ELEN4010 Software Development III: Group Report 2D Fluid Simulator for an HPC cluster

Rudolf Hoehler (0600134Y) Ronald Clark (363095)
Graham Peyton (309684) Justin Wernick (380536)
Edward Steere (400589)

March 9, 2012

Declaration:

Table 1: Breakdown of contributions by group members

Group member	**Project Hours**	**Contribution Field**	**Signature**
Rudolf Hoehler	16	Architecture research Development environment setup Report writing	
Justin Wernick	15	Algorithm research Setup of version control Report writing	
Edward Steere	15	Algorithm research User interface Research Report writing	
Graham Peyton	15	Software engineering methodology research MPI research Report writing	
Ronald Clark	17	Algorithm research Proof of concept implementation Report writing	

Abstract

High performance computing clusters are an effective means of performing complex simulations when significant computational power is required. In this report, a high level design of a computational fluid dynamics (CFD) simulation application is presented, and the requirements and proposed project plan are discussed. The system comprises a master program utilising MPI that communicates with a graphical user interface and the cluster nodes via input and output modules. Acceptance, integration and unit of testing will be used to validate the system. The work is split between the five group members based on strengths and weaknesses, and an iterative development methodology will be used to refine the solution.

1 Introduction

Computer simulation is a powerful tool for the study of complex physical systems [5]. Large computational requirements often neccessitate the use of parallel computing clusters in order to speed up the simulation process.

The aim of this project is to build a scalable application that utilises the additional memory/computational power available on a small cluster. The chosen application will perform a computational fluid dynamics (CFD) simulation. MPI will be used as a standardised interface between front end and back end components.

Section 2 of this report presents a review of the historical basis for parallel computing as well as existent solutions. A detailed problem specification is provided in section 3 and the associated requirements, contraints and resources are presented in section 4. The proposed design model is given in section 6. Sections 7 and 8 discuss the software engineering methodology that will be used and the proposed team organisational plan.

2 Existing Solutions

Parallel programming is not an entirely new concept. What is viewed as parallel programming today actually had its roots in the architecture of the first processors [3]. The first computational machines were entirely serial; that is, they processed information bit by bit [3]. There is an obvious reason why such a configuration was quickly replaced: it was slow and did not take advantage of parallel logic, e.g. two level logic. While the advance from a purely serial computer to computers which process words was the beginning of parallelisation, it is far from the modern definition. Today parellelisation is viewed as the process of making use of several computers (or processors internally in some cases) to complete some "super process". In order to simplify the process, several APIs have been developed to provide programmers with usable interfaces to such systems. MPI is one such system.

According to Pacheco, MPI is the most widely used parallel infrastructure today [2]. At the time of publication (1997) he claims that: "more and more, businesses and educational institutions are beginning to embrace parallel computing as a means to achieving higher performance in their computer systems" [2].

This is certainly evident when looking at the approaches adopted by algorithms in Bio Informatics. For example "WCD" (pronounced wicked) is an application, implemented in C, which performs genetic grouping algorithms (typically on ESTs or mRNAs) to group genetic sequences into similar clusters [1]. Here a data set of 6.6M 454 DNA sequences took approximately 35.5 hours to cluster on 32 machines, whereas a data set of 18M 454 DNA sequences took approximately 84 hours on a cluster of 128 machines [1].

Computational fluid dynamics (CFD) is another field in which the performance of applications can be dramatically increased through the use of high-performance computing (HPC) clusters. There is therefore a clear need for a good understanding of how MPI functions.

3 Problem Specification

The aim of the project is to develop a scalable applicatin capable of running on a HPC cluster. The application will utilise simple CFD analysis to carry out a 2D simulatino of a gaseous substance. It will allow the user to build a computational representation of a physical system by specifying the location and nature of disturbances. These disturbances will take the form of flow velocities (simulating wind sources) and boundaries (simulating objects). Simple fluid flow physics will then be applied to allow the system to output a prediction of the fluid dynamics. Performing these calculations at a high resolution will require a considerable number of computations and therefore take a long time to process. In this regard, the application is to be designed in a manner that allows it to take advantage of the supplied HPC cluster. All interaction between nodes is to be handled by a suitable, standardised API such as MPI/OpenMP. This is to allow the system to be as portable and scalable as possible - relinquishing the need to interface with the network at a low-level. The final application must provide a graphical user interface to create, monitor and view jobs.

4 Requirements and Constraints

4.1 Use Cases

The following use cases elaborate on the requirements presented in section 4.

Flow of Events for the User Running the Application Through the GUI

Preconditions

None

Main flow

The user starts the application in GUI mode. From here, they can set the simulation's initial conditions, and stimuli that will occur through the simulation. The user will also select the input and output files [E1]. The simulation settings, including the output file, will be saved in the input file [E2]. When the user clicks the simulate button, the simulation is run. The result of the simulation is written to the output file, and the simulation is displayed.

Subflows

None

Alternative flows

[E1] If the input or output files are not specified, default values will be used. [E2] The user is also able to set up a simulation to be run at a later time, on a different cluster. In this case, the user would stop after saving to the input file, and would not click the simulate button.

Flow of Events for the User Running the Application Through the Command Line

Preconditions

An input file must have been created through the GUI.

Main flow

The user launches the application from the command line. The input file giving the initial conditions and stimuli of the simulation is specified using a command line parameter [E1]. The simulation is started automatically, and the results are written to the output file. When the simulation is complete, the command line will display a completion notification and the application will exit.

Subflows

None

Alternative flows

[E1] If an input file is not specified, or the specified file does not exist, and error message will be displayed, and the application will exit.

4.2 Nonfunctional Requirements

4.2.1 Scalability

The application should be able to be run on a variety of sizes of data set. The number of computers in the cluster should also be able to vary. A larger number of computers in the cluster should result in a simulation completing in a faster time than if it were run on a smaller number of computers.

4.2.2 Accuracy

Errors resulting from approximations in numerical computations should be less than 0.1%.

4.3 Constraints

1. MPI must be used to interface with the cluster.
2. The Java programming language will be used.
3. Builds will be performed using OpenJDK version 1.6.0_23. The builds will be scripted using ANT.
4. Source code control will be performed using Git. A Git repository hosted on Github will be used.
5. Unit testing will be performed using the JUnit testing framework.
6. The simulation must be decoupled from the GUI, so that users may launch a simulation from the command line.
7. Third party source code may be used, but at least 30% of the application must be original source code, created by the group.
8. The project must be completed and submitted, including documentation, before 7:50 am on 21 May 2012.

5 Resources

A small high performance cluster will be available to simulate the final system. Initial prototyping will be done using personal computers. The following software packages and developments tool will be used:

1. Ubuntu Linux Operating System
2. JDK toolkit
3. MPIJava from HPJava
4. Eclipse IDE
5. git version control
6. Doxygen documentation formatting and management
7. Virtual Box virtual machines for easy multi-tier testing

Skills are another important resource. Table 2 shown in appendix A outlines the various strengths, weaknesses and working preferences of each group member. This table is used in section 8 as a basis for the distributions of project responsibilities among the group members.

6 Design Model

After careful consideration of the project specifications, project constraints, available resources, and the project team members' strengths, weaknesses and personal preferences a solution model was developed that is both functional as well as easily split between team members. The model illustrated in figure 6, breaks down the problem of a 2D simulation of the flow of a gaseous substance into modular manageable software elements, each discussed below.

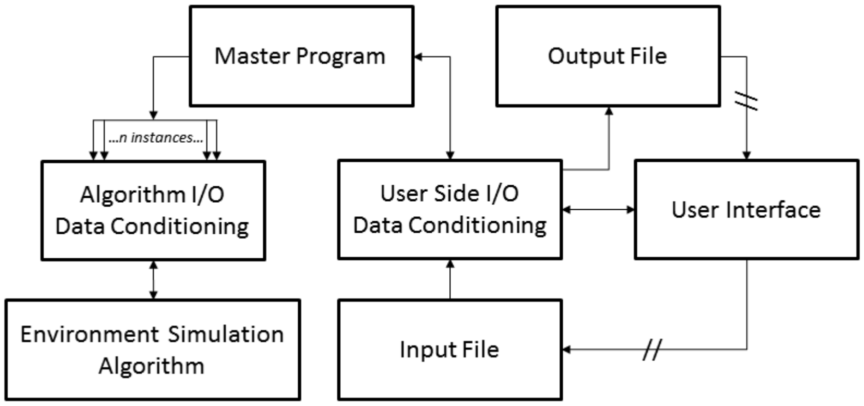
Master Program
Output File
...n instances...
Algorithm I/O Data Conditioning
User Side I/O Data Conditioning
User Interface
Environment Simulation Algorithm
Input File

Figure 1: Overview of the system architecture

6.1 User Interface

The user interface (UI) allows the user to set the initial conditions of the two dimensional gaseous environment as well as allowing the user to manually input any number of disturbances to the environment. Initial conditions and disturbance inputs may be set by uploading a data file or through user mouse interaction on a miniaturised version of the full scale two dimensional environment. The user based input is then converted to the required data format. Once the user is satisfied with the initial conditions as well as the trajectories and densities of disturbances, the user may commit the configuration to the input file and execute the master program.

If the main node configuration and regulation allows for a user interface, the user interface may receive input from the master program as far as simulation progress is concerned. The UI is completely decoupled from the master program and is not required for the master program to run. Once the simulation is done, the UI may play the finalised high-resolution simulation back to the user. Initially the UI will we built using the Java Swing library. If time allows, a more elaborate user interface will be implemented.

6.2 Input File

The program requires a raw data set of initial conditions as well as the proposed course of time dependent density co-ordinates (x-axis, y-axis, v–velocity, d-density) that the disturbances in the environment will take. The raw data set is to be contained in one file that may be generated by 3rd party software or by the team designed UI. The master program is tightly coupled to the input file and will not run if not supplied with a correctly formatted complete data input file.

6.3 User Side I/O Data Conditioning

The supplied data set is validated for correct formatting and completeness. The initial conditions of the gaseous environment are read and transferred to a super data set. The disturbance trajectory and intensity information is stored in a separate data set.

Once the simulation is completed the Master Node will process the production data of the simulation through the user side I/O module and output a data set to a binary file to be interpreted by the GUI (which then generates the graphical output to the user).

6.4 Master Program

The master program accepts the super data set and breaks it up into manageable subsets based on the number of available cluster nodes. As will be discussed in detail later, the environmental simulation module can only operate on a rectangular matrix of data. The rectangular configuration is necessary as the algorithm requires boundary condition input from adjacent matrices. Thus the partitioned data subsets need to fit exactly on a rectangle. For example, if 11 threads are available the master program will only create 10 sub data-sets from the super data set. That is, the master program will attempt to break up the super data set such that it maps directly to a rectangular matrix with the number of cells in the rectangular matrix making optimal use of the available threads.

The master program is responsible for managing nodes on a synchronised time step basis. After each step in time has been completed two options are available to the team for the operations which follow.

Master-Slave Architecture

Upon spawning each thread, the environmental simulation contained in each thread is provided with the initial conditions for its segment and disturbance data for the specified time. When the computation for a specific time step is completed, all information is transferred back the master node, which re-compiles a super data set for each time step. The master node is responsible for providing each thread with updated boundary conditions, new disturbance data and most importantly, keeping all threads synchronised.

Data-Parallel Architecture

Upon spawning each thread, the environmental simulation contained in each thread is provided with the initial conditions for its segment, disturbance data for the specified time and the thread identifiers for the threads that are calculating the adjacent environmental systems. Boundary conditions can then be passed between threads directly, without going through the master node first. Depending on storage availability on the nodes, each spawned process may store its time stepped segments locally, and only once the simulation has reached completion transfer all the data back to the master node that will then recompile and order all the data. This approach hopes to make slave nodes more intelligent and reduce latency introduced by master-node/slave-node communications.

The team will endeavour to explore and experiment with both approaches.

6.4.1 Algorithm I/O Data Conditioning

While the main program ensures correct thread spawning and management, this data conditioning module ensures that data sent to a thread is packaged and formatted correctly. This module is tightly coupled to the environment simulating module. A conditioning module is spawned and integrated with every environment simulation thread. This ensures that data conditioning is done on the slave nodes and not on the master node. This approach leave the master node free to analyse and distribute data as necessary and reconstruct results as they arrive.

6.4.2 Environment Simulation Algorithm

The cornerstone of the project is the environment simulation algorithm (ESA). Mathematically, the state of a fluid at a given instant of time is modeled as a velocity vector field; that is, a function that assigns a velocity vector to every point in space.

If one were to imagine movement of steam out of a kettle or the smoke twirling up from the tip of a cigarette bud, the complexity of the simulation becomes immediately clear. Each particle of smoke will have to be tracked and traced, as well as the trajectory of reflected light from the particle with respect to the observer calculated and rendered.

By reducing the problem to two dimensions, the problem complexity is reduced considerably. A two dimensional vector field requires significantly less computation to solve than a 3 dimensional problem. Furthermore the problem of calculating light interactions is collapsed to a simple problem of density – that is, a little bit of smoke will reflect little light and thus appear less dense. Conversely, a dense plume of smoke will reflect a lot of light and appear almost solid.

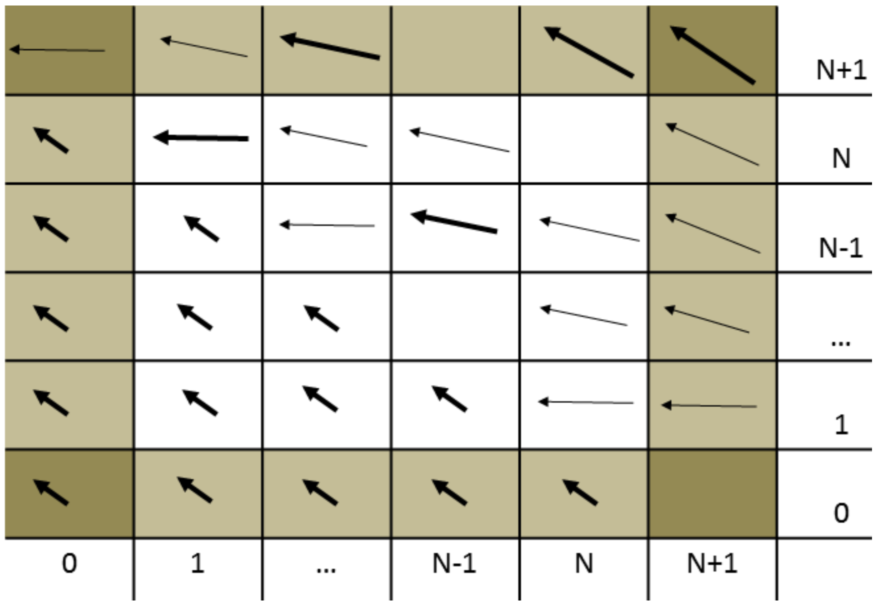

Figure 2: Illustration of the vector and density fields

Figure 2 graphically illustrates how the algorithm deals with the velocity and density fields. Each cell in figure 2 represents a potential smoke particle. The thickness of the

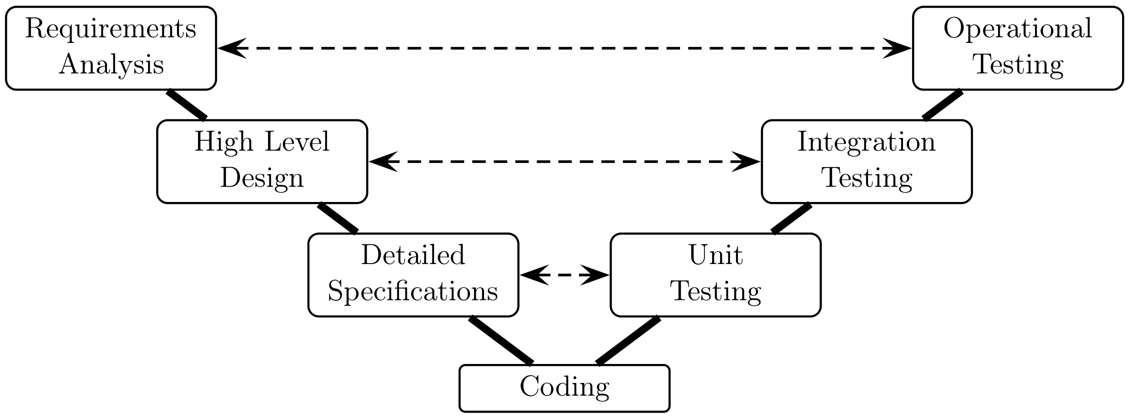

Figure 3: Illustration of the V-Model

arrow is directly related to its density, while arrow length and direction vectorially illustrates the smoke particle's velocity. An arrow of zero thickness (a block with no arrow) represents a smoke particle with zero density. Areas of zero density appear clear to the user on the simulation.

From figure 2 the reader should deduce that dense, relatively slow moving smoke particles are flowing from right to left diagonally upwards. The laminar flow is being disturbed by a thin, fast moving field of irregularly dense smoke particles flowing in an anti-clockwise direction.

The shaded areas indicate the boundary information from adjacent matrices. After each time step calculation, each data sub set needs new initial condition input from the adjacent data subsets before it can continue with the next time step calculation. While only one row of boundary conditions are shown in figure 2 for simplicity, multiple rows will need to be passed in the actual simulation to minimise numerical errors. The specific number of rows required will be calculated based on the maximum densities and velocities. From figure 2 it should be noted that eight sets of initial conditions are required for every data set.

7 Software Engineering Methodology

An incremental approach will be used during development. The nature of the project necessitates a more lightweight process model to account for at least two iterations of development. The various levels of progress are shown in figure 7.

Although the V-model will not strictly be used, it gives a good indication of how progress at each level may be quantified by means of various testing procedures.

Acceptance testing focuses on functionality and the inputs and outputs of the system. Functionality will be measured against the system requirements presented in section 4. The system will initially be simulated on a local network, and finally be run on the HPC cluster. Various indicators will be used to evaluate the performance of the system [4]:

1. Simulation time (of each node as well as the combined system)
2. Rendering efficiency
3. Aesthetic quality of the fluid simulation
4. Scalability of the system
5. Response of the system to erroneous input files.

Integration testing will be performed using a bottom-up approach [6]. Low-level components (such as the algorithm) will be tested and then integrated and coupled with components at the next higher level. The process will be repeated until the component at the top of the hierarchy is tested [6]. Integration tests will therefore be written to verify:

1. External display (sending messages to terminals)
2. Auditing and logging (performance data, obtaining, updating, or creating transactions on storage files)
3. I/O functionality (verify data integrity and successful transmission of boundary conditions and/or commands between components and the MPI framework)
4. Interoperability between subsystems
5. Cleanup operations (temporary files deleted, no memory leaks, pending/unused network connections)
6. Systems performance under severe resource constraints.

Lastly, unit testing will be used to test individual branches of code. Specific test cases will arise as coding progresses. Examples of tests that verify the algorithm will ensure that:

1. Erroneous inputs such as negative velocity or resolution are handled correctly
2. Accurate numerical calculations are performed (adding positive density and velocity, etc.)
3. Physical interactions occur realistically (density diffuses appropriately; density advects according to velocity field)

8 Group Organisation

The organisation of the team is based on a combination of the Agile and Skilled With Advanced Tools (SWAT) approaches. The combination allows for small independent groups to work according to their own schedule and requirements. This approach requires short communication channels, frequent incremental builds and much discipline from each team member [6]. The group will be split into four divisions based on individual strengths and weaknesses presented in table 2, appendix A:

Rudolf and Graham will coordinate the group as their strengths lie in communication, initiation and project facilitation. They will also be jointly responsible for the implementation of the Master Program and the maintenance of the development environments. The master program is intricately dependent on the I/O modules on both the front and back ends.

Justin will implement the I/O modules and will also be be partially responsible for the environment simulation module, as his strengths lie in low-level implementation. Justin will also manage the Git repository.

Ronald will act as the project architect since he possesses a good understanding of system integration. Ronald showed great competence at solving computational problems and will therefore be responsible for implementation of the environment simulation module, which is tightly coupled to the back-end I/O module.

Edward Steere has assumed responsibility for the user interface. Even though the user interface is completely independent of the rest of the program, clear communication will be required between Edward, Justin, Rudolf and Graham, as the user interface and the rest of the program are tightly coupled via the input and out files.

Appendix A, table 2 summarises how the project will be subdivided among group members, while appendix B and C give an overview of the important milestones.

9 Conclusion

High Performance Computing is becoming an invaluable tool to many researchers. By effectively exploiting the parallelism (and hence computing power), the cost and time required for complex simulations can be greatly reduced. This report provides an essential overview of the high-level components of such system which will be refined and implemented. The application will carry out fully configurable fluid simulations at high resolutions. Four major components were identified: the user interface, file input/output, user side I/O data conditioning and the master program. To ease the development effort, each component has been assigned to an individual team member(s). Finally, the success criteria and performance specifications have been set out which focus on the efficiency, scalability and accuracy of the system.

References

[1] S. Hazelhurst. *The wcd-express Manual.* University of the Witwatersrand, 2011.

[2] S. Pacheco. *Parallel Programming with MPI.* Morgan Kaufmann, New York, 1997.

[3] D. Padua. *Encyclopedia of Parallel Computing.* Springer, Urbana-Champaign, 2011.

[4] SideFX. Render quality and improving render time. http://www.sidefx.com/docs/houdini11.0/rendering/renderquality, Accessed: 7 March 2012, November 2011.

[5] Academia Sinica. How to build a parallel computing cluster. http://proj1.sinica.edu.tw/ statphys/computer/buildPara.html, Accessed: 7 March 2012, 2008.

[6] H. Vliet. *Software Engineering: Principles and Practice.* John Wiley and Sons, Georgia, July 2008.

Appendix A

Table 2: Personal attributes of group members

Group Member	**Strengths**	**Weaknesses**	**Work division**
Edward Steere	Programming on multiple levels of complexity Efficient, fast execution programming Experience in many languages Research and additional reading Strong Languages: C, C++, Python, HTML, JavaScript	Poor report writing Limited experience in Java (currently working through tutorials) Tend to over extend scope beyond what is required	Graphical User Interface Front end I/O (partially)
Graham Peyton	C++ and small amount of Java experience Report writing Research ability Fairly good initiator /facilitator	Poor Linux, networking and VM knowledge Lack of general architecture understanding	ManagerMaster Program & MPI
Justin Wernick	Good C++ coding, fair in Java mostly from hobbyist tinkering and experimenting Overall architecture designSome (limited) general knowledge of web applications from vac work experience Problem solving and debugging	Lack of enthusiasm for report writing Need to remember to listen to other people's ideas	Front and back end I/O CFD algorithm (partially)
Ronald Clark	Experience with graphics programming and simulation (OpenGL, Rendering) Good knowledge of C++, D, OpenGL, DirectX Good understanding of overall system and integration	Little experience with programming in Linux	Architect CFD algorithm
Rudolf Hoehler	Code Practically Good Communicator C++, Delphi EnterpriseHas multi threading experience	Lack of broad based knowledge and exposure Might sacrifice performance for practicality Limited Java Exposure	ManagerMaster Program & MPI

Appendix B

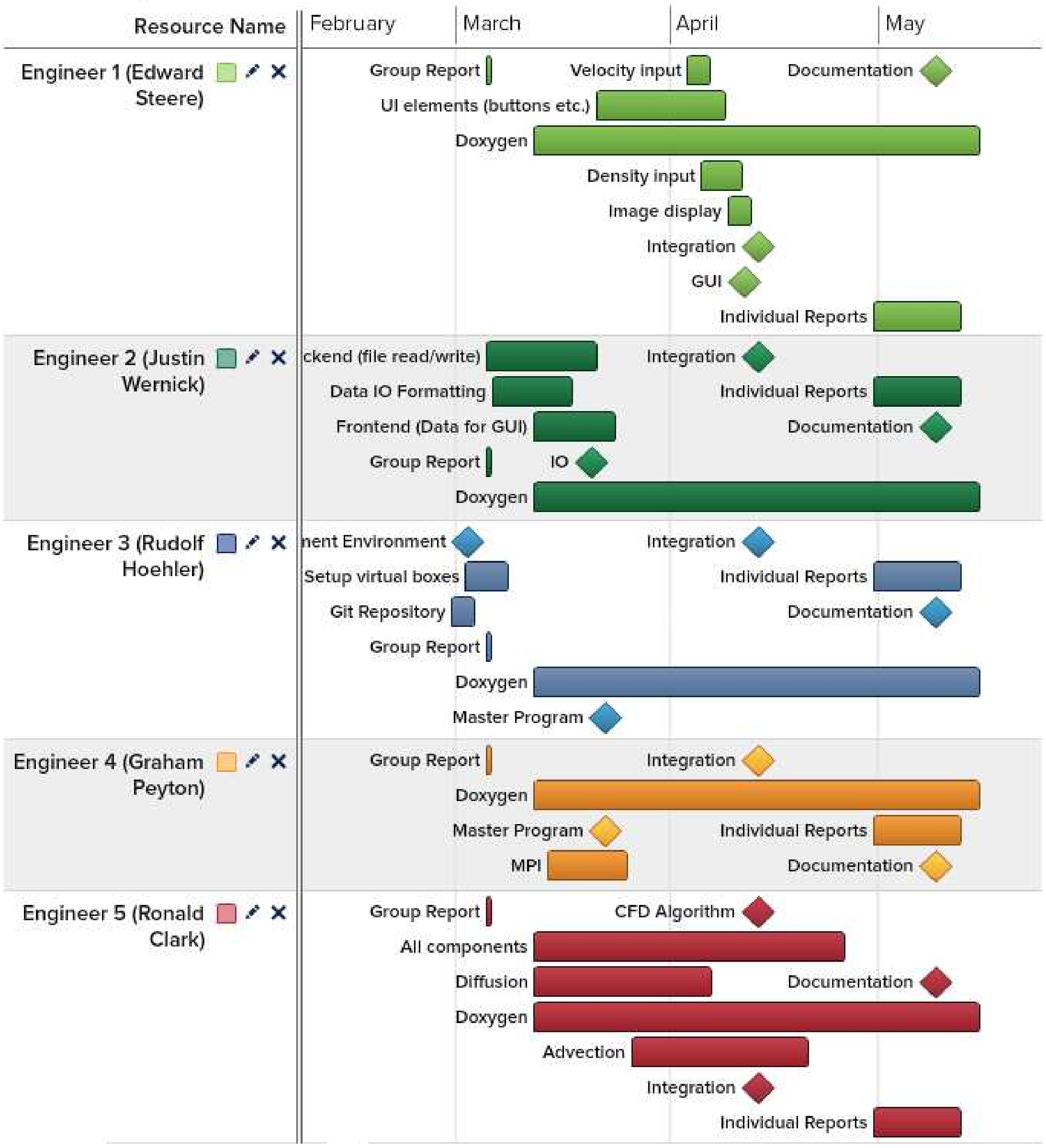

Figure 4: Gantt chart showing project milestones

Appendix C

The specific milestones set out for the project are as follows:

Laying the foundation - 16 March 2012

- MPI Master Program able to solve simple bulk counting problem across 5 virtual machines.
- Environment Simulation Algorithm solving on a single rectangular matrix.
- User Interface capturing mouse input.

Building Complexity - 23 March 2012

- MPI Master Program able to decompose large data set, perform remote operation across 5 virtual machines and recompose data set.
- Environment Simulation Algorithm solving 2 adjacent rectangular matrices
- User Interface accurately capturing mouse data and finalising data file format.

First Joint Build - 30 March 2012

- User Interface successfully generates input data file and executes MPI Master Program
- MPI Master Program correctly formats incoming data.
- MPI Master Program and Environment Simulating algorithm successfully spawn and solve two adjacent data sets.
- MPI Master Program correctly formats output data.
- User Interface successfully renders data.

Master Slave first build - 13 April 2012

- Successful implementation of a master slave architecture across 4 adjacent data sets - simple square.

Master Slave second build - 20th April 2012

- Successful implementation of a master slave architecture across 9 adjacent data sets - complex square.

Feature Freeze, Master Slave third build - 27th April 2012

- Successful implementation of a master slave architecture across n adjacent data sets.
- Feature Freeze on master slave architecture.
- Fix bugs, code clean-up, documentation update and empirical experimentation.

Parallel Communication first build - 4 May 2012

- Successful implementation of a parallel communication architecture across 4 adjacent data sets.

Feature Freeze, Parallel Communication second build - 11 May 2012

- Successful implementation of a master parallel communication across n adjacent data sets.
- Feature Freeze on parallel communication architecture.
- Fix bugs, code clean-up, documentation and empirical experimentation.

Development Freeze

- Bug Fixing
- Code Clean-up
- Empirical experimentation and comparison of master slave architecture vs. parallel communication architecture.