the black cells are updated. It is to be noted that the gamma parameter has to be set to a lower value in the vase of RB-SOR Scheme (gamma = 1.3 is preferred).

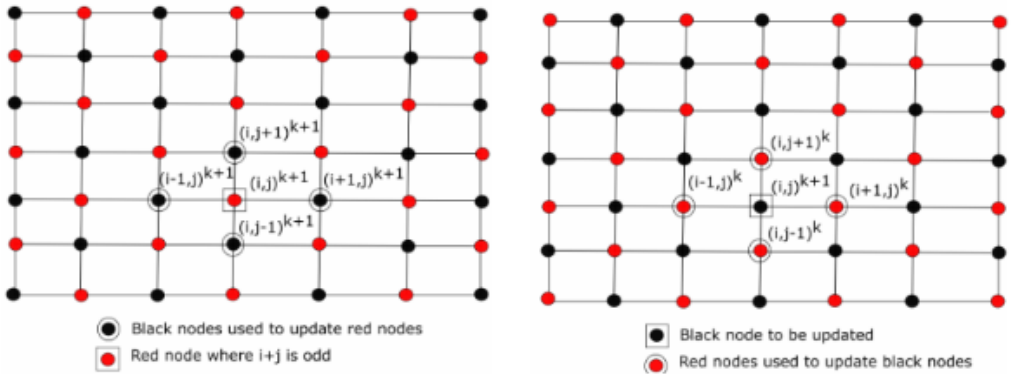

Figure 2: Red-Black scheme

4 Building the Project

4.1 Software Requirements

- CMake
- OpenMPI
- CUDA
- VTK

Each case file consists of a .dat file and .pgm file which will be read at runtime.

```
git clone https://gitlab.lrz.de/vaish/group-f-cfd-lab.git
cd group-f-cfd-lab
mkdir build
cd build
cmake ..
make
./fluG ../example_cases/<Case_Name>/<Case_Name.dat>
```

Before building and running the cases, make sure to remove the output directory from the previous run and remove the make files using

```
rm -r ../example_cases/<Case_Name>/<Case_Name>_Output
rm -rf *
```

Currently, both Single precision as well as Double precision computations are supported and this can be modified in the Datastructures.hpp file:

```
#define USING_SINGLE_PRECISION 0 //For double precision
#define USING_SINGLE_PRECISION 1 //For single precision
```

5 Validation

To ensure accurate results on the GPU accelerated code, the results of serial run and GPU run were compared for Channel flow. Velocity profile at the outflow was compared between the two and no noticeable difference was observed as seen in figure 3.

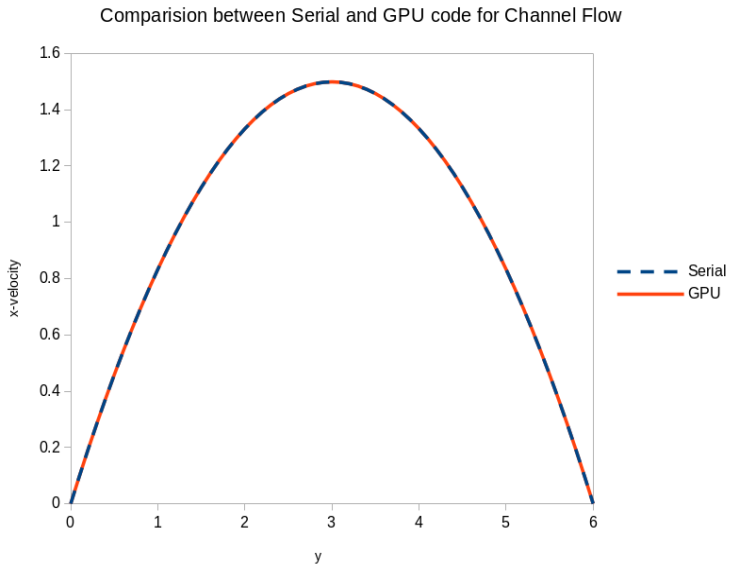

Figure 3: Validation of GPU solver with Channel Flow

6 Performance Study - Results and Observations

Performance study was done on Channel Flow and Fluid trap case by scaling the domain. For scaling factor of 3, run times of serial code, MPI code (using 16 processes) and GPU Code (in both single and double precision) is compared (In Figures 4 and 5). A speedup of around 35 - 36x was observed when serial code was compared with GPU single precision code. For scaling factor of 5, serial code was too slow and hence only the MPI and and GPU Code was compared (In 6 and 7). A speedup of around 8x and 3x respectively for Channel flow and fluid trap was observed when MPI implementation (with 16 processes) was compared with GPU single precision code.

7 Debugging and Profiling Tools

For debugging, CUDA-gdb and NVIDIA NSights Compute were used. The latter was also helpful for profiling and tracking memory allocations. NVIDIA Nsight Systems was useful for profiling the application. A snippet of a profiling session in Nsight Systems is found in figure 8.

8 Trial implementations which are not included in the project

A direct solver was tried to solve the sparse linear system resulting from discretization of Pressure Laplace equation. QR decomposition using CUSolver was tried, which resulted in slow run-times.

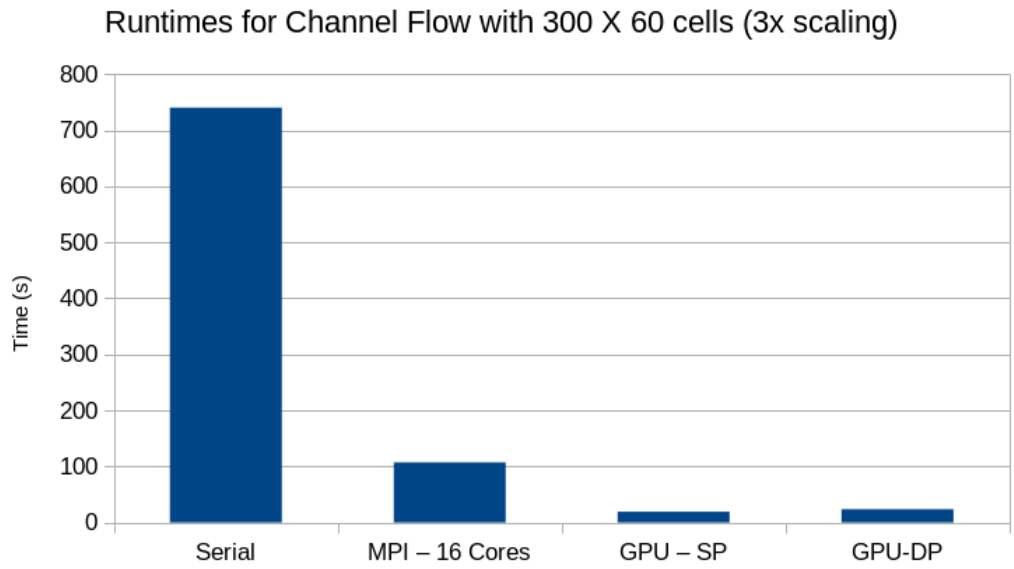

Figure 4: Comparison of Run-times for 3x scaled Channel Flow

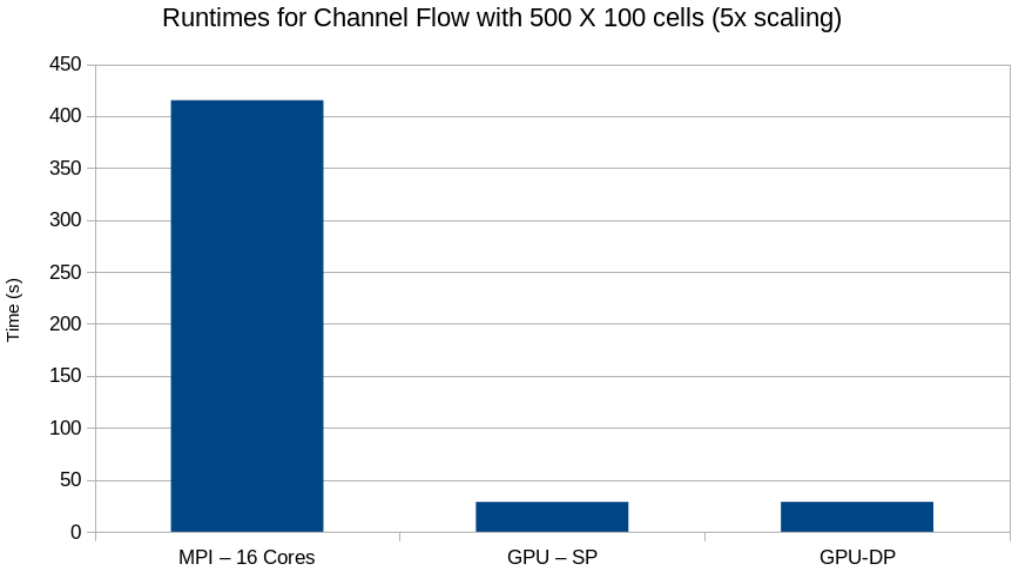

Figure 5: Comparison of Run-times for 5x scaled Channel Flow

Analysis on the NSights-Systems also showed that CUSolver was the bottleneck in each SOR iteration.

Thrust is a C++ template library for CUDA based on the Standard Template Library (STL).

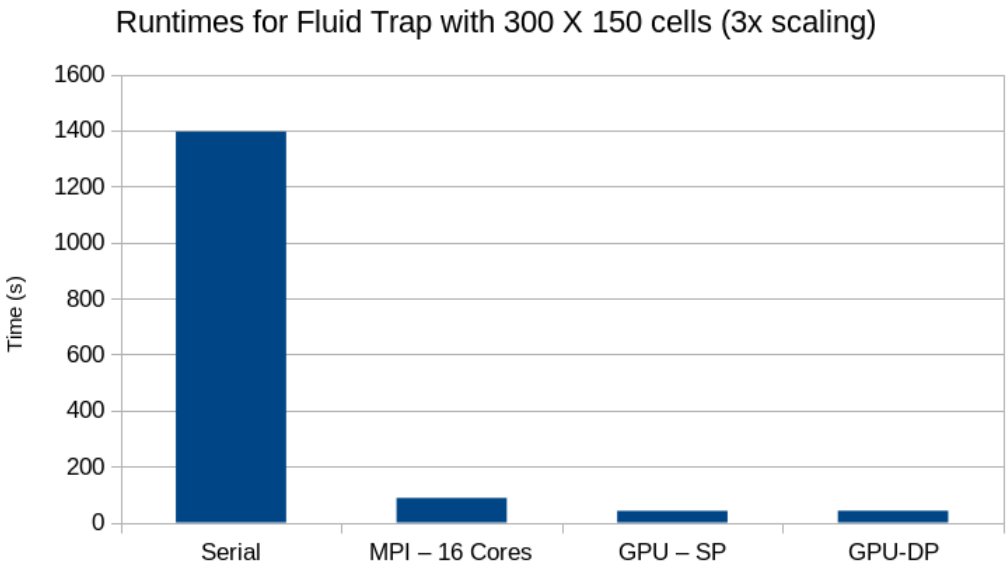

Figure 6: Comparison of Run-times for 3x scaled Fluid Trap

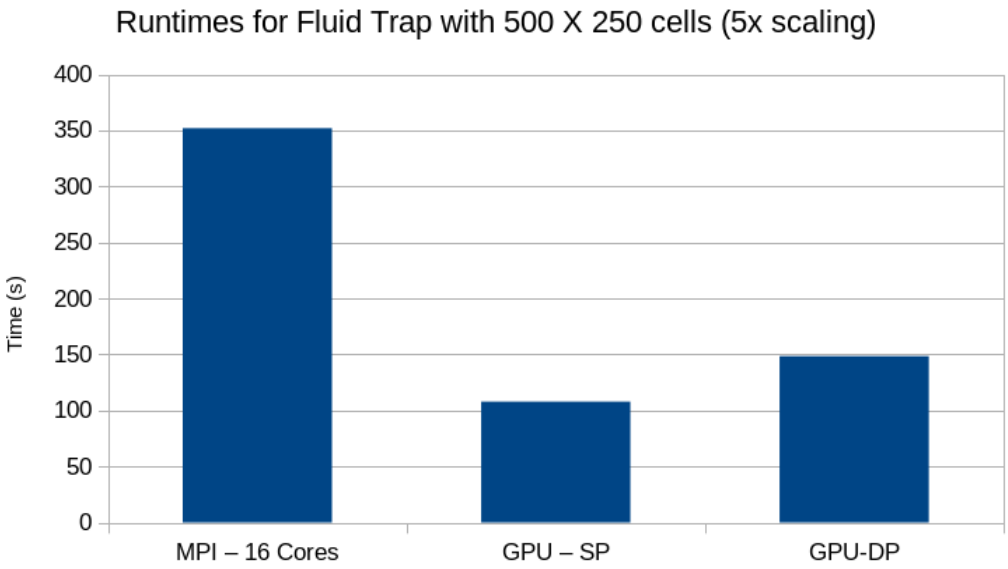

Figure 7: Comparison of Run-times for 5x scaled Fluid Trap

Reduction operations for residuals were done with thrust functions. Also, maximum element amongst x-velocity and y-velocity (for dt calculation) was done using thrust::maximum. However, it was observed that these were slower than the current implementation based on atomic methods.

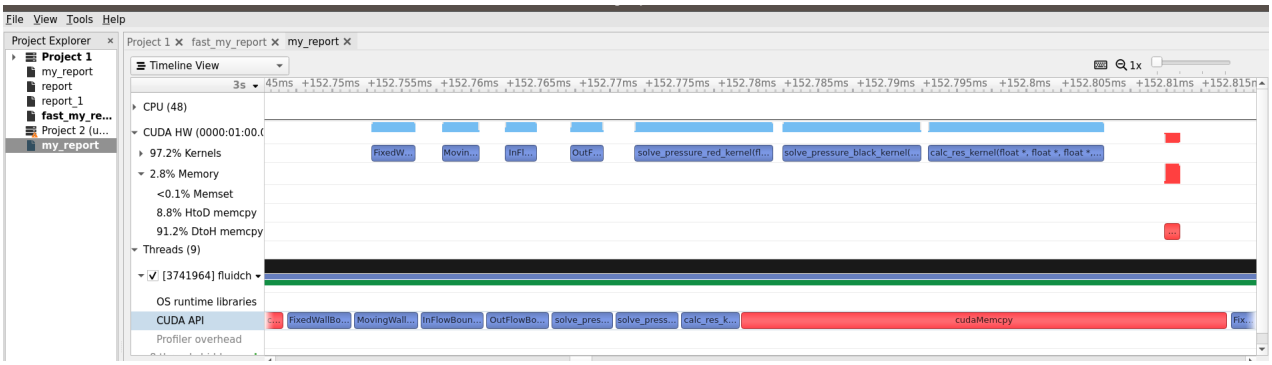

Figure 8: Profiling of a SOR iteration using Nsight Systems

Further, Jacobi Method (easy for parallelization) was also tried which resulted in higher number of iterations to converge/ high residual values and hence deemed not suitable for the solver.

References

[1] Dennis C. Jespersen. 2010. Acceleration of a CFD code with a GPU.

[2] E. Kandrot and J. Sanders. 2010. CUDA by Example.

[3] C.Shah, D. Majumdar, S. Sarkar. 2019. Performance Enhancement of an Immersed Boundary Method Based FSI Solver Using OpenMP