

OpenFOAM on GPUs

Thilina Rathnayake

158034R

Thesis/Dissertation submitted in partial fulfillment of the requirements for the
degree Master of Science in Computer Science and Engineering



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

Department of Computer Science & Engineering

University of Moratuwa

Sri Lanka

July 2016

DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature:

Date:

 The above candidate has carried out research for the Masters thesis/Dissertation under my supervision.
University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

Signature of the Supervisor:

Date:

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to Prof. Sanath Jayasena for supporting me through out my research work. I am especially thankful for committing his valuable time to help me tackle down the problems I faced during my research. I had a very slow start at the beginning of my research and I was hardly making any progress. Nothing was working for me at the time. Prof. Jayasena's words of encouragement and acts of support were a huge strength for me to stay focused and achieve the goals of my research. I also gained a lot of experiences and knowledge through the weekly research meetings I had with him. It was a real pleasure to brainstorm as a group and come up with things to try out. Without him, my research wouldn't have become such a success.

I like to thank Dr. Mahinsasa Narayana of the Department of Chemical and Process Engineering of University of Moratuwa for providing an interesting problem for me to work on. I thankfully acknowledge the support he gave me at the beginning of my research to get acquainted with OpenFOAM. My sincere thanks goes to Mr. Niranjani Fernando for helping me on various domain specific problems I came across during the research.


My sincere thanks goes to the Department Computer Science and Engineering of the University of Moratuwa for providing me with the necessary resources throughout my research. I also like to thank Lanka Software Foundation for giving their nice office space for our use.

I thank the Senate Research Committee of the University of Moratuwa for supporting this research under a Senate Research Grant Award. I also thank the LK Domain Registry for supporting this research through Prof. V.K. Samaranayake grant.

ABSTRACT

Open source Field Operation and Manipulation (OpenFOAM) is a free, Open-source, feature rich Computational Fluid Dynamics (CFD) software that is used to solve a variety of problems in continuum mechanics. It has a large user base spread across various science and engineering disciplines and used in both academic and commercial contexts.

Depending on the type of problem and required accuracy, an OpenFOAM simulation may take several weeks to complete. OpenFOAM simulations generally involve preprocessing, discretization, applying linear solvers and post processing. For sufficiently large simulations, linear solvers contribute to a large portion of the execution time. Hence, Graphics Processing Units (GPU) based linear solvers can give a significant speedup compared to the native CPU implementation.

 AmgX is a state of the art, high performance library which provides an elegant way to accelerate linear solvers on GPUs. AmgX library provides various flavors of multi-grid solvers, Krylov methods, smoothers, support for block systems and support for MPI. It also provides a flexible way to use nested solvers, smoothers and preconditioners.

University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

In this work, we implemented OpenFOAM solvers on GPUs using AmgX library and a set of helper functions which enables seamless integration of these solvers to OpenFOAM. These will take care of converting the linear system to AmgX's format and apply the user specified configurations to solve it. Experiments carried out using a wind rotor simulation and a Fan wing simulations shows that the use of AmgX library gives upto 10% speedup in the total simulation time and **2x** speedup in solving the linear system.

Keywords: OpenFOAM; GPUs; AmgX; CFD;

TABLE OF CONTENTS

Declaration of the Candidate & Supervisor	i
Acknowledgement	ii
Abstract	iii
Table of Contents	iv
List of Figures	vi
List of Tables	vii
List of Abbreviations	1
1 Introduction	2
1.1 Computational Fluid Dynamics (CFD)	2
1.2 Software for CFD	3
1.3 Problem Statement	4
1.4 Objectives	5
1.5 Contributions	5
1.6 Organization	6
2 Literature Survey	7
2.1 OpenFOAM	7
2.1.1 General Flow of a OpenFOAM Simulation	8
2.1.2 OpenFOAM Linear Solvers	9
2.2 General CFD on GPUs	10
2.3 OpenFOAM on GPUs	11
2.4 Existing Software for porting OpenFOAM to GPUs	14
2.4.1 SpeedIT	14
2.4.2 ofgpu	14
2.4.3 Cufflink	14
2.4.4 RapidCFD	15
2.4.5 Paralution	15
2.5 NVIDIA AmgX	16

3	Methodology	17
3.1	Profiling Data	17
3.2	AmgX Wrapper Library for OpenFOAM	18
3.2.1	Data Structure Conversion	19
3.2.2	MPI Support	20
3.2.3	Multiple solver support	22
3.2.4	AmgX wrapper with RapidCFD	23
3.2.5	AmgX wrapper API	23
4	Experimental Results	25
5	Conclusions and Recommendations	29
	References	30



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

LIST OF FIGURES

Figure 1.3.1	Geometry of the simulation as seen from positive X-axis	5
Figure 2.1.1	Overview of OpenFOAM structure (source: OpenFOAM website)	8
Figure 2.1.2	OpenFOAM's complete linear solver structure	10
Figure 3.1.1	Top 15 Hot-spots by accumulated overhead of the method	17
Figure 3.1.2	Top 10 Hot-spots by self contributed overhead of the method	18
Figure 3.2.3	AmgX Wrapper Library	19
Figure 3.2.4	Conversion of lduMatrix to CSR Format	20
Figure 3.2.5	GPU Cluster with three nodes	21
Figure 3.2.6	Global communicator split into local in-node communicators	21
Figure 3.2.7	Local in-node communicators split by device	22
Figure 3.2.8	OFAmgX API Flow	24
Figure 4.0.1	Performance of windLM simulation in A	27
Figure 4.0.2	Performance of windLM simulation in B	28



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

LIST OF TABLES

Table 1.1	Free CFD software packages	3
Table 1.2	Commercial CFD software packages	4
Table 2.1	OpenFOAM linear solvers	10
Table 4.1	Different Experimental Environments	26
Table 4.2	Different Simulations used in the benchmarks	26
Table 4.3	Total simulation time for windLM without MPI (in seconds)	26
Table 4.4	Total simulation time for windLM with 2 MPI processes (in seconds)	26
Table 4.5	Total simulation time for windLM with 4 MPI processes (in seconds)	26
Table 4.6	Total simulation time for windLM with 8 MPI processes (in seconds)	26
Table 4.7	Total simulation time for FanWing2D without MPI (in seconds)	26



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

LIST OF ABBREVIATIONS

OpenFOAM	Open source Field Operation and Manipulation
CFD	Computational Fluid Dynamics
GPU	Graphic Processing Unit
HPC	High Performance Computing
GPGPU	General-purpose computing on graphics processing units
PDE	Partial Differential Equation
MPI	Message Passing Interface
RC	RapidCFD



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

Chapter 1

INTRODUCTION

This chapter gives a brief introduction to CFD and currently available software for solving CFD problems. It also briefly describes the problem we intend to address, objectives we hope to fulfill while solving the problem, contributions made by the thesis work and overall thesis organization.

1.1 Computational Fluid Dynamics (CFD)

Scientific Computing has become a very important part of modern research. It is used in a wide variety of science and engineering disciplines from Fluid Dynamics, Acoustics, Solid Mechanics to Electro-magnetics and many others. Often scientists and engineers carry out simulations in these areas to model and better understand important phenomena like wind patterns around a fan or temperature variation of a gas. These simulations can save a whole lot of time, money and often gives more flexibility than actually carrying out them in the real world.

CFD is one of the most important subfields in scientific computing. It is used to model various phenomena in aerodynamics, meteorology, chemical engineering, medicine, etc. CFD is essentially a combination of fluid dynamics, numerical methods and computer science where computers are used to run numerical algorithms to solve a fluid dynamics problem. Using computers to solve fluid dynamics problems has become very popular in past decades. Two major reasons behind this popularity are the availability of high performance computers and the accuracy of existing numerical methods to solve these problems.

OpenFOAM	OpenFOAM is a general purpose open-source CFD code written in C++ and uses an object oriented approach which makes it easy to extend
SU^2	The Stanford University Unstructured (SU^2) suite is an open-source collection of software tools written in C++ for solving PDE's and performing optimization problems
PyFR	PyFR is a Python based framework using Flux Reconstruction and Explicit Runge-Kutta time integration
Code_Saturn	A quickly developing code from EDF with full source code access
FEniCS	An open-source package for computational mathematical modeling
OpenFVM	A free CFD solver distributed under GPL
ISAAC	A compressible Euler/Navier-Stokes code written in F77

Table 1.1: Free CFD software packages

1.2 Software for CFD

High Performance Computing (HPC) platforms has seen a dramatic improvement over the last decades. These platforms can operate in petascale level and now reaching for the Exascale [1]. Modern day computers are equipped with multiple cores and often these nodes have Graphics Processing Units (GPUs) that act as co-processors or accelerators. These changes in the HPC hardware require changes in the software packages used in CFD to fully utilize them.

Several decades ago, CFD codes were written using languages like **FORTRAN** which was especially designed for the use of scientific community. Over the years, CFD community has shifted into more high level languages like **C**, **C++** and even **Python**. There are a number of software packages used in CFD. These software packages can be divided into two broad classes: **Free** and **Commercial**. Table 1.1 includes a list of free software and the Table 1.2 includes a list of commercial CFD software packages.

Open source Field Operation and Manipulation (**OpenFOAM**) [2] library is a widely used free CFD software package in both academia and industry. Open-

ADINA	Finite element software for structural, fluid, heat transfer, electromagnetic, and multi-physics problem
Autodesk Simulation	Finite Element software of Autodesk
ANSYS	US-based and -developed full CAE software package
COMSOL	COMSOL Multiphysics Finite Element Analysis Software
SimScale	100% web-based CAE platform

Table 1.2: Commercial CFD software packages

FOAM is written in C++ and can be used to solve partial differential equations (PDEs). OpenFOAM can be used in all three phases of of a simulation: pre-processing, solving and post-processing. OpenFOAM contains various utilities and meshing tools for pre-processing and visualization software like **ParaView** for post-processing. Also, it has a wide range of CFD solvers like Laplace, Poisson, incompressible flow, etc. OpenFOAM comes with built in MPI functionality which allows users to decompose a given mesh into multiple chunks and use multiple nodes (or a single node with multiple cores) to process each chunk in parallel.


University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

1.3 Problem Statement

The main problem addressed in this project was to speed up a OpenFOAM simulation of a wind turbine using GPUs. This simulation has a very intricate design and has around 1166000 mesh points. This simulation takes weeks to complete (for the required accuracy and time step) in a multi-core CPU setup. Geometry of the wind turbine is shown in Figure 1.3.1 shows geometry of the simulation as seen from positive X-axis.

Recently GPUs have been used to speed up various computationally intensive tasks like audio/video signal processing, medical imaging, deep learning, etc. (Originally, the GPUs were only used for computer graphics). These various general computations done using GPUs are known as **GPGPU**: *General-purpose computing on graphics processing units*. GPUs have been widely used in CFD problems as well. Especially, CFD solvers have been successfully implemented in GPUs and generally give a huge speed up compared to their CPU counterparts.

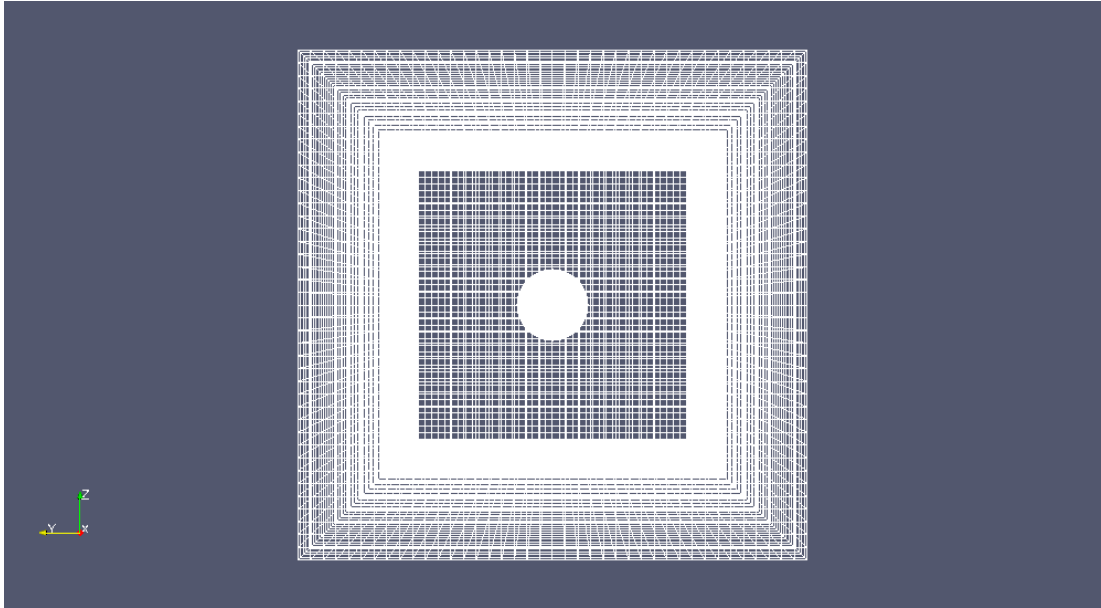


Figure 1.3.1: Geometry of the simulation as seen from positive X-axis

1.4 Objectives

Objectives of this research are to survey the existing methods for porting OpenFOAM simulations to GPUs and see how the wind turbine simulation can be sped up using the GPUs.



University of Moratuwa, Sri Lanka
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

1.5 Contributions

We ported OpenFOAM's CFD solvers into GPU using NVIDIA's AmgX library. We created a wrapper library that enables easy usage of AmgX solvers from OpenFOAM.

Our contributions in the thesis are:

- Created a wrapper library that enables OpenFOAM user's to run their simulations in GPUs using the NVIDIA AmgX library.
- Enabled Multiple instances of AmgX solvers to be used from OpenFOAM.
- Enabled the MPI support through the wrapper so that multiple GPUs can be used to run the simulation.

- Created an AmgX wrapper for **RapidCFD**: A GPU based version of OpenFOAM.

1.6 Organization

This thesis is organized as follows. In chapter 2, structure of OpenFOAM and the related work in the area are discussed. Methodology we followed in the research has been extensively described in chapter 3. Experiments we carried out with AmgX wrapper library are presented in chapter 4. Finally, the conclusions and recommendations are presented in chapter 5.



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

Chapter 2

Literature Survey

In this chapter, we are going to have a deeper look at OpenFOAM and survey current methodologies that have been used to port general CFD codes and OpenFOAM simulations to GPUs. We are also looking at existing software solutions for porting OpenFOAM functionalities into GPUs. At last, we are introducing AmgX; NVIDIA's GPU based linear solver library which we used with OpenFOAM to speed up the simulations.

2.1 OpenFOAM

OpenFOAM is written in C++ and heavily uses object oriented features in C++ to build the framework required for simulations. Primary use of OpenFOAM is to create executables known as applications. These applications can be broadly categorized into two categories: solvers and utilities. Solvers are created to solve a specific problem in continuum mechanics like calculating pressure and velocities of an incompressible flow flowing through a specific tube geometry. Utilities are designed to perform tasks that involve data manipulation. Users can create custom solvers and utilities by using OpenFOAM with some knowledge about underlying CFD algorithms, physics and programming techniques.

OpenFOAM ships with pre and post processing tools. OpenFOAM utilities have been written on top of these tools to enable users to easily access them. Thus, the interface to these pre and post processing tools are consistent even though underlying tool environments can change. Overall structure of OpenFOAM is shown in Figure 2.1.1.

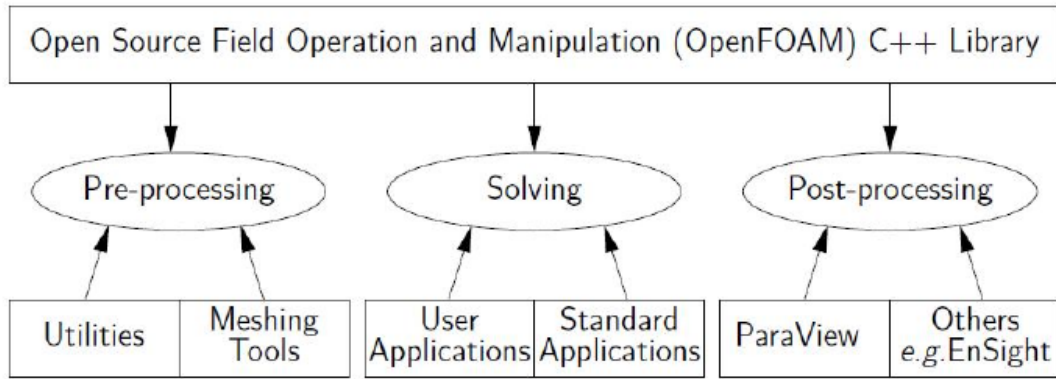


Figure 2.1.1: Overview of OpenFOAM structure (source: OpenFOAM website)

2.1.1 General Flow of a OpenFOAM Simulation

Preprocessing tools are responsible for converting the input of a flow problem to a format that can be usable by the solver. Normally, input of a flow problem consists of a specification of the computational domain (mesh), the equations to be solved in the domain (e.g., Navier Stokes equation) and the boundary and initial conditions of the problem.



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mru.ac.lk

OpenFOAM provides a mesh generation tool called *blockMesh* which generates the mesh based on a configuration file called *blockMeshDict*. This file is created by the user in the format of a dictionary. The user can use various keywords and values to specify the mesh geometry or the computational domain of the problem.

A mesh is described using four building blocks: *points*, *faces*, *cells* and *boundary*. Point is simply a vector in 2D or 3D space. All the points in the mesh are compiled into a list and each point has a label which refers to the position of the point in the list. A face is made up of these points as an ordered list of corresponding labels. Similar to points, faces are also compiled into a list and each face has a label that corresponds to its position in the list. Cell is a collection of faces represented by corresponding face labels. Patch consists only of boundary faces.

The numerical methods on this computational domain are done using three steps. First step is to approximate the flow variable. Flow variable can be any physical quantity related to the flow like pressure, velocity, etc. At the very beginning of the simulation, initial and boundary values are used for the approximation. The second step is the application of a discretization method to estimate the PDEs. Discretization step is carried out in both the computational domain and in the PDEs which the solver is going to solve. Third step is the solution of the algebraic equations which resulted from the discretization. There are various discretization schemes available and OpenFOAM uses the finite volume discretization.

After the discretization procedures, various numerical algorithms can be run on the computational domain. Most of the times, iterative solution procedures are used. In our work, we worked on the *pimpleDymFoam* solver from the OpenFOAM solver set which uses **PIMPLE** algorithm. Our wrapper library can be used with any OpenFOAM solver, either default or user made. High level algorithms like PIMPLE define how the solution procedure should be carried out. For example, in the PIMPLE algorithm, momentum and pressure equations may be solved twice in a single time step.

2.1.2 OpenFOAM Linear Solvers

OpenFOAM has several methods (algorithms) to solve the linear system resulting after the finite volume discretization. Algorithm selection depends on the resulting linear system (symmetric, asymmetric), initial and boundary conditions and the convergence characteristics of the matrix. Table 2.1 shows the solvers available in OpenFOAM.

Different types of preconditioners and smoothers are used in OpenFOAM to make the solution process of linear system more efficient. Preconditioners transform the linear system so that the transformed system converges much faster than the original. Smoothers on the other hand reduce the mesh dependency of

BICCG	Diagonal incomplete LU preconditioned BiCG solver
diagonalSolver	diagonal solver for both symmetric and asymmetric problems
GAMG	Geometric agglomerated algebraic multi-grid solver (also named Generalized geometric- algebraic multi-grid)
ICC	Incomplete Cholesky preconditioned Conjugate Gradients solver
PBiCG	Preconditioned bi-conjugate gradient solver for asymmetric lduMatrices using a run- time selectable preconditioner
PCG	Preconditioned conjugate gradient solver for symmetric ldu- Matrices using a run-time selectable preconditioner
smoothSolver	Iterative solver using smoother for symmetric and asymmetric matrices which uses a run-time selected smoother

Table 2.1: OpenFOAM linear solvers

the numbers of iterations. Figure 2.1.2 shows the complete structure of linear solvers available in OpenFOAM along with preconditioners and smoothers.

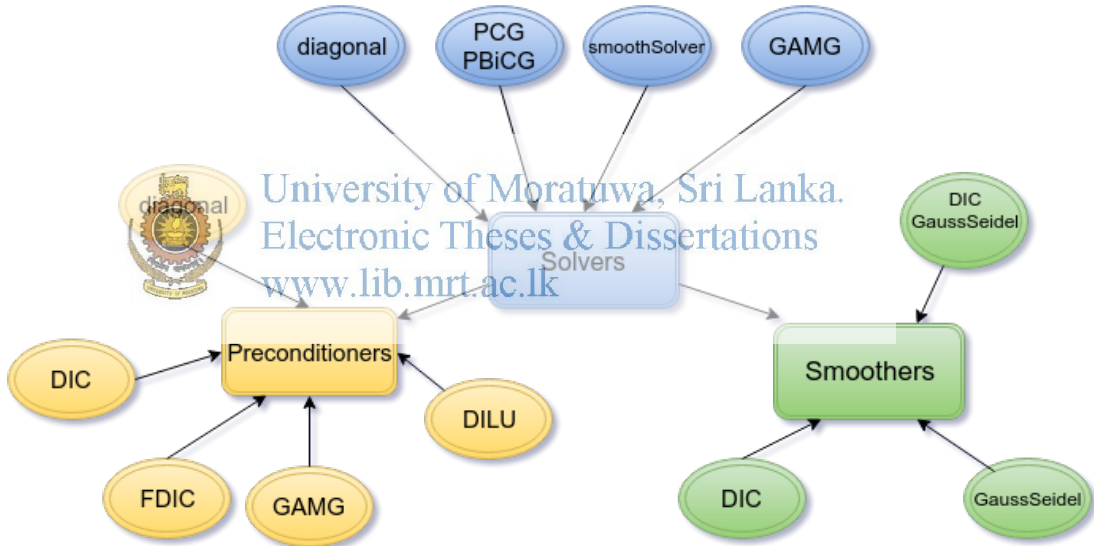


Figure 2.1.2: OpenFOAM's complete linear solver structure

2.2 General CFD on GPUs

There have been various attempts made to harness the power of GPUs in CFD. Some of the CFD codes involve performing a homogeneous set of operations over a large set of nodes in a mesh. These types of simulations can exploit SIMD type parallelism using GPUs.

Cohen et al. [3] describe a second-order double precision finite volume Boussinesq code implemented using the CUDA platform. They compared and validated their results with a Fortran code running on a high-end eight-core CPU. The CUDA-accelerated code achieved approximately an eight-time speedup versus the Fortran code on identical problems. They explored a number of GPU-specific optimization strategies focused on optimizing memory access patterns to take advantage of their GPU's streaming memory architecture. They have used a technique called congruent padding which amortizes the index to memory location translation cost between grids.

Thibault and Senocak [4] have implemented a Navier-Stokes solver for incompressible fluid flow using desktop platforms equipped with multi-GPUs. They have used NVIDIA's CUDA programming model to implement the discretized form of the governing equations. The projection algorithm to solve the incompressible fluid flow equations is divided into distinct CUDA kernels, and a unique implementation that exploits the memory hierarchy of the CUDA programming model is also suggested. They could gain a two orders of magnitude speedup relative to a serial CPU implementation.

2.3 OpenFOAM on GPUs

More complex simulations created with CFD software packages like OpenFOAM use different approaches to parallelize CFD codes. Since OpenFOAM is widely used in both academia and industry, there are various attempts made to use OpenFOAM with GPUs.

Amaniz AlOnazi et al. [1] have tried to design and optimize OpenFOAM based CFD applications to hybrid heterogeneous HPC platforms. Although OpenFOAM supports MPI natively, it doesn't scale well for heterogeneous sys-

tems [5]. Authors have extensively studied Conjugate Gradient (CG) method (which is a Krylov subspace solver) and identified the bottlenecks when it is run in a distributed memory system using MPI. Authors have proposed two major improvements to the existing CG solver.

First the authors have implemented a hybrid conjugate gradient using MPI and CUDA. The interesting aspect of their implementation is the addition of a load balancing step during the heterogeneous decomposition which decomposes the computing domain taking into account the performance of each computing device and minimizing communication. This heterogeneous decomposition method consists of two steps: building the accurate performance model of the application using the approach of FuPerMod [6, 7, 8, 9], and then using this performance model as input to MeTiS [10] /SCOTCH [11] libraries.

Secondly, authors have introduced an algorithmic improvement to the existing CG solver. They have implemented the pipelined conjugate gradient algorithm of Ghysels and Van Roose [12] which can be considered as a communication startup-reducing algorithmic improvement of the original CG method. The algorithm modifies and reorders the s-step CG method, which is introduced by Chronopoulos and Gear in their original paper [13] and minimizes the global communication to only one collective communication per loop body instead of three.

Qingyun He et al. [14] have used wide variety of existing libraries to speed up OpenFOAM solvers. Authors have implemented the pressure-implicit with splitting of operators (PISO) magnetohydrodynamics MHD solver on Kepler-class graphics processing units (GPUs) using the CUDA technology. Authors observed that a GPU (GTX 770) can outperform a server-class 4-core, 8-thread CPU (Intel Core i7-4770k). Author's have used the following libraries for CFD acceleration of the MHD solver:

1. CUDA for OpenFOAM Link (Cufflink) [15], an open source library for GPU acceleration in OpenFOAM. It supports single and double precision.
2. SpeedIT Plugin [16] to OpenFOAM by Vratiss released for demonstration purposes for GPU acceleration. The free version supports only single precision.
3. GPU linear solvers library for OpenFOAM (ofgpu) [17] by Symscape under GPL license. It supports only single precision.

Authors accelerated the MHD solver by only replacing its linear system solvers since solving matrices occupy most of program running time. Also, authors replaced sparse matrix vector production (SMVP) kernels with the corresponding GPU implementations. The vector–vector scalar product is calculated using the NVIDIA CUBLAS [18] library. Authors were able to get a 4x speedup for the benchmarks using a single GPU.

Jamshidi and Khunjush [19] have used the CUSPARSE [20] and CUBLAS [18] libraries to implement some of the OpenFOAM solvers. Author's have identified that the main computational intensive step in OpenFOAM solvers is the solving systems of linear equations. Among these solvers, Preconditioned Bi-conjugate Gradient (PBiCG), Preconditioned Conjugate Gradient (PCG) and Diagonal Solver are the linear solvers that are widely used.

Authors have tested their implementations in three different multi-core platforms including the Cell Broadband Engine, NVIDIA GPU, and an Intel quad-core Xeon CPU. According to the achieved results, the GPU implementations achieve the best performances due to having large number of threads. The maximum observed speedups are 15.5x and 345x for CG and Diagonal Solver, respectively.

2.4 Existing Software for porting OpenFOAM to GPUs

There are a handful of software solutions which attempt to port OpenFOAM simulation to GPUs.

2.4.1 SpeedIT

SpeedIT provides GPU-accelerated iterative solvers for large systems of sparse linear equations. The performance of the library was tested on matrices from the University of Florida Sparse Matrix Collection and compared against CUSPARSE and CUSP. According to the website, in average *SpeedIT* outperforms CUSPARSE and CUSP by a factor of x2 and x4, respectively [16]. There is a plugin for OpenFOAM that enables the use of *SpeedIT* linear solvers in OpenFOAM. But the plugin only supports old OpenFOAM versions (1.7.1, 2.1 and 2.2). Also, the free version only supports the single precision calculations which is not acceptable for a large number of scientific computations which needs to be done in double precision.



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

2.4.2 ofgpu

ofgpu is a free GPL library that provides GPU (sometimes referred to as GPGPU) linear solvers for OpenFOAM v2.2.x (an older version) [17]. The library targets NVIDIA CUDA devices on Windows, Linux, and (untested) Mac OS X. GPU acceleration holds the promise of providing significant speed up at relatively low cost and with low power consumption compared to other alternatives. *ofgpu* only supports single precision calculations.

2.4.3 Cufflink

Cufflink is a library for linking numerical methods based on NVIDIA's CUDA Architecture C/C++ programming language and OpenFOAM. This is somewhat of a dead project now. The last version of the OpenFOAM supported by the *cufflink* library was v1.6 using the CUDA framework 4.0. This also lets users

to replace OpenFOAM's linear solvers with its own GPU based linear solver implementations.

2.4.4 RapidCFD

RapidCFD is different to other libraries in that it uses GPU computation for performing most of OpenFOAM's functionality, not only the linear solvers. Most of the computation is done entirely on GPU. *RapidCFD* avoids copying data during calculations between CPU and GPU as much as possible. Most of the data structures are created on the GPU itself. Operations on these are then done by using thrust [21] library. Main features of *RapidCFD* are [22]:

- most incompressible and compressible solvers on static mesh are available.
- can run in parallel on multiple GPUs.
- Most of the calculations are done on the GPU and the overhead for GPU-CPU memory copy is minimum.



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

Although the library is open source it takes a significant amount of time to set it up on one's own. Also, the library doesn't support the GPUs of *sm_20* architecture.

2.4.5 Paralution

Paralution is a library that enables users to perform various sparse iterative solvers and preconditioners on multi/many-core CPU and GPU devices. Based on C++, it provides a generic and flexible design that allows seamless integration with other scientific software packages [23]. It supports various back-ends like OpenMP, CUDA and OpenCL. It also provides various plug-ins to common and popular softwares like Deal II, OpenFOAM, Matlab/Octave and FORTRAN programming language. Paralution is released under dual license scheme with an open source GPLv3 license and a commercial license. Free license doesn't support MPI functionality.

2.5 NVIDIA AmgX

AmgX provides a simple way to access accelerated core solver technology on NVIDIA GPUs [24, 25]. AmgX provides up to 10x acceleration to the computationally intense linear solver portion of simulations, and is especially well suited for implicit unstructured methods. It is a high performance, state-of-the-art library and includes a flexible solver composition system that allows a user to easily construct complex nested solvers and preconditioners. AmgX is available with a commercial and a free license. The free license is limited to Accelerated Computing Developers and non-commercial use.

The AmgX library offers optimized methods for massive parallelism, the flexibility to choose how the solvers are constructed, and is accessible through a simple C API that abstracts the parallelism and GPU implementation. Using the methods and tools from the AmgX library, developers can easily create specialized solvers using AmgX core methods and rapidly deploy solution on GPU workstations, servers and clusters. Main features of the AmgX library include:

- Flexible configuration allows for nested solvers, smoothers, and preconditioners.
- Ruge-Steuben algebraic multigrid.
- Un-smoothed aggregation algebraic multigrid.
- Krylov methods: PCG, GMRES, BiCGStab, and flexible variants.
- Smoothers: Block-Jacobi, Gauss-Seidel, incomplete LU, Polynomial, dense LU.
- Support for Scalar or coupled block systems.
- MPI and OpenMP support.
- Flexible and simple high level C API.

Chapter 3

METHODOLOGY

This chapter presents the profiling data of our OpenFOAM simulation we gathered using *callgrind* and describes the methodology we used to port OpenFOAM's linear solvers to GPUs. We talk about the structure and API of our AmgX wrapper library.

3.1 Profiling Data

In order to speed up the simulations, first thing we need to do is identifying the performance bottlenecks. We ran our simulation using *callgrind* tool which ships with *valgrind* to identify the computationally intensive sections of the program [26]. Figure 3.1.1 and 3.1.2 shows the profiling results we got.

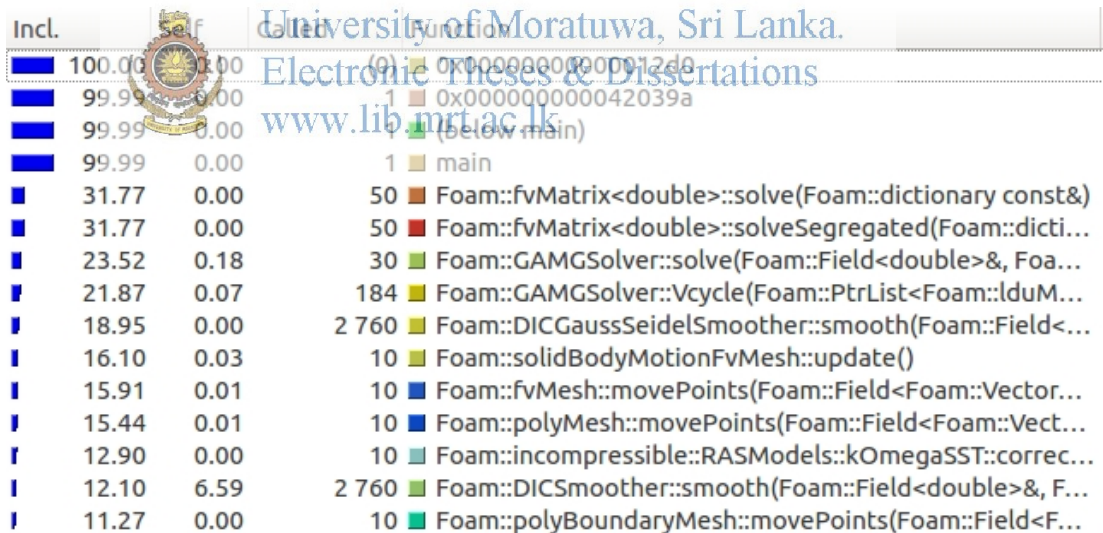


Figure 3.1.1: Top 15 Hot-spots by accumulated overhead of the method

According to Figure 3.1.1, *Foam::fvMatrix::solve* method has the highest accumulated overhead (discarding the *main* method). Nearly 1/3 of the simulation time is spent on this single method. In OpenFOAM, *Foam::fvMatrix* is the class that holds the matrix resulting from the finite volume discretization. *solve*

Incl.	Self	Called	Function
1	12.10	6.59	2 760 Foam::DIC smoother::smooth(Foam::Field...
1	7.68	6.27	2 885 Foam::GaussSeidel smoother::smooth(Fo...
	6.15	5.21	10 245 Foam::lduMatrix::residual(Foam::Field<d...
1	8.99	3.96	81 Foam::fv::cellLimitedGrad<Foam::Vector...
	4.52	3.73	4 680 Foam::lduMatrix::Amul(Foam::Field<dou...
	2.96	2.96	79 032 713 0x00000000000002a30
	4.11	2.34	32 469 010 Foam::face::sweptVol(Foam::Field<Foam::...
	2.16	2.16	78 921 229 Foam::face::centre(Foam::Field<Foam::V...
	3.79	1.78	450 Foam::GAMG Agglomeration::agglomerat...
	1.79	1.68	119 110 821 Foam::faceAreaIntersect::triSliceWithPla...

Figure 3.1.2: Top 10 Hot-spots by self contributed overhead of the method

method is the member which solves the linear system associated with this matrix. So, it can be concluded that a large portion of the simulation time is spent solving the linear system.

This fact becomes more obvious with Figure 3.1.2. Figure 3.1.2 shows the overhead introduced by methods ignoring the overhead of the callees. Four out of the top 5 hot-spots are methods used in solving the linear system. It is evident that we can maximize our speed up by making linear solvers run faster i.e., linear solvers are the perfect candidates to be implemented in GPU.

3.2 AmgX Wrapper Library for OpenFOAM

We used the NVIDIA's AmgX library described in section 2.5 to solve the OpenFOAM's linear system in the GPU. We implemented a wrapper library which enables the easy use of AmgX's linear solvers from OpenFOAM. Figure 3.2.3 shows the overall structure and interaction of the wrapper library with OpenFOAM. We found the work done by Pi-Yueh Chuang and Lorena A. Barba to add AmgX support to *PETSc* [27] really helpful when writing our wrapper library.

As shown in the Figure 3.2.3, pre-processing and post-processing takes place in the CPU using normal OpenFOAM utilities. But during the solving process, linear system (Matrix and the right hand side vector) is copied to the GPU by the AmgX wrapper library and the AmgX solvers are invoked on the system. After the AmgX library finishes solving the system, results are copied back to the CPU

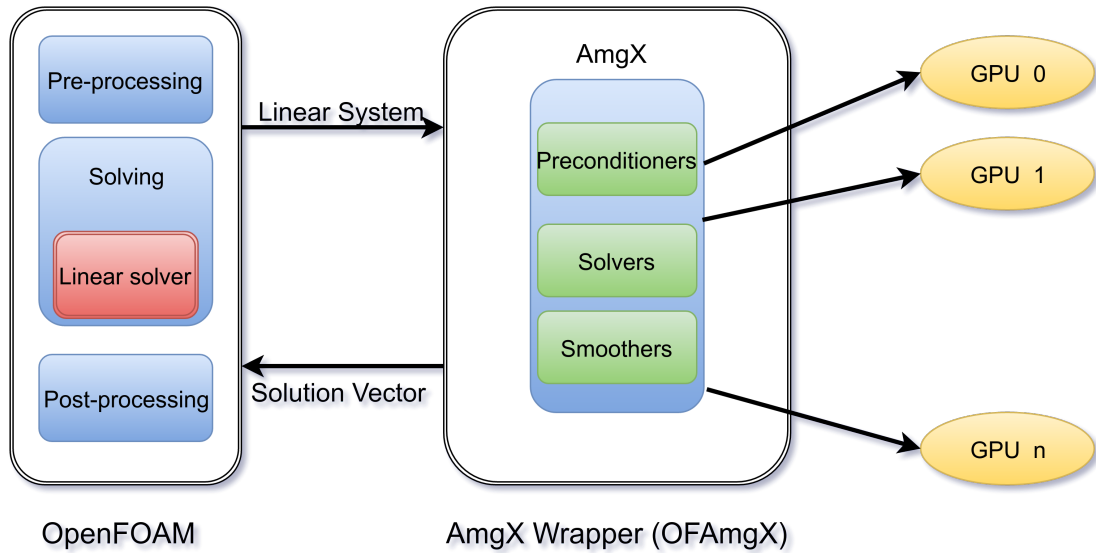


Figure 3.2.3: AmgX Wrapper Library

by the AmgX Wrapper.

User can specify the solution algorithm, tolerance values, preconditioners and smoothers in the OpenFOAM side as in a normal OpenFOAM simulation. AmgX wrapper will read those values and setup the solver in the GPU according to that information.

3.2.1 Data Structure Conversion

Before solving the linear system, AmgX wrapper has to convert the matrix and the right hand vector to a format consumable by the AmgX library. After solving the system, wrapper has to convert the solution vector back to a OpenFOAM vector. Conversion between the vectors is pretty straight forward. But the conversion between the matrices require additional work.

OpenFOAM stores its matrices in the *lduMatrix* format [28, 29]. In the *lduMatrix* format, lower, diagonal and upper elements of the matrix are stored separately in different arrays. Addressing for these elements are stored in another two arrays which stores the row and column index of each element. This storage method is extremely efficient for storing the matrices resulting from finite volume

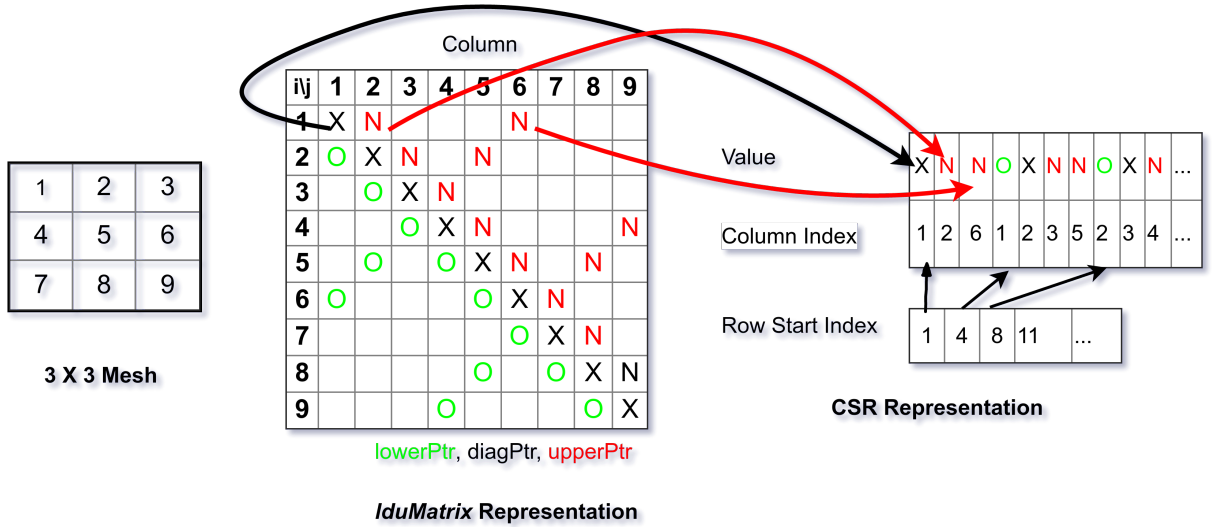


Figure 3.2.4: Conversion of *lduMatrix* to CSR Format

discretization.

AmgX solver library only accepts matrix in CSR format [30]. In CSR format, only the nonzero elements in the matrix are stored. All the nonzero elements of the matrix are stored row wise in one array (*value* in Figure 3.2.4) and the column indices of these elements are stored in another array (*column index* in Figure 3.2.4). A third array keeps track of the index of the start element in each row. To use AmgX with OpenFOAM, we need to convert *lduMatrix* format to the CSR format and this is done by our AmgX wrapper. Figure 3.2.4 shows visually how the conversion can be done from *lduMatrix* to *CSR*.

3.2.2 MPI Support

Our AmgX wrapper supports using multiple GPUs in a cluster and/or node to solve the linear system with MPI. The major issue in enabling multiple GPU support is the mapping of MPI processes to the GPUs. Suppose we have a GPU cluster like Figure 3.2.5 with three nodes.

First, the global communicator is split into in-node communicators which are

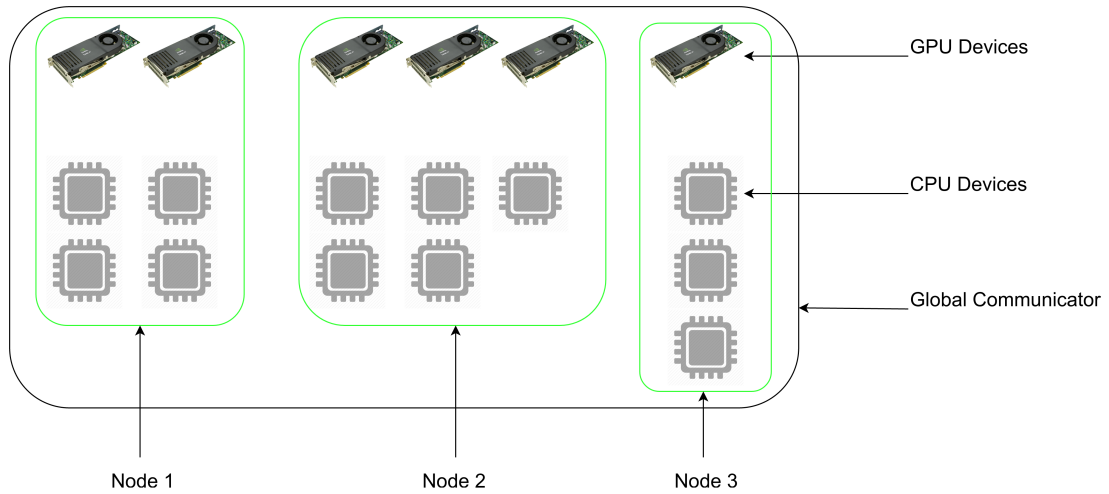


Figure 3.2.5: GPU Cluster with three nodes

local to each node. Figure 3.2.6 shows the cluster after this initial split.

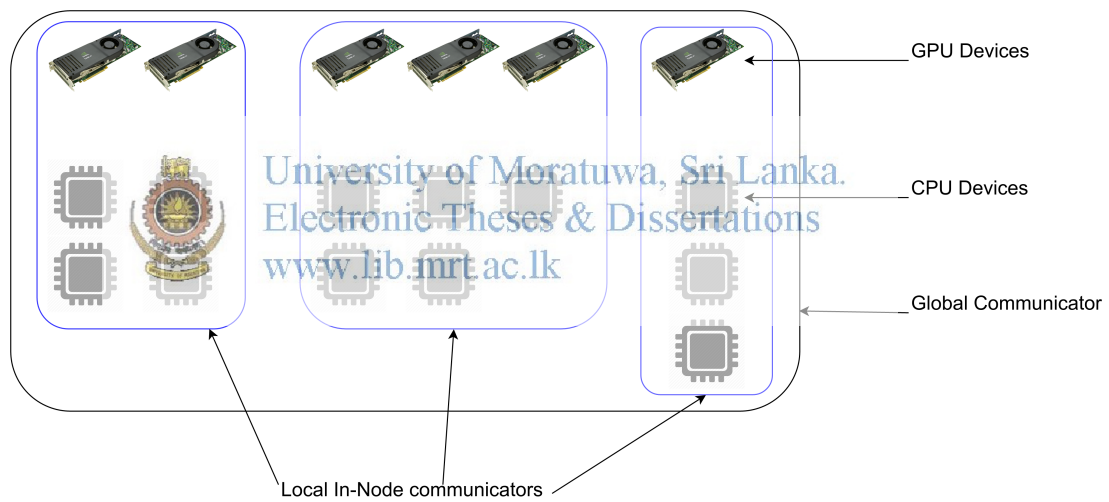


Figure 3.2.6: Global communicator split into local in-node communicators

Next, each in node (or local) communicator is divided depending on the number of GPUs available at the node and the number of MPI processes started at the node. Figure 3.2.7 shows the communicators local to each GPU device after this split. Usually, in OpenFOAM, the number of MPI processes equal to the number of cores in the node. We want to make sure that each GPU device have almost equal loads. Suppose we have n GPU devices and m MPI processes in a given node.

If m is divisible by n , then each GPU device will get $\frac{m}{n}$ MPI processes. This is the case with the leftmost and the rightmost nodes in Figure 3.2.7. Suppose m is not divisible by n and leaves a remainder r after division. Then $m - r$ is divisible by n . In this case, r devices will get $\frac{m-r}{n} + 1$ MPI processes each and the rest $n - r$ devices get $\frac{m-r}{n}$ MPI processes each. This is the case with the middle node in Figure 3.2.7.

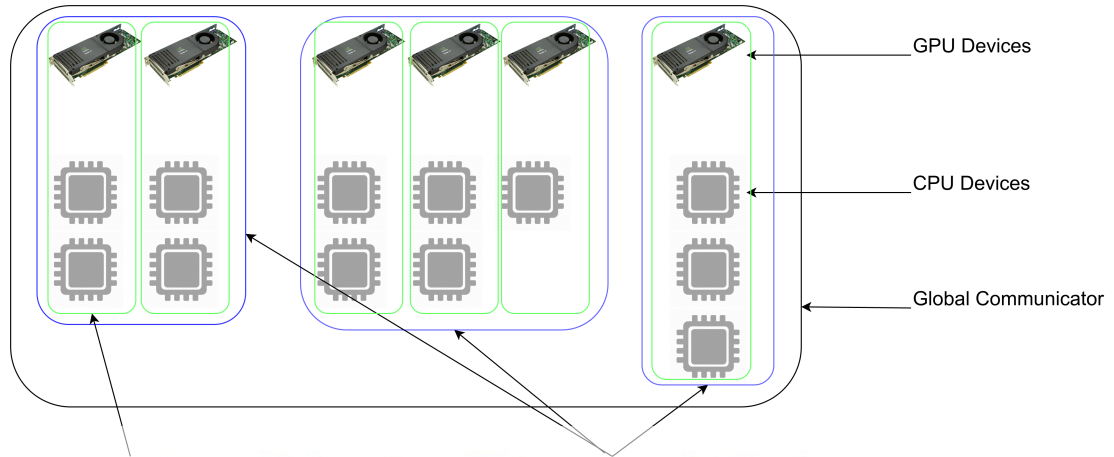


Figure 3.2.7: Local in-node communicators split by device

3.2.3 Multiple solver support

In a general OpenFOAM simulation, different types of solvers may be used to solve for different physical quantities. For example, the solver used for calculating pressure in the mesh points may not be used for calculating the speed at the mesh points. So, it is essential that our wrapper supports multiple types of solvers to be used in the same simulation.

When transforming solver configurations from OpenFOAM to AmgX, different solvers used in OpenFOAM have different configurations in AmgX as well. So, the configuration string used by AmgX wrapper to initialize solvers in the GPU are unique. We can use this configuration string and AmgX data structures

used to run that particular solver in GPU as key value pairs in dictionaries.

For a solver to be run by AmgX, a resource handle and a solver handle must be created. To support multiple solvers, we stored resource handlers of the solvers in one dictionary and solver handlers in another dictionary. During the simulation, we can select appropriate solver and resource handles depending on the configuration string and run the required solver on the GPU. Another advantage of this method is that we only need to initialize the AmgX library once. After initialization is complete, any solver can be run in the GPU. After the simulation is over, we can finalize the AmgX library. This step needs to be done only once as well. This saves a lot of time, especially in smaller simulations.

3.2.4 AmgX wrapper with RapidCFD

We improved the wrapper to work with RapidCFD described in section 2.4.4. However, we couldn't figure out how to enable MPI functionality in RapidCFD. RapidCFD is compiled using `nvcc` (Nvidia CUDA Compiler) with `gcc` as normal OpenFOAM library. We ran into a scope resolution error while trying to copy the global MPI communicator from RapidCFD to AmgX wrapper.

3.2.5 AmgX wrapper API

AmgX wrapper API is very simple. Amgx wrapper library has a only single class called *AmgXSolver*. User just need to instantiate an object of this class with the required configurations whenever he/she needs to create an solver which uses AmgX library. For a given simulation, first AmgXSolver instance does the initialization and finalization of the AmgX library. This object need to be initialized with null configuration. Figure 3.2.8 shows how the library should be used.

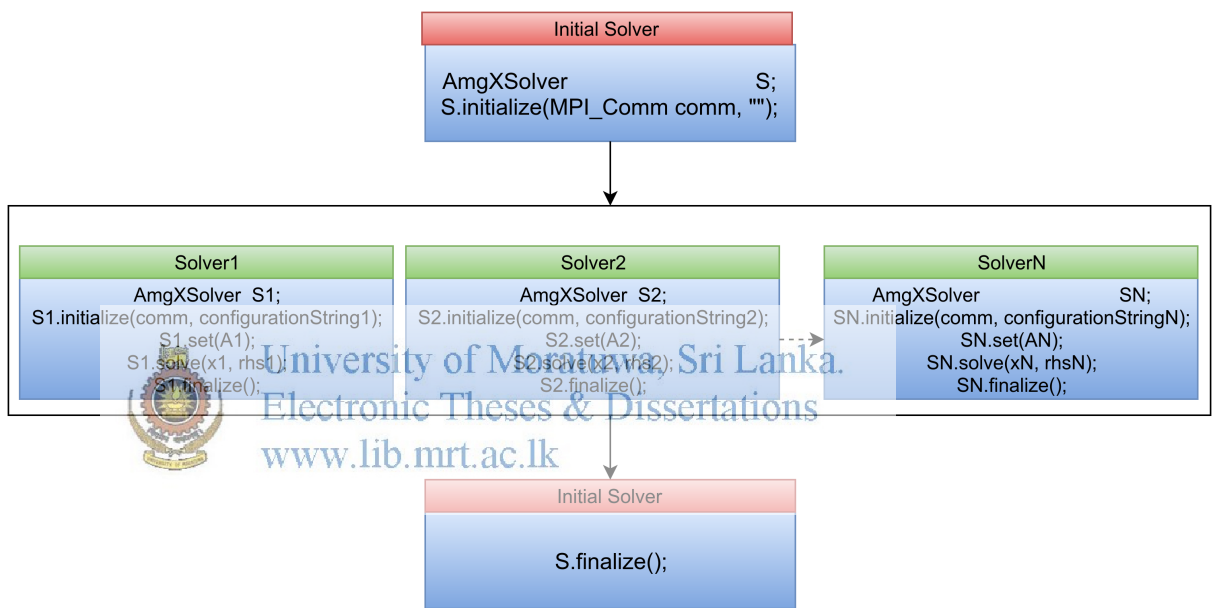


Figure 3.2.8: OFAmgX API Flow

Chapter 4

EXPERIMENTAL RESULTS

This chapter describes our experimental setup and the results of the experiments we did to benchmark AmgX wrapper library with native OpenFOAM and other GPU implementations of OpenFOAM

We benchmarked native OpenFOAM, AmgX wrapper library, Paralution library (section 2.4.5) and RapidCFD (section 2.4.4) library under three different hardware environments. These environments are listed in Table 4.1. We used a server (A) and two desktop machines (B and C). Each machine is equipped with GPUs and two desktop machines have two GPUs each. All the benchmarks were run using Ubuntu 14.04 and CUDA 6.5. OpenFOAM version 2.4.x was used in the benchmarks and the latest RapidCFD master from its git repository was used. All the measurements are in seconds.



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

We used two different simulations under each environment to benchmark the libraries. We used our original simulation described in section 1.3 and another simulation which simulates wing of a fan (FanWing2D). The latter simulation is relatively smaller than the former. Table 4.2 provides a summary of the simulations used.

For the windLM simulation, experiments are carried out with MPI and without MPI. Table 4.3 lists the execution time for windLM simulation for the three environments without using MPI (i.e., without domain decomposition). Table 4.4 – 4.6 list the execution time for the windLM with MPI using 2,4 and 8 MPI processes.

For the FanWing2D simulation, experiments are carried out without MPI since

Table 4.1: Different Experimental Environments

Environment	Details
A	32 Core Machine with 64 GB memory and a Tesla C2070 GPU
B	Intel core i7 @ 3.40GHz X 4, 16 GB ram with 2 GTX 480 GPUs
C	Intel core i7 @ 1.6GHz X 4, 8 GB ram with 2 GTX 480 GPUs

Table 4.2: Different Simulations used in the benchmarks

Simulation	Details
windLM	Simulation of a wind turbine with 1,166,000 mesh-points
FanWing2D	Simulation of a 2D Fan with 60,000 mesh-points

Table 4.3: Total simulation time for windLM without MPI (in seconds)

	OpenFOAM	AmgX Wrapper	RapidCFD	Paralution
A	648.93	634.07	887.86	971.87
B	313.18	296.06	Note I	288.98
C	463.46	434.15	Note I	428.73

Table 4.4: Total simulation time for windLM with 2 MPI processes (in seconds)

	OpenFOAM	AmgX Wrapper	RapidCFD	Paralution
A	361.30	350:00	Note III	Note II
B	216.80	201.15	Note I	Note II
C	404.86	365.91	Note I	Note II

Table 4.5: Total simulation time for windLM with 4 MPI processes (in seconds)

	OpenFOAM	AmgX Wrapper	RapidCFD	Paralution
A	144.33	152.33	Note III	Note II
B	154.33	141.67	Note I	Note II

Table 4.6: Total simulation time for windLM with 8 MPI processes (in seconds)

	OpenFOAM	AmgX Wrapper	RapidCFD	Paralution
A	102.67	113.67	Note III	Note II
B	154	144.33	Note I	Note II

Table 4.7: Total simulation time for FanWing2D without MPI (in seconds)

	OpenFOAM	AmgX Wrapper	RapidCFD	Paralution
A	67.61	79.51	668.70	32.71
B	29.18	26.85	Note I	31.52
C	50.44	45.24	Note I	51.76

the domain decomposition is not possible. Table 4.7 lists the execution time for FanWing2D simulation for the three environments without using MPI.

Notes:

- I We were unable to configure RapidCFD to work in GTX 480 GPUs
- II Paralution free version doesn't support MPI
- III RapidCFD gives an error when run using parallel mode

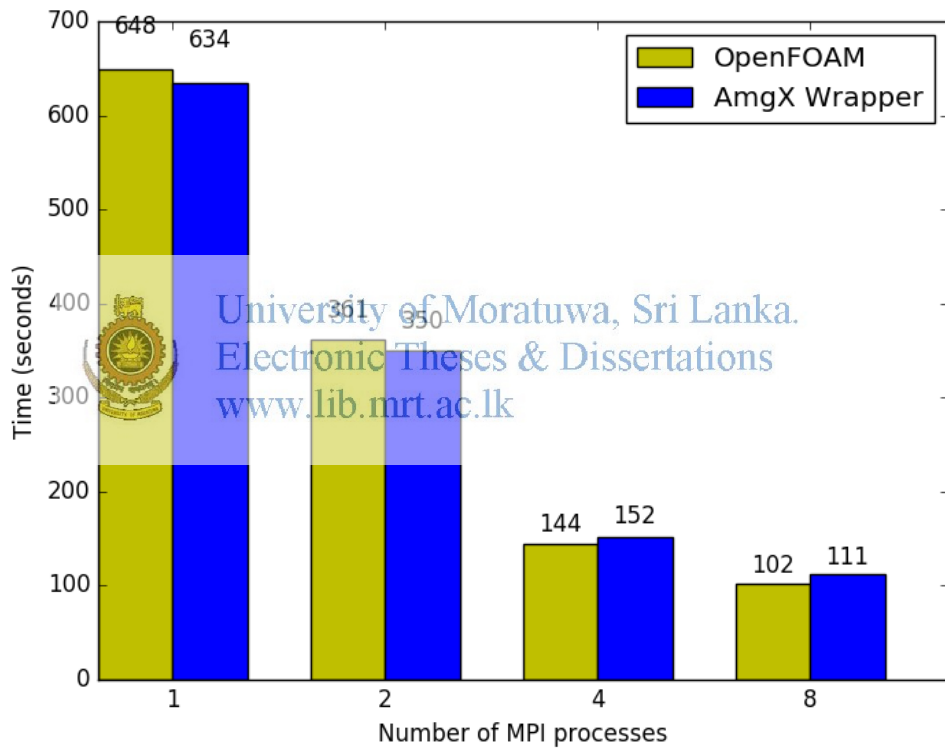


Figure 4.0.1: Performance of windLM simulation in A

Figures 4.0.1 and 4.0.2 summarizes the results of windLM simulation for the environments A and B respectively. In A, AmgX wrapper gets slower as the number of MPI processes increase. This is due to the fact that it has a single GPU and all the MPI processes start competing to use this GPU. In B, AmgX wrapper is always faster but the performance for 8 MPI processes is slower than for

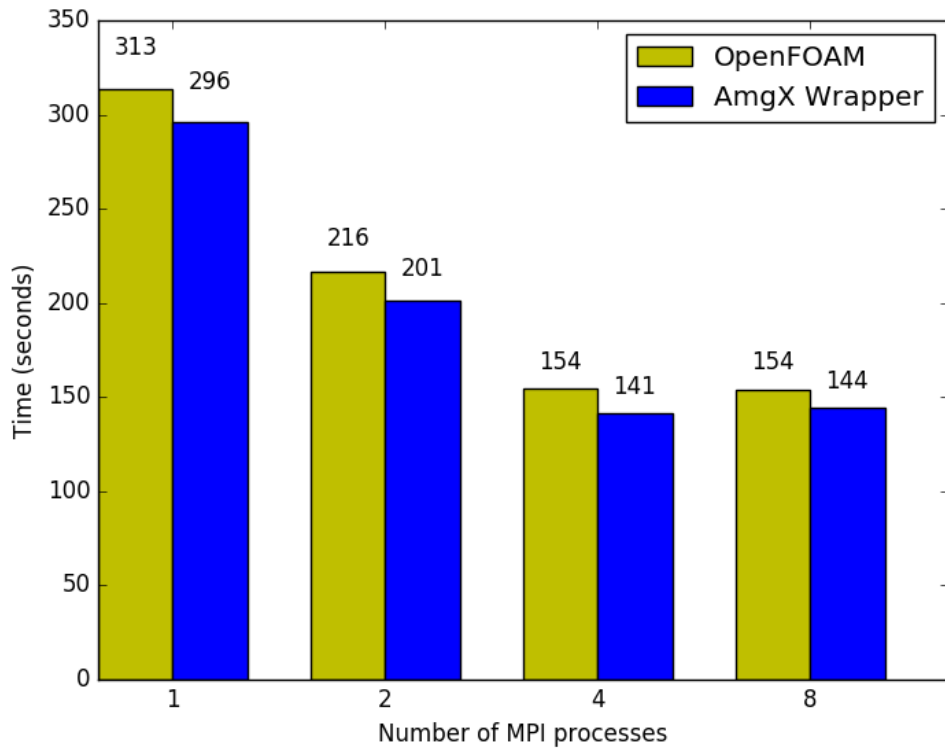


Figure 4.0.2: Performance of windLM simulation in B
 University of Moratuwa, Sri Lanka.
 Electronic Theses & Dissertations
 www.lib.mrt.ac.lk
 4 MPI processes. This is due to the same reasons which caused the performance drop in A.

If we consider the time taken to solve the linear system instead of the total simulation time, AmgX wrapper gives around **2x** speedup depending on the solver. For example, Preconditioned Biconjugate Gradient (PBiCG) solver with DILU preconditioner takes 441383 seconds to complete in OpenFOAM. In the AmgX wrapper, this only takes 213115 seconds to complete.

Chapter 5

CONCLUSIONS AND RECOMMENDATIONS

This chapter analyses the experimental results presented in chapter 4 and suggests possible future work


According to chapter 4, in most of the experimental cases we get closer to a 8% speedup in the total simulation time by using AmgX wrapper compared to native OpenFOAM. In some cases it is even faster than Paralution library. Moreover, our AmgX wrapper supports domain decomposition using MPI and the free version of the Paralution library doesn't have MPI support.

RapidCFD library doesn't really show any promising results. This may be due to the fact that it doesn't support *sm_20* GPU architecture. All the GPUs we used in our experiments are built on *sm_20* architecture. We had to change some of its code to get it work on *sm_20* architectures. For these reasons, it is hard for a non technical user to setup RapidCFD on his/her own.

Backed by promising results we got, to our knowledge, AmgX wrapper library is the best non-commercial option to run OpenFOAM's linear solvers in GPUs. Since we are using AmgX library, users get access to a lot of functionalities which are not available in original OpenFOAM.

There are some areas in our wrapper library which can be improved further. Although we added support for multiple solvers to be used in the same solution, we couldn't get significant improvements by using multiple solvers in the same simulation. Also, we couldn't beat OpenFOAM's multi-grid solver (GAMG) with the multi-grid solver available in AmgX library. Improving these may involve finding the best parameter values to be used with AmgX.

References

- [1] Amani A AlOnazi. *Design and optimization of openfoam-based CFD applications for modern hybrid and heterogeneous HPC platforms*. PhD thesis, 2014.
- [2] Hrvoje Jasak, Aleksandar Jemcov, Zeljko Tukovic, et al. Openfoam: A c++ library for complex physics simulations. In *International workshop on coupled methods in numerical dynamics*, volume 1000, pages 1–20. IUC Dubrovnik, Croatia, 2007.
- [3] J Cohen and M Jeroen Molemaker. A fast double precision cfd code using cuda. *Parallel Computational Fluid Dynamics: Recent Advances and Future Directions*, pages 414–429, 2009.
- [4] Julien C Thibault and Inanc Senocak. Cuda implementation of a navier-stokes solver on multi-gpu desktop platforms for incompressible flows. In *Proceedings of the 47th AIAA aerospace sciences meeting*, pages 2009–758, 2009.  www.lib.mrt.ac.lk
- [5] Amani AlOnazi, David Keyes, Alexey Lastovetsky, and Vladimir Rychkov. Design and optimization of openfoam-based cfd applications for hybrid and heterogeneous hpc platforms. *arXiv preprint arXiv:1505.07630*, 2015.
- [6] Jack Dongarra and Alexey L Lastovetsky. *High performance heterogeneous computing*, volume 78. John Wiley & Sons, 2009.
- [7] Alexey Lastovetsky, Vladimir Rychkov, and Maureen O’Flynn. Accurate heterogeneous communication models and a software tool for their efficient estimation. *International Journal of High Performance Computing Applications*, 2010.
- [8] David Clarke, Ziming Zhong, Vladimir Rychkov, and Alexey Lastovetsky. Fupermod: a software tool for the optimization of data-parallel applications

- on heterogeneous platforms. *The Journal of Supercomputing*, 69(1):61–69, 2014.
- [9] David Clarke, Aleksandar Ilic, Alexey Lastovetsky, and Leonel Sousa. Hierarchical partitioning algorithm for scientific computing on highly heterogeneous cpu+ gpu clusters. In *European Conference on Parallel Processing*, pages 489–501. Springer, 2012.
- [10] George Karypis and Vipin Kumar. Metis – unstructured graph partitioning and sparse matrix ordering system, version 2.0. Technical report, 1995.
- [11] Cédric Chevalier and François Pellegrini. Pt-scotch: A tool for efficient parallel graph ordering. *Parallel computing*, 34(6):318–331, 2008.
- [12] Pieter Ghysels and Wim Vanroose. Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Computing*, 40(7):224–238, 2014.
- [13] OW GEAR.  step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics*, 25:153–163, 1939. www.lib.mrt.ac.lk
- [14] Qingyun He, Hongli Chen, and Jingchao Feng. Acceleration of the openfoam-based mhd solver using graphics processing units. *Fusion Engineering and Design*, 101:88–93, 2015.
- [15] DP Combest and J Day. Cufflink: a library for linking numerical methods based on cuda c/c++ with openfoam. URL: <http://cufflink-library.googlecode.com>, 2011.
- [16] Vratis. Speedlt plugin for openfoam. URL: <http://speedit.vratis.com/index.php/products> (retrieved: 04-Jul-2016).
- [17] Symscape. ofgpu: Gpu linear solvers for openfoam. URL: <http://www.symscape.com/gpu-1-1-openfoam> (retrieved: 04-Jul-2016).
- [18] CUDA Nvidia. Cublas library. *NVIDIA Corporation, Santa Clara, California*, 15:27, 2008.

- [19] Zahra Jamshidi and Farshad Khunjush. Optimization of openfoam's linear solvers on emerging multi-core platforms. In *Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on*, pages 824–829. IEEE, 2011.
- [20] NVIDIA CUSPARSE. Cublas libraries.
- [21] Jared Hoberock and Nathan Bell. Thrust: A parallel template library. *Online at <http://thrust.googlecode.com>*, 42:43, 2010.
- [22] simFlow. Rapidcfd: Openfoam running on gpu. *URL: <https://sim-flow.com/rapid-cfd-gpu/> (retrieved: 04-Jul-2016)*.
- [23] Dimitar Lukarski and Nico Trost. Paralution project. *URL <http://www.paralution.com>. Accessed: December, 2014*.
- [24] NVIDIA. Amgx. *URL: <https://developer.nvidia.com/amgx> (retrieved: 04-Jul-2016)*.
- [25] Maxim Naumov, Marat Arsaev, Patrice Castonguay, Jonathan Cohen, Julien Demouth, Joe Eaton, Simon Layton, Nikolay Markovskiy, Nikolai Sakharnykh, Robert Strzodka, et al. Amgx: Scalability and performance on massively parallel platforms. In *SIAM workshop on exascale applied mathematics challenges and opportunities*. SIAM, 2014.
- [26] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavy-weight dynamic binary instrumentation. In *ACM Sigplan notices*, volume 42, pages 89–100. ACM, 2007.
- [27] Lorena A. Barba Pi-Yueh Chuang. Using amgx to accelerate petsc-based cfd codes. *GPU Technology Conference*.
- [28] CFD online community. ldumatrix. *URL: <http://www.cfd-online.com/Forums/openfoam-programming-development/67452-ldumatrix.html> (retrieved: 04-Jul-2016)*.

- [29] openfoamwiki. Matrices in openfoam. *URL:*
https://openfoamwiki.net/index.php/OpenFOAM_guide/Matrices_in_OpenFOAM
(retrieved: 04-Jul-2016).
- [30] Jack Dongarra. Compressed row storage (crs). *URL:*
http://netlib.org/linalg/html_templates/node90.html *(retrieved: 04-Jul-*
2016).



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk