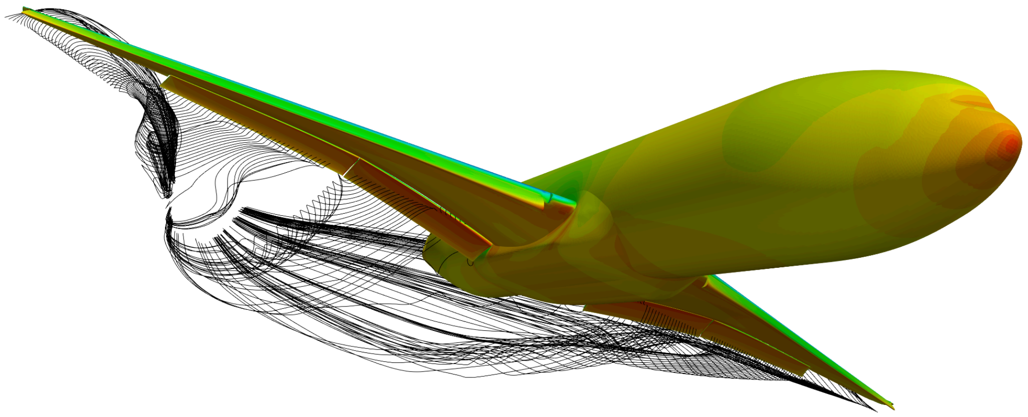
Google Summer of Code



Feasibility of GPU Acceleration in SU2

Contributor - Areen Raj
Mentors - Leonardo Cavanha, Ole Burghardt

Project Size - Medium (175 Hours)

24-03-2024

Stichting SU2
Faculty of Aerospace Engineering
Kluyverweg 1
Delft 2926HS
The Netherlands

SU2 foundation

About Me

Hello, I am Areen, currently in my fourth year of Mechanical Engineering at BITS Hyderabad, India. Physics has always interested me since I was in high school and I was thrilled to work in fluids during my education. I've been engrossed in Computational Fluid Dynamics ever since, playing around mostly with open-source codes like OpenFOAM. The concept of having access to such versatile codes was not only mind-blowing but powerful in showcasing what community coding could achieve. I am also proficient in C++, with an intermediate proficiency in Python. Allowing me the ability to code multiple computational solvers ranging from solving Navier Stokes to the Black Scholes equations. My experience also includes accelerating codes using CUDA and improving execution times. My OpenFOAM portfolio includes creating custom solvers and working on various projects from MHD to Turbulence.

On a more personal note, I come from the state of Bihar, the home of the famous Litti Chokha - a beautiful place in the northern parts of India. And despite all my endeavors in research and coding, my original love has always been writing.

Personal Details

Name - Areen Raj
Email ID - f20201965@hyderabad.bits-pilani.ac.in
Phone Number - +917702687452
Github - https://github.com/areenraj
LinkedIn - https://www.linkedin.com/in/areen-raj-179527186/

Synopsis

The following project involves the GPU acceleration of Linear Algebra Computations using NvBLAS in resource-intensive FEM solvers of the Open-Source Code - SU2. Higher-order methods are potentially more accurate for the same amount of mesh density but involve large matrix multiplications, which are currently handled by OpenBLAS. cuBLAS has been known to be much more effective in handling these operations compared to a serial CPU execution. This implementation will help in decreasing execution time and making FEM solvers feasible even for large-scale applications.

Benefits - Such an undertaking will allow SU2 to be seen as a viable option for higher-order problems - such as in-depth turbulent studies - and add to the growing range of open-source fluid codes exploiting GPU acceleration to improve performance. Fluid modeling has always been a resource-expensive task, but this will give smaller groups a chance to carry out more detailed simulations without the need for large-scale CPU clusters that are inaccessible to them - opening the door to greater participation in such work and removing barriers that previously existed due to monetary constraints.

Overview

SU2 is an open-source CFD code written mainly in C++ with standout features like easy integration of multiple physical phenomena, a versatile design optimization system, the powerful Python wrapper capable of granting fine-tuned precision, along with support for external CGNS meshes - just to name a few.

Written in a modular fashion, SU2 provides a straightforward case setup and the ability to modify it easily. It also hosts a variety of solvers for a number of applications, each of which are based on different methods tailored to provide a diverse approach.

However, two of the main solvers that are the primary workhorses are

- Euler and Navier Stokes solvers based on the Finite Volume Method
- Euler and Navier Stokes solvers based on the Finite Element Method

The FEM solver is based on the Discontinuous Galerkin Finite Element Method. While the FVM solvers are the most widely used, they are limited to second-order accuracy. In the case of many applications, a higher level of fidelity is required. Wake analysis, shock capturing, etc., require higher-order methods to resolve small fluctuations that may get ignored in lower-order methods. This is extremely important for analyzing turbulence - one of the most active areas of fluid research - to get the best overview of the system.

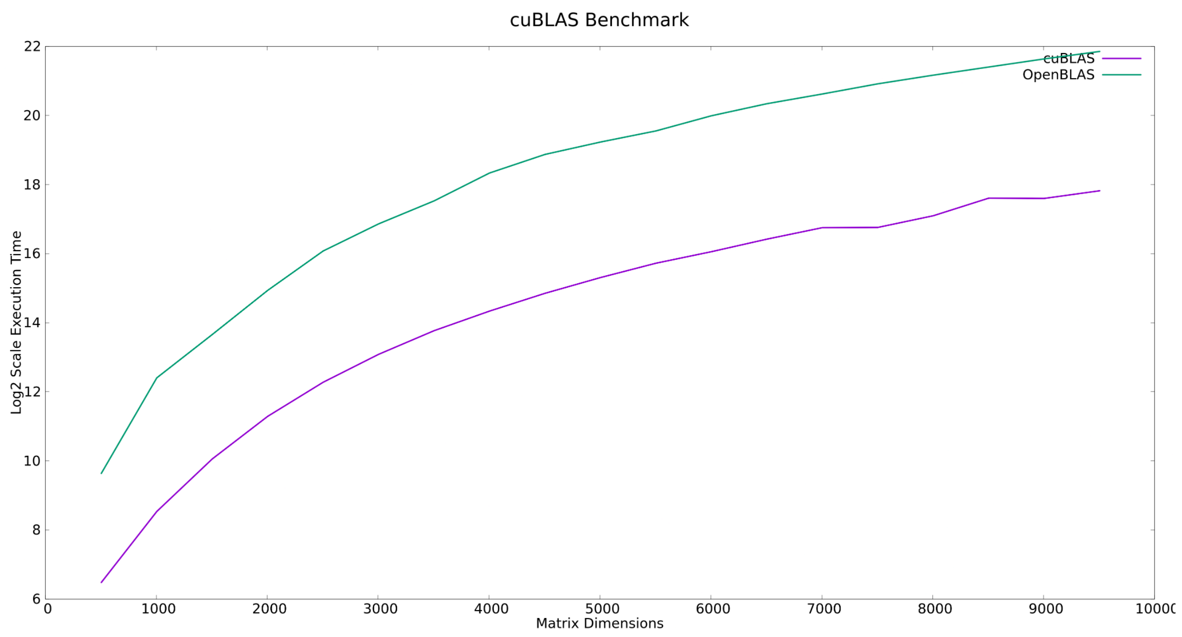
For these reasons, FEM was employed with support for higher-order discretization schemes. These methods are potentially more accurate than lower order ones, however this is subjective and depends on the implementation of the code.

So why not use FEM by default?

The answer lies in the computational cost. Higher-order FEM schemes require more expensive polynomial calculations for interpolation and discretization. These could possibly add up significantly and slow down the code when dealing with large-scale problems. These methods also involve heavy linear algebra calculations with matrix multiplication. They are slightly optimized by involving BLAS + LAPACK functions to speed up certain parts of the code.

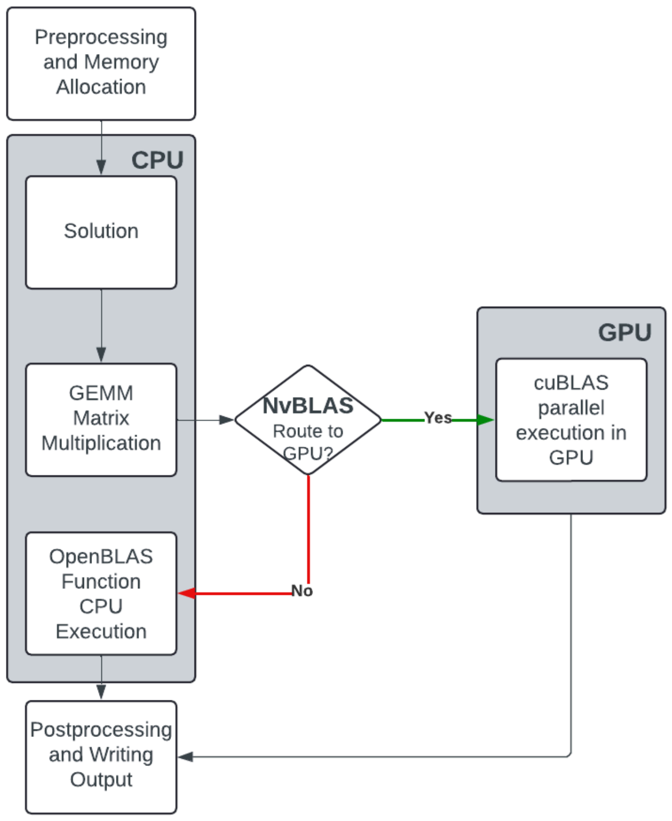
Currently the entire solver is CPU-based and incorporates OpenBLAS into it. While being extremely efficient, there are still ways to speed up this implementation with the addition of cuBLAS.

cuBLAS is a CUDA library focused on linear algebra subroutines but diverted toward GPU execution that make it orders of magnitude faster than OpenBLAS. A small computational study was done with increasing dimensions of square matrix multiplications, comparing the difference between cuBLAS and OpenBLAS. Both of them ran in cubic time with the OpenBLAS exponent $k = 2.98$ and that of cuBLAS $k = 2.75$. k being part of the expression - $O(n^k)$.



Such a performance uplift could be highly beneficial to the runtime of the code and make the FEM solver a very feasible option, even for larger cases. This can be achieved by either a raw implementation of cuBLAS functions into the existing code or through a more streamlined method, which is the integration of NvBLAS. NvBLAS automatically routes certain computationally intensive tasks from the CPU to the GPU to speed up the code without any extra changes done to it.

A flowchart view of the algorithm's workings is given below.


Preprocessing and Memory Allocation
CPU
Solution
GEMM Matrix Multiplication
NvBLAS Route to GPU?
Yes
No
GPU
cuBLAS parallel execution in GPU
OpenBLAS Function CPU Execution
Postprocessing and Writing Output

Milestones

The project can be broken down into a few major parts, which are as follows

1. Integration of NvBLAS into already existing code to route the matrix operations directly to the GPU.
2. Using NSight Compute to analyze code execution behavior and perform optimizations based on this data.
3. Improving memory allocation, distribution, and access further removing bottlenecks and improving performance.
4. Formulating a plan to extend the work further to incorporate these techniques beyond just the FEM solver.

Deliverables

The following deliverables can be expected out of the project.

1. **A pull request containing the GPU accelerated version with NvBLAS** - enabled into the FEM Euler and Navier Stokes solvers. This will be accompanied by proper documentation regarding the changes and guidance on running the code in GPU mode.

2. **A computational study and execution report** - compiled with the help of NSight, providing a look at the inner workings of how the code runs. Following this, a proposal will be put forth for the optimizations that can be made.

   This implementation will require deeper modification of the code and could potentially be time consuming. If completed, **a follow up pull request** of the optimized version will be initiated.

3. **An extension report** will be compiled which contains the information needed for future work to continue on other solvers either by integrating them with NvBLAS or modifying the source code itself to support GPU acceleration.

Color Coding Key - Green represents the deliverables that are promised unconditionally as a part of this project. Yellow indicates those that are time dependent and subject to the amount of modification required in the code to achieve the said goal.

Timeline

Weeks 1-2 (35 Hours)

NvBLAS can be integrated into the code in a straightforward manner. This week will focus on reviewing the available documentation and adding it into the existing code.

Weeks 3-4 (35 Hours)

Every pull request also needs to pass a variety of test cases before being merged. This will be the second focus - to document and run these benchmark simulations off it properly. Compiling a user guide and accompanying reports will also be a part of this time period.

Weeks 5-6 (35 Hours)

This period will be dedicated to optimizing the code that was created in the previous weeks. It will involve using NSight Compute to gain knowledge of the bottlenecks and memory transfers between the CPU and GPU. These transfers contribute to the execution time and it is essential to optimize them. The code will be improved depending on the analysis. A report will also be created to record the results of this entire endeavor

---

The Mid-Term Evaluation is scheduled after Week 6

Weeks 7-8 (35 Hours)

Code optimization will continue as such integrations are bound to be time-consuming, with each solver requiring custom implementations while modifying it. We will move on to explore larger test cases that will solidify the intended use of higher-order methods. These test cases will involve testing the Euler solver with turbulent setups employing Reynolds Averaged Navier Stokes methods. The criteria for benchmarking will be comparing the accuracy and wall time of the simulation with the CPU code.

Weeks 9-11 (35 Hours)

Avenues will be explored to introduce this form of acceleration to other solvers. Involving editing the source code to make it more apt for GPU integration. If time does not permit for these changes, then a report will be compiled on how to move forward, and an extension may be submitted to continue forward with.

Week 12 (15 Hours)

A spare week is given as a buffer to account for delays or any breaks that might be taken in between.

Closing Comments

My motivation for this project came from previous CUDA work that I have done with accelerating LBM solvers. The duration of GSoC is not the only time period that I would like to work on this. Hence, the inclusion of the extension report that carries the prospect of future work.

As of writing this proposal, I have been in contact with the SU2 team for a month now and have gotten to know a group of really helpful and passionate individuals. I would be extremely grateful to work with them. I would also like to contribute to other projects, namely those associated with the Python wrapper. I wholeheartedly believe that this will be both an amazing opportunity and an enriching experience for me.

Thank You.

References

I would like to thank my mentor - Leonardo Cavanha - for providing me with the reference material for this project. He has been an amazing and kind guide who is ever ready to help and lead me.

1. FEM computational cost reference - https://su2code.github.io/documents/su2_dev_vanderweide.pdf
2. CFD solvers relying on GPU acceleration
    a. https://www.ansys.com/blog/unleashing-the-full-power-of-gpus-for-ansys-fluent
    b. https://ntrs.nasa.gov/citations/20140003093
    c. https://exafoam.eu/wp3/
3. NvBLAS Docs - https://docs.nvidia.com/cuda/NvBLAS/