The Edward S. Rogers Sr. Department of Electrical and Computer Engineering

University of Toronto

ECE496Y Design Project Course

Group Final Report

Title: Performance Prediction of an MPI Application on an FPGA Cluster

Team #:	2018907	
Team members:	Name:	Email:
	Carlos Buenconsejo	carlos.buenconsejo@mail.utoronto.ca
	Ihor Stempen	igor.stempen@mail.utoronto.ca
	Zhaotong Zheng	ztony.zheng@mail.utoronto.ca
Supervisor:	Paul Chow	
Section #:	3	
Administrator:	Nick Burgwin	
Submission Date:	March 21, 2019	

Group Final Report Attribution Table

This table should be filled out to accurately reflect who contributed to each section of the report and what they contributed. Provide a **column** for each student, a **row** for each major section of the report, and the appropriate codes (e.g. 'RD, MR') in each of the necessary **cells** in the table. You may expand the table, inserting rows as needed, but you should not require more than two pages. The original completed and signed form must be included in the hardcopies of the final report. Please make a copy of it for your own reference.

Section	**Student Names**			
	Ihor Stempen	Zhaotong Zheng	Carlos Buenconsejo	
Executive Summary	ET	RD, MR		
Group Highlights		RD	ET	
Introduction	MR	MR	RD	
Final Design	RS,RD,MR	ET	RS,RD	
Testing and Verification	RD,MR	MR,ET	MR	
Conclusion	ET	RD	ET	
All	ET	ET	ET	

Abbreviation Codes:

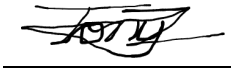
Fill in abbreviations for roles for each of the required content elements. You do not have to fill in every cell. The "**All**" row refers to the complete report and should indicate who was responsible for the final compilation and final read through of the completed document.

RS – responsible for research of information
RD – wrote the first draft
MR – responsible for major revision
ET – edited for grammar, spelling, and expression
OR – other
"All" row abbreviations:
FP – final read through of complete document for flow and consistency
CM – responsible for compiling the elements into the complete document
OR - other
If you put OR (other) in a cell please put it in as OR1, OR2, etc. Explain briefly below the role referred to:
OR1: enter brief description here
OR2: enter brief description here

Signatures

By signing below, you verify that you have read the attribution table and agree that it accurately reflects your contribution to this document.

Name Ihor Stempen **Signature**  **Date:** 2019-03-21

Name Zhaotong Zheng **Signature**  **Date:** 2019-03-21

Name Carlos Buenconsejo **Signature**  **Date:** 2019-03-21

Voluntary Document Release Consent Form[1]

To all ECE496 students:

To better help future students, we would like to provide examples that are drawn from excerpts of past student reports. The examples will be used to illustrate general communication principles as well as how the document guidelines can be applied to a variety of categories of design projects (e.g. electronics, computer, software, networking, research).

Any material chosen for the examples will be altered so that all names are removed. In addition, where possible, much of the technical details will also be removed so that the structure or presentation style are highlighted rather than the original technical content. These examples will be made available to students on the course website, and in general may be accessible by the public. The original reports will not be released but will be accessible only to the course instructors and administrative staff.

Participation is completely voluntary and students may refuse to participate or may withdraw their permission at any time. Reports will only be used with the signed consent of all team members. Participating will have no influence on the grading of your work and there is no penalty for not taking part.

If your group agrees to take part, please have all members sign the bottom of this form. The original completed and signed form should be included in the hardcopies of the final report.

Sincerely,
Khoman Phang
Phil Anderson
ECE496Y Course Coordinators

Consent Statement

We verify that we have read the above letter and are giving permission for the ECE496 course coordinator to use our reports as outlined above.

Team #: _____ Project Title: _____

Supervisor: _____ Administrator:

_____

Name		**Signature**		**Date:**	
Name		**Signature**		**Date:**	
Name		**Signature**		**Date:**	

[1] This form will be detached from the hardcopy of the final report. Please make sure you have nothing printed on the back page.

Executive Summary (author: Zhaotong Zheng)

Field Programmable Gate Arrays (FPGAs) are emerging as a new area of research in the cloud computing industry due to their reconfigurability and performance. When used together with CPUs in a heterogeneous network, the FPGAs can function as hardware-based accelerators for software programs. The University of Toronto (UofT) is currently implementing such a network of FPGAs using Message Passing Interface (MPI) and High Level Synthesis (HLS). However, these technologies are immature, and can result in poor performance compared to pure software-based cloud computing. Developers must then spend valuable time determining what parts of their software to optimize.

In order to increase workflow efficiency, UofT wants a simulation tool that can predict the performance of an MPI program if it is accelerated using the MPI FPGA cloud, based on factors such as network specifications, FPGA hardware speed, and MPI communication time. The simulator must present the data in a manner that is easy to process and analyze, must have a user interface that minimizes the amount of time spent inputting data, and must be accurate within 10% of actual runtime results. Additionally, UofT wants development of an MPI application that utilizes the MPI FPGA cloud. This application should demonstrate the functionality and accuracy of the performance simulator. Thus, the MPI application also serves as a primary validation test for the performance simulator.

The performance simulator is implemented using an existing open source framework called *MPI Parallel Environment (MPE).* MPE works via instrumentation of source code to generate performance measurement logs during runtime of a purely software based MPI program. Runtimes and communication times of MPI messages within the MPE data logs are then modified through a back-propagation algorithm based on user inputted specifications of the MPI FPGA cloud network. This will generate new output logs that predict the overall runtime performance of an MPI program running on the MPI FPGA cloud versus a purely software-based solution. This serves as the main backend algorithm design of the performance simulator. A GUI serves as the UI for user input and control.

For the MPI application, a molecular simulation called Molecular Dynamics (MD) was selected. It's design is the same for both software and hardware versions, and consists of 3 components. An *Atom Manager* acts as the main control thread that controls the *N* number of parallel *Compute Engine* threads that actually do the physics calculations. Lastly, the visualization component then outputs a graphical visualization of the moving atoms via an open source framework called VMD.

Fully functional implementations of the simulator and the software version of MD are complete. However, the porting of MD to hardware on the FPGA platform has not been achieved due to issues outside of our project's scope and our team's knowledge. We have worked together with our supervisor and his team to try and resolve the issues, but have not found a solution in time for this report. As a result, we have not met all project requirements as initially proposed (simulator accuracy untested, etc) and cannot call our project a full success. However, we have still managed to deliver a functional simulator program and software MD application. With these two working deliverables, we were able to prove our initial design idea and workflow by running software MD through our simulator. Through it, we were able to pinpoint bottlenecks in software preventing hardware acceleration in MD, and demonstrated reasonable simulator accuracy; albeit in a way different than our initial proposed test. Thus, despite our shortcomings, we have overall completed 2 out of 3 demonstrable deliverables.

Group Highlights (author: Zhaotong Zheng)

As a quick reminder, the structure of our project consists of two distinct parts to be completed in each respective semester. The two distinct parts consists of the Performance Simulator and the MPI FPGA Application. Of these two parts, the Performance Simulator was scheduled to be completed first by the start of the winter semester. This goal was achieved according to Gantt chart schedule (see Appendix A) as was detailed in the January progress report.

To summarize the contents of the progress report, we finalized the system-level design of the Performance Simulator by mid-November, which consists of 3 distinct components. These 3 components were the Frontend GUI, the Workflow Manager, and the MPE Backend. Detailed information on these components can be found in the Final Design and Individual Contributions sections.

Implementation and verification testing of the Performance Simulator was completed by mid January, with only a single test (simulator accuracy) unable to be verified until the second half of the project is completed (see Appendix A and System Level Overview). Regardless of the last unverified validation test, the Performance Simulator is fully functional, and ready for usage and demonstration purposes.

Following completion of the Performance Simulator (post progress report), we began work on the second part of our project: the MPI FPGA Application. For this application, we proceeded with our Molecular Dynamics (MD) design proposal and followed our Gantt chart schedule of implementing the software version with graphical output first, and then porting it over to use the HLS MPI FPGA hardware workflows.

The software implementation of the MD backend algorithms was completed by early February, and graphical visualization of atoms in MD was implemented by mid February (see Appendix A and System Level Overview). Thus, the software version is ready for demo purposes and was completed according to Gantt chart schedule.

Work then began on porting over software MD to use the HLS MPI FPGA hardware workflows that enable the application to be compiled for use on our supervisor's Heterogeneous FPGA network to achieve hardware acceleration. However, as of the writing of this report, this task is still incomplete due to a variety of technical issues with the workflows not working properly (compilation and bitstream loadings errors), which are outside the scope of our knowledge. Thus, we are working closely with our supervisor's graduate students to resolve the issues, however delays in communication are bottlenecking the work that we can do until the workflow issues are resolved.

Thus, for the final phase of our project, we are behind schedule with porting over to hardware MD, and this is having a cascading effect on our later tasks and milestones. Furthermore, we have no clear timeline on when these issues can be resolved, and thus future milestones are at risk. However, previous milestones and deliverables have been completed on schedule, and are ready for the design fair.

Individual Contributions

Zhaotong (Tony) Zheng

Background

As mentioned previously in the *Group Highlights* section, our capstone project is divided into two major components to be done in two distinct timeframes:

1. FPGA MPI Performance Simulator (September 2018 - December 2018)
2. MPI FPGA Application (January 2019 - April 2019)

The first part: the FPGA MPI Performance Simulator and the corresponding work I was responsible for and completed was detailed in the January individual progress report. As a quick recap of that progress report though, I was primarily responsible for the Frontend GUI design and its implementation for the Fall semester in regards to the Performance Simulator. That work was completed according to schedule in late december, and full module integration of the different components to form the Performance Simulator was completed according to schedule in early January. Further details on the Performance Simulator can be found in the *Final Design* section. Thus, for the first phase of our project, I completed all tasks according to schedule and contributed towards delivering a finished end product.

Following the completion of the first phase of our project in early January and completing the January progress reports, my attention turned towards the next set of required deliverables: the Oral Presentation, and the second phase of our project; the MPI FPGA application.

For the Oral Presentation, I was the major driver in getting discussions about the task started and organizing and designing the overall flow and structure of the presentation. I took into account all the feedback from the practice presentation to completely redesign the powerpoint design/theme, structure, and flow to better suit the requirements. Ultimately, this led to a well rehearsed and successful presentation in early February.

In regards to the MPI FPGA Application, I contributed early on in January and early February in helping to finalize the design for our MD application, and researching alternative open source library methods of visualizing the molecular dynamics information. However, starting from mid February and ongoing, I was faced with an unexpected family health emergency that has unfortunately required most of my time and attention. Thus, I have not been able to contribute towards the project since the start of this emergency. However, from the time of writing of this report, I am being re-integrated back into the daily workings of the team and can hopefully begin contributing again very soon.

Igor Stempen

My most significant contributions to this capstone project were the creation of the Back Propagator module in the Performance Simulator, and the initial research and development of the software version of our Molecular Dynamics Benchmark application.

The Back Propagator module initially described in the *Final Design: System Level Overview* section, is the most important component of our Performance simulator. It fulfills the ultimate goal that our simulator is expected to provide: giving the user an estimate of their theoretical performance for an arbitrary FPGA/network hardware specification. I took on the challenge of developing the initial backpropagation algorithm. This entailed determining the optimal data-structure for storing our log information (a graph structure was chosen), deciding how to transform MPE log information into graph form, and implementing an iterative algorithm that would modify the graph to represent the estimated performance record of the user's chosen platform definition. The research, design, coding, and verification was done almost entirely by me, and ultimately our team was able to deliver a functional simulator that impressed our supervisor. To add to this, early in the development I realized that the tool we were using to generate logging information for the user's MPI program did not output human-readable log files. Thus, I added new code to the open source tool MPE that would transform these log files to a human-readable/parseable format. These are the the Text Log Generator and Slog Generator modules in the *Final Design: System Level Overview* section.

Another very important contribution that I made was the initial research and development of the software version of our Molecular Dynamics benchmark application. Before we could proceed to create a hardware benchmark, a software version first needed to be created. I created the initial design of the application based on industry research. For the final implementation, I scaled down the design based on previous demos created by our supervisor's graduate students. The final Implementation was an extended (code added) version of said demo, with tweaks, polish and initial integration with an industry standard visualization tool called VMD. Having learned much about our supervisor's platform from my work on this application, I also spearheaded the drive to port this this application to run on our supervisor's platform. Due to the issues we faced when attempting to use our supervisor's platform (which are described in more detail in the *Final Design: Assessment of Final Design* section), I performed a lot of initial debugging and troubleshooting. Although we ultimately could not run our MD benchmark on our supervisor's platform at the time of writing this report, the debugging work I provided to our supervisor's graduate students ultimately helped improve the state and quality of our supervisor's platform.

Finally, there are also a few minor accomplishments for which I was responsible for that were critical for our capstone. These include: the initial setup of our virtual development environment, researching the capabilities of MPE, and managing our source control system.

Carlos Buenconsejo

My significant contributions to this capstone project consists of designing and developing the workflow manager module in the Performance Simulator, and providing development and debugging support for the synthesized hardware version of the Molecular Dynamics application on our supervisor's platform.

The technical details of the Workflow Manager module in the Performance simulator can be seen in the *Final Design* section. The Workflow Manager is the script that initiates the entire simulation process once a user clicks "Start Simulation" in the GUI. It ties together all the modules required for simulation and acts acts as the main connection between the Frontend GUI and the Back Propagator module. Creating the Workflow Manager required consulting with my team members who developed the GUI and Back Propagator to find the most convenient method of passing and receiving data in our code. Through this, I designed and implemented the internal interface that allowed for coherent data and parameter transfer between the two modules. Since the Workflow Manager initiates and controls the simulation process, I had to implement methods to create and manage files and directories which were passed between several modules. This resulted in a clean workflow where required files were automatically generated and cleaned up once simulation was complete. Finally, the Workflow Manager enabled for saving and loading a user's configuration of the GUI to disk. This required researching methods of saving settings, which resulted in our application using .cfg files. This method creates human readable configuration files for easy manipulation and interpretation.

My other main contribution was providing development and debugging support for the hardware version of the Molecular Dynamics application. With Igor leading the initial drive to port the software into our supervisor's platform, he was plagued with many software issues which hindered our progress. Since we were working with a new framework called Galapagos that was developed by our supervisor's graduate students, we were very reliant on them in order to help fix issues we were having. Communication between us and the graduate students was also very slow, so I tried to take initiative in debugging the problems. This allowed us to fix some issues but ultimately we were still hindered by software issues that were outside of our scope. Additionally, I was required to implement a benchmarking application to automatically determine characteristics of the platform such as network speeds, but since we were unable to run any program on our supervisor's platform this was deemed unfeasible.

Some other minor contributions I have made to the project were to the software Molecular Dynamics application. This was through writing a script that initializes VMD with the necessary settings in order to automatically load and be able to view files that describe molecular animation. I also researched and found a compatible file format used (.xyz files) in order to take the output data from the MD application and output it into VMD.

Acknowledgements

Without the help we received from a number of amazing people, this project would not have been possible. We would like to dedicate this section to thanking the following people:

Prof. Paul Chow for giving us the opportunity to work on this project to expand our knowledge and interests in the realm of heterogeneous FPGA computing. His guidance and determined patience were invaluable in steering us towards successful design solutions.

Nicholas Burgwin for providing feedback and guidance during the design review, project proposals, and oral presentation. Incorporating this feedback allowed us to greatly improve our documents and presentation. His help was greatly appreciated.

Nariman Eskandari for his initial guidance during the early phases of our project, which helped us in creating a realistic scope for our project. And also for the vast amount work he did during his Master's thesis, which we have used to form the backbone of our MPI FPGA application component.

Naif Tarafdar for the vast amount of help he has provided us in trying to set up, compile, and use Galapagos (the Heterogeneous FPGA network). Without his help, we would be lost for the second phase of our project.

Varun Sharma for his help in setting up our development environment so that we can use Galapagos and the Vivado CAD suite.

Clark Qianfeng Shen for helping us set up, compile, and use Galapagos with the Sidewinder FPGA board. Without his help, we would also be lost.

Table of Contents

Introduction (Author: Carlos Buenconsejo)

This report summarizes the motivation, design, implementation, and testing of an application that predicts the performance of synthesized hardware on an FPGA compared to its software version. Additionally, we will summarize the challenges and work done on using our supervisor's "Galapagos" platform, a full stack framework that enables FPGA cloud computing. The report concludes with suggestions of improvements and future work.

Background and Motivation (Author: Everyone)

The heterogeneous cloud computing industry is currently dominated by CPUs and GPUs [1], with extensive software support allowing GPUs to act as hardware accelerators. However, due to the reconfigurability, low latency, and power efficiency of Field Programmable Gate Arrays (FPGA), many organizations want to leverage FPGA technologies instead of GPUs as software accelerators in their cloud computing platforms [2]. The versatility of FPGAs stems from their design; they are composed of a large array of programmable digital logic blocks, allowing users to reconfigure them to replicate any digital logic circuit [3]. Unfortunately, since readily available FPGA cloud computing services have only recently been made available to the general public, many organizations do not have sophisticated software support for FPGA based cloud computing [4].

The University of Toronto (UofT) is currently in the process of implementing a dedicated FPGA cloud computing environment. They have recently developed a software interface for FPGA cloud computing using a framework known as "Message Passing Interface" (MPI) [5]. This framework allows computing tasks running on FPGAs and CPUs to pass messages amongst each other in order to synchronize between themselves. As a result, specific software processes of a very complex distributed parallel program can be synthesized into hardware and executed on an FPGA. Their computed data is then passed to other processes running on either FPGAs or CPUs using MPI. This framework is called Galapagos and it is UofT's method to accelerate software applications in the cloud using FPGAs.

Software can be transformed (compiled) into hardware for use on an FPGA using an automated design process known as High Level Synthesis (HLS). HLS algorithms are provided by many prominent FPGA vendors to allow software developers to leverage the performance increases of hardware [6][7]. The UofT FPGA cloud leverages HLS to allow a subset of a program's MPI processes to be synthesized into hardware and then accelerated on FPGAs. Unfortunately, existing HLS tools are not sophisticated enough to generate hardware circuits that can consistently beat the performance of well optimized MPI software processes running on CPUs. Additionally, communication between MPI processes in the FPGA cloud will have different behavior than their CPU counterparts due to differences in the network parameters and structure. As a result, many users of the FPGA cloud might prematurely spend time tweaking and structuring their MPI programs to work on the MPI FPGA cloud, only to find out later that unforeseen bottlenecks in network communications or nonoptimal HLS synthesized circuits could hinder the possibility of any acceleration.

In order to increase workflow efficiency and speed up optimization tasks, our supervisor wants a simulation sandbox tool that can predict the performance (overall latency) of a an MPI program if it is accelerated using the MPI FPGA cloud. Fundamentally, this simulator will determine the performance of a normal MPI program running on software, and estimate how much faster or slower it will be when it runs on the MPI FPGA cloud. Based on this performance value, users of the MPI FPGA cloud can determine the feasibility of accelerating their program with this infrastructure. Our simulation tool will report this performance value to the user based on inputs such as the latency and throughput of the network connections between FPGAs, the user's overall process topology (which processes run on FPGAs or CPUs), and the latencies of individual MPI processes.

After implementing this simulation tool, our supervisor would like us to demonstrate its functionality on an application that uses the MPI FPGA cloud. After using our simulator to receive a performance value for our hardware kernels, we should implement the application in the FPGA cloud and attempt to reach an expected overall performance that matches a pure software implementation. This is important for our supervisor, as it would not only prove the correctness of our simulator, but it could also potentially prove that their MPI FPGA cloud can provide value as a heterogeneous computing platform. However, our supervisor has emphasized that achieving performance speedup using the MPI FPGA cloud will be very difficult due to its complexity and immaturity.

Project Goal (Author: Everyone)

The goal of this project is to implement a simulator for UofT's MPI FPGA Cloud. This tool will estimate the performance of an MPI program when it is run using the MPI FPGA platform based on factors such as network parameters of the platform, the MPI program's process topology and the latencies of individual MPI processes.

Upon completion of the simulator mentioned above, an MPI FPGA cloud application should be created that demonstrates functional correctness of our simulator and attempts to achieve a performance result that is at least equal to the performance of an equivalent software implementation.

Project Requirements (Author: Everyone)

Table 1. List of project requirements and their descriptions

ID	Project Requirement	Description
1.0	Create MPI FPGA cloud simulator for performance measurements	**Primary Functional Requirement:** Create software simulator that can predict the performance of an MPI program running in the MPI FPGA cloud environment. It will interface with blocks or units that provide process topology, network parameters and individual MPI process latencies.

2.0	Create user interface for simulator	**Primary Functional Requirement:** The simulator should provide the user with an interface for inputting the process topology and network characteristics. It also needs to relay the simulation output to the user.
3.0	Minimize user time spent using UI	**Objective:** The user should spend a minimal amount of time configuring the simulator and interpreting simulation results. It should take no more than 5 minutes for a user to configure their simulation parameters, and no more than 15 minutes to interpret the results of the simulation. The numbers were chosen because they are small relative to simulation time but not unachievable.
4.0	Simulation needs to complete quickly	**Objective:** Since the simulator will need to run the user's MPI software code to assess performance, the simulation should not add any human-perceivable overhead time to its runtime.
5.0	Create software-based MPI application	**Primary functional requirement:** In order to benchmark the performance of the MPI FPGA cloud and verify the functionality of our simulator, we must first create an MPI application entirely in software and measure its performance without FPGA hardware acceleration. This application should have the potential to be accelerated in hardware. Meaning it must not have an already optimal solution in software. There are certain bottlenecks in software applications that cannot be resolved unless implemented in hardware (for example, networked software applications, memory intensive applications).
6.0	Port software-based MPI application to use FPGA hardware acceleration (HLS) and the MPI FPGA cloud	**Primary functional requirement:** We need to create an application using the MPI FPGA cloud infrastructure that will be at least as fast as an equivalent software implementation. It should also demonstrate the accuracy of our simulator. This application will effectively use the same code as our software version but will link to different MPI libraries.
7.0	FPGA MPI cloud application should be faster than equivalent software implementation	**Objective:** Ideally, our FPGA MPI cloud application should be faster than its equivalent software implementation in order to show that meaningful acceleration can be achieved using UofT's MPI FPGA cloud. A run time speed up of at least 5% is enough to show an indisputable performance increase. Applications that run on the FPGA MPI cloud usually take several minutes or hours to complete. Therefore, a 5% increase in speedup is significant.
8.0	Simulator should be able to support MPI applications using the MPI_send and	**Constraint:** At a minimum our simulator should be able to handle any MPI applications using the two fundamental MPI library calls (MPI_send and MPI_recv). Since other MPI functions are just extensions of these two calls, it will be sufficient to support just MPI_send and MPI_recv

	MPI_recv functions	
9.0	Create benchmarking tool for determining network parameters of MPI FPGA platform	**Objective:** If there is a change to the characteristics of the MPI FPGA cloud platform (such as a change in network properties), a tool is required that can quickly benchmark these new characteristics and update the simulator. This tool is optional because our supervisor is able to manually determine a characteristic change of our platform.
10.0	Simulator tool must function in the MPI FPGA cloud server environment	**Constraint:** Our supervisor interfaces with the MPI FPGA cloud platform using several specific server machines. We need to ensure that the simulator tool functions on these machines.
11.0	Simulator should be accurate to a certain range	**Constraint:** Our simulator should report to within 10% accuracy of real measurements. This number was chosen because 10% is not unrealistic to achieve, but a lower tolerance would be outside of the scope of this project. According to our supervisor, a more detailed analysis of performance could be considered a Master's Thesis.
12.0	GUI should be able to save its existing state to a config file	**Objective:** The GUI should be able to save its existing state; defined as all the user inputted values to a config file
13.0	GUI should be able to load a saved state from a config file	**Objective:** The GUI should be able to load in an existing state from a config file, and then restore itself to said state (defined as the prior user inputted values saved in the config file)
14.0	Workflow Manager should generate a .cfg file to save user input	**Objective:** The Workflow Manager should create a configuration file when the GUI calls its save function with user inputted values.
15.0	Workflow Manager should return user inputted values that were saved to a .cfg file	**Objective:** The Workflow Manager should be able to fetch the user inputted values that were saved to a configuration file and return them to the GUI in the same format as it was passed.
16.0	Simulator tool needs to convert from .slog file format to text log format	**Objective:** The simulator tool should convert .slog files to text log files for easy manipulation.

Final Design (Author: Ihor Stempen and Carlos Buenconsejo)

System-Level Overview

Our project is made up of two independent components: the performance simulator, and the MPI FPGA cloud application. Both of these have their own system level overviews.

Performance Simulator (Author: Ihor Stempen)
The performance simulator is centered around the functionality of an open source software called MPI Parallel Environment (MPE)[8]. MPE is able to log the timestamps of MPI communications for software programs. It is also capable of providing visualization tools for the aforementioned log files. Due to its open source nature, we are able to modify MPE to provide additional features if necessary. The system level diagram below shows which functionality in the simulator is entirely developed by us, which comes from open source software, and which is extended from open source software.

The overall goal of the simulator is to provide the user of our supervisor's MPI FPGA infrastructure the ability to estimate the performance of their accelerated MPI programs under various different platform characteristics (ie. different network speeds, process speeds, etc). The mechanism used to meet this goal is to use MPE to generate performance tracking log files for the user's program, and modify them to mimic how the program could have theoretically ran on a different environment.

The simulator program flow will behave as follows. The full system overview is shown in figure 1:

1. The user will provide his MPI source files and MPI FPGA cloud platform parameters to the program through a graphical **User Interface**. After the user has provided all the necessary inputs, they may click a button to begin the simulation.
2. When starting the simulation, the user's source files and platform parameters are passed to the **workflow manager**. The workflow manager is responsible for running controlling the individual phases of the simulation process, and passing any necessary data between them.
3. The first phase of the workflow manager is to compile the user's source files using an open source program called MPE. One of the features of the **MPE Toolkit** is the ability to add instrumentation code to a compiled MPI program to log any MPI events that occur. After being compiled with MPE, the program is executed, and a log file containing the timestamps of MPE events is outputted. This log file is passed to the next phase.
4. Unfortunately, the log file outputted by the instrumented executable is in a non human-readable file format (called .slog). In order to be able to process this log file, the log file must be converted into a readable format. The **Text Log Generator** extends the functionality of the MPE toolkit to parse the unreadable .slog file, and transform it into a human-readable/parseable file (named .textlog).
5. The final platform parameters and readable textlog are passed to the **Back Propagator**. This module parses the textlog to create a graph data structure that represents the flow of all important events that occured in the user's MPI program. The Back Propagator then uses the

user's platform characteristics (ie. speed of network, speed of MPI processes) to make modifications to the graph to estimate the performance of the user program on their defined platform. The final graph is then converted back to a new .textlog file.

6. The new textlog is sent to the **Slog Generator** which effectively performs the inverse of the Text Log Generator. It extends the functionality of MPE to turn the readable textlog file back into a .slog file which can be interpreted by MPE for the final step of the simulation process: visualization.
7. The final .slog is passed to the **Visualization Tool** called Jumpshot. This tool is part of MPE, and it is able to visually show the contents of an slog file. At this stage, the user can see the impact of their platform characteristics on the their program. They can check the final performance, and see where a potential bottleneck exists.

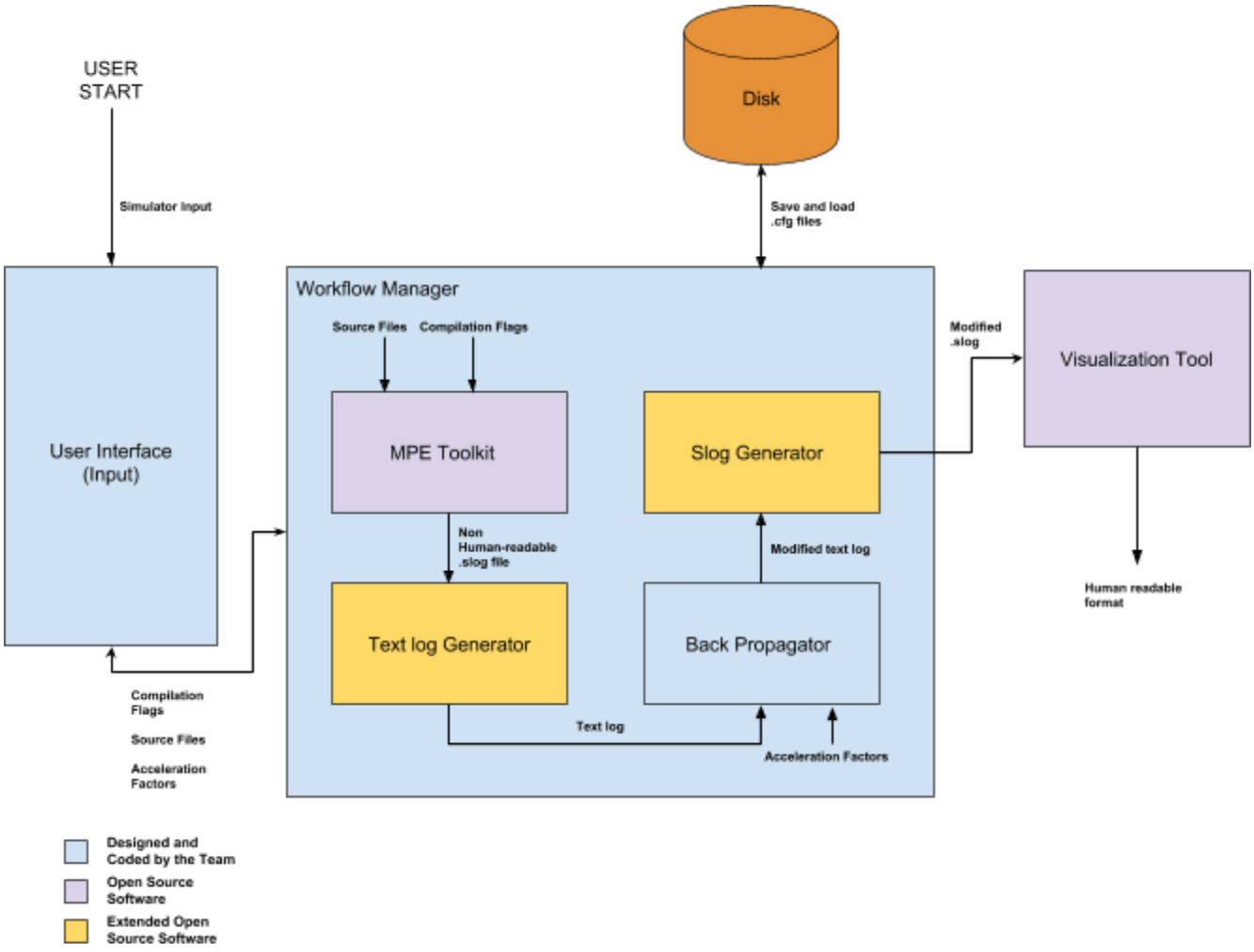


Figure 1. System Level Diagram for Simulator

MPI FPGA Cloud Application (Molecular Dynamics) (Author: Carlos Buenconsejo)

Our primary goal for the MPI FPGA cloud application is not to create the next state-of-the-art Molecular Dynamics application, but to demonstrate the success of our simulator and to attempt to demonstrate a performance speedup when using the MPI FPGA cloud. Thus, our algorithm will be a lightweight optimized version of the algorithm provided in *Heterogeneous, Purpose Built Computer Architecture For Accelerating Biomolecular Simulation* [9]. More specifically, the algorithm is reduced in scope such that we will only be calculating short-range electrostatic forces and neglecting long-range Van Der Waal forces. The purpose of Molecular Dynamics is to determine the positions, accelerations and velocities of the atoms/molecules in a chemical system. The simulation is done in several small (nanosecond) timesteps: in each timestep the positions, velocities, and accelerations of the atoms/molecules in the system are updated.

The algorithm for doing this is described below. The system level overview is shown in figure 2:

1. One **Atom Manager** initializes scene and is responsible for keeping track of the position of each atom in the simulation. At each timestep, the Atom Manager will collect the positions of all of the atoms based on force information from the compute engines and write the positions of each atom to a file in the .XYZ file format. This is a method to store atom positions in cartesian coordinates used by the Atom Visualizer. The Atom Manager is implemented in software as a single process that communicates to the Compute Engines over the network.
2. Each **Compute Engine** is responsible for computing the forces on all atoms within its physical area segment. Each compute engine will receive the initial coordinates, forces and velocities of the atoms from the Atom Manager. It will then calculate the electrostatic forces on each atom and return them to the atom manager. The Compute Engines are implemented in the FPGA cloud network as hardware kernels. There is an arbitrary number of compute engines.
3. The **Atom Visualizer** will take the positions of the atoms and display them through a graphical interface. The application used is Visual Molecular Dynamics (VMD) which is a tool used to view and analyze molecular simulations[10]. It does this by reading the XYZ file generated by the Atom Manager. This is done through a script that automatically loads and sets up the required settings to view the atoms.

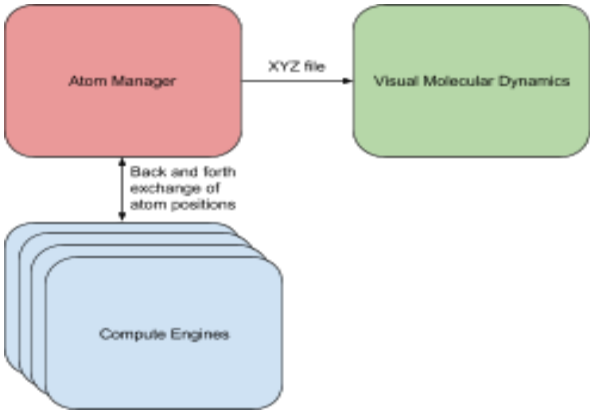


Figure 2. System Level Diagram For Molecular Dynamics

Module-Level Descriptions

Performance Simulator (Author: Ihor Stempen)

User Interface (Author: Zhaotong Zheng)

Inputs: ● MPI Application Source Code directory from user ● Application command line arguments from user ● MPI FPGA Cloud Platform Network Parameters from user ● MPI FPGA Cloud Platform Hardware Acceleration parameters from user ● Config file of prior saved GUI state
Outputs: ● All user inputted data to the workflow manager
Function: The user interface acts as the central control hub for user commands. It is split into two distinct UIs (see Figure 3 and Figure 4). In the starter GUI (Figure 3), the user can specify the number of different FPGA hardware ranks and the number of different network connections (of different speed) that they have. This will load the required number of input fields when the main GUI is launched. They can also load a specific GUI state saved into a config file from a prior run, or from a user generated config file. Choosing to load from a config file will transfer control to the workflow manager to parse the config file and then wait to receive parsed data from the Workflow Manager. In the main GUI (Figure 4), the user can input the source code directory through a file manager, enter the MPI FPGA Cloud Platform parameters, save the GUI state to a config file, and then start the simulation. Starting the simulation or saving to a config file will transfer user inputted data and program control to the Workflow Manager. See Appendix N for implementation details

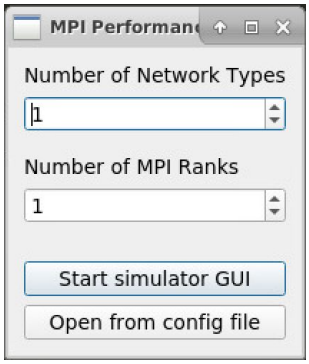


Figure 3. The Starter GUI

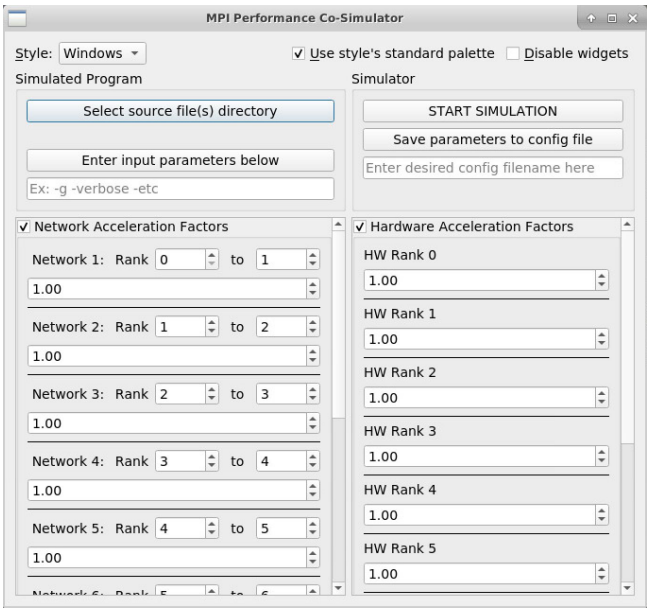


Figure 4. Main GUI

Workflow Manager (Author: Carlos Buenconsejo)

Inputs:

- Source file directory from GUI
- Compilation Flags from GUI
- Network and Hardware acceleration factors from GUI
- Start simulation call from GUI
- Save parameter call and config file name from GUI
- .cfg file from disk

Outputs:

- Source files and compilation flags to MPE Toolkit
- Modified .slog to Visualization tool
- .cfg file to disk
- All configuration settings back to GUI

Function: After receiving the call to start simulation, the Workflow Manager takes all the inputs and passes them to the necessary modules. It creates files and folders to manage the inputs and outputs for all modules that it encapsulates, which are automatically cleaned up. It then passes the final modified log files into the Visualization tool for user interpretation. The Workflow Manager also is responsible for loading and saving user settings.

MPE Toolkit (Author: Ihor Stempen)

Inputs:
- User's MPI program source files from Workflow Manager
- User's MPI program compilation flags from Workflow Manager

Outputs:
- Performance Log File in non human-readable file format (.slog) to Text Log Generator

Function: After receiving the user's source code and compilation flags, this module will compile and run the code through MPE. This produces a log file measuring performance of the MPI program[8]. This module is provided to us entirely by the MPE toolkit. An example of how to use the MPE toolkit to compile MPI programs with logging capability is shown below in figure 5. This log file will then be sent to the Text Log Generator to be transformed into human-readable format.

```
mpecc -mpilog -o <additional_compilation_flags> <executable_name> <source_files>
```

Figure 5. The following command uses MPE to compile the user's MPI program. The -mpilog flag instruments the program for log output

Text Log Generator (Author: Ihor Stempen)

Inputs:
- Non-human readable .slog file containing the performance log of the user's MPI program from the MPE Toolkit.

Outputs:
- Human-readable .textlog file containing the performance log of the user's MPI program to the Back Propagator.

Function: The Text Log Generator will convert a .slog file to a .textlog file. Both file formats contain the same information, a list of MPI events that occurred during the program runtime, the times these events occurred, and the processes that triggered these events. The .slog file format is unreadable to humans and can only be interpreted by MPE. The .textlog file format is designed by us, and is easily readable and parsable. This functionality is implemented by extending (adding on to) the existing features in the MPI toolkit. An example of the extended Java code can be found in Appendix E. An example of a .slog file and a .textlog file can be seen in figures 6 and 7 respectively.

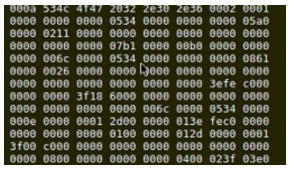


Figure 6. A .slog file opened in a text editor

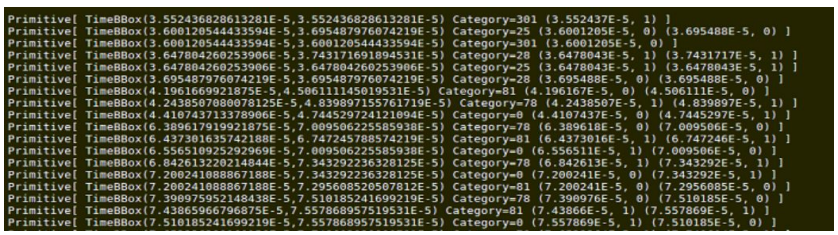


Figure 7. a .textlog file opened in a text editor

Back Propagator (Author: Ihor Stempen)

Inputs: ● Textlog file from the Text Log Generator ● The user's specified Platform Parameters from Workflow Manager
Outputs: ● Modified textlog File to Slog Generator
Function: The Back Propagator first creates a graph representation of the inputted textlog file. The events in the textlog file become nodes in the graph, and dependencies between different events (ie event B depends on event A) are the edges between nodes. Each node has a start and end time. Intuitively, a node's start time is equal to the maximum end time of all nodes that it depends on. An example of a created graph can be seen in figure 8. A more detailed description of how the graph is generated can be seen in Appendix D. Once this graph is created, the back-propagation algorithm can be run on it to modify the start/end times of the graph nodes to mimic how the program might have run in the user's specified environment. The Pseudo-code of the back-propagation algorithm can be seen in Figure 9. More details on how the algorithm works can be found in appendix D. Once the back-propagation algorithm is complete, the modified graph is converted back to a .textlog file and passed to the Slog Generator module. An example of backpropagation code can be seen in Appendix E.

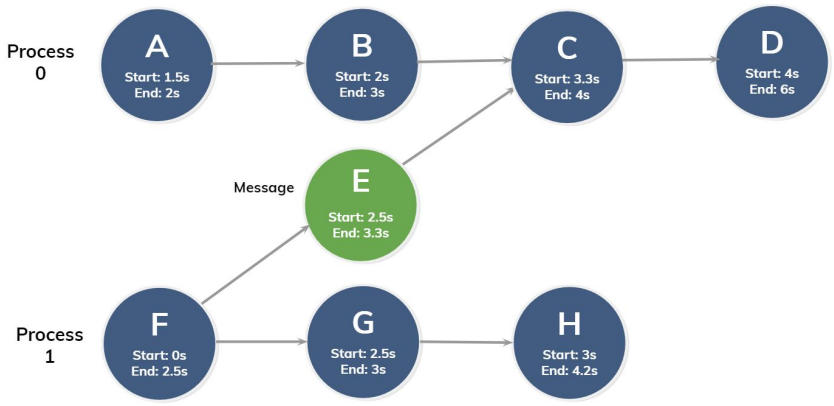


Figure 8. An example of a graph generated from a .textlog file.

```
1  for characteristic in users_platform_characteristics:
2      for node in graph_nodes:
3          if node.isAffectedBy(characteristic):
4              node.modifyStartAndEndTimes(characteristic)
5              node.propagateEndTimeChangeToDependentNodes() #RECURSIVE
6  DONE
```

Figure 9. A pseudo-code synopsis of the back-propagation algorithm. An important note is that the function call on line 5 will recursively update all nodes dependent on node.

Slog Generator (Author: Ihor Stempen)

Inputs: ● Modified .textlog file containing the simulated performance log of the user's MPI program from the Back Propagator.
Outputs: ● Modified .slog file containing the performance log of the user's MPI program to the Visualization tool.
Function: This module effectively performs the inverse of the Text Log Generator module. It transforms the 'simulated' .textlog file created by the Back Propagator and turns it into .slog file that the MPE toolkit can interpret. Just like the Text Log generator, this module extends the functionalities of the MPE toolkit.

Results Displayer (Author: Ihor Stempen)

Inputs: ● Modified .slog file containing the performance log of the user's MPI program from the Slog Generator
Outputs: ● Visualized simulation output to the user
Function: This module will take the simulated .slog log file and parse it in order to present the results to the user. This functionality is provided entirely by the MPE Toolkit via a tool called Jumpshot. Jumpshot visually describes the contents of the .slog/.textlog files from the previous steps. Here, the user can see their final estimated runtime, and detect any bottlenecks. An example of a Jumpshot session is shown in Figure 10.

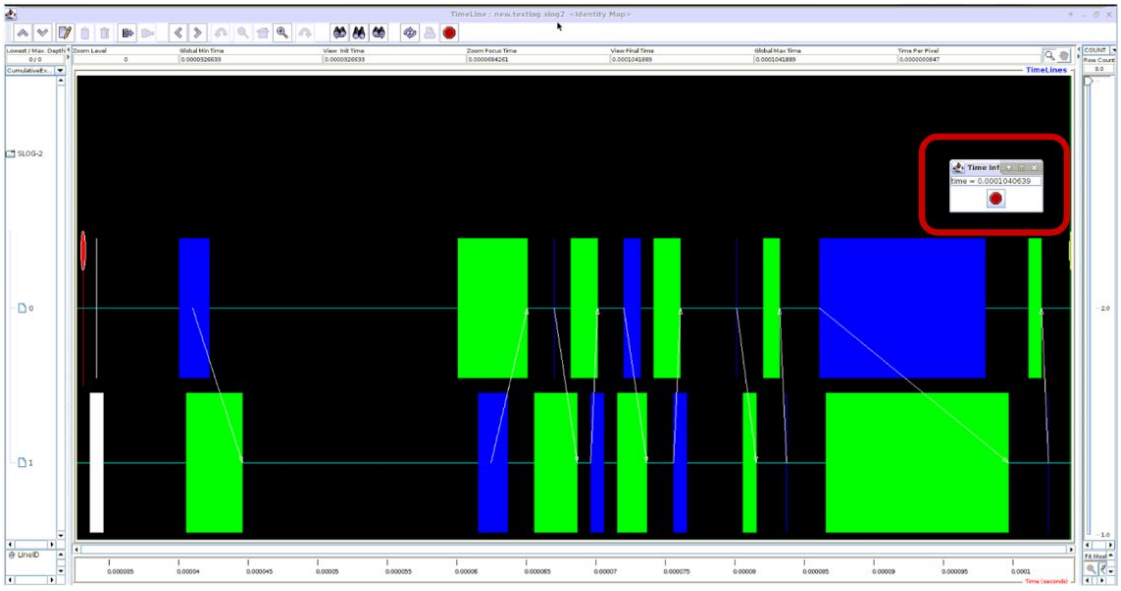


Figure 10. An example of a Jumpshot visualization at the end of the simulation flow. The final runtime appears in the box at the top right, which has been highlighted red here.

Molecular Dynamics (Author: Carlos Buenconsejo)

Atom Manager (Author: Carlos Buenconsejo)

Inputs: ● Initial conditions of the molecular system from user ● XYZ positions of each atom from Compute Engines
Outputs: ● Initial conditions of the molecular system to Compute Engines ● XYZ file with positions of every atom at each timestep
Function: The atom manager is responsible for collecting and keeping track of the positions of all atoms. It is not very compute intensive and is only concerned with collecting and writing these positions to the XYZ file format in order to be analyzed by VMD.

Compute Engines (Author: Carlos Buenconsejo)

Inputs: ● Initial conditions of the molecular system from the Atom Manager
Outputs: ● Positions of atoms in XYZ to the Atom Manager
Function: There is one compute engine per group of atoms which is determined dividing the atoms evenly between compute engines. The compute engine is responsible for calculating the forces on all atoms in its physical section. These forces are short range electrostatic forces. It uses these forces to calculate the updated positions of the atoms which are then sent to the Atom Manager. The force calculations are shown in Figure 11.

```
// Calc distance.
float dx = pos[i * 3] - pos[j * 3];
dx = dx - roundInt(dx/CUBELENGTH) * CUBELENGTH;
float dy = pos[i * 3 + 1] - pos[j * 3 + 1];
dy = dy - roundInt(dy/CUBELENGTH) * CUBELENGTH;
float dz = pos[i * 3 + 2] - pos[j * 3 + 2];
dz = dz - roundInt(dz/CUBELENGTH) * CUBELENGTH;
float r2inv = 1.0 / (dx*dx + dy*dy + dz*dz);
float r6inv = r2inv * r2inv * r2inv;
float r12inv = r6inv * r6inv;
float pe = (vdw * r12inv) - (vdw * r6inv);
float gr = ((12 * vdw * r2inv * r12inv) - 6 * vdw * r2inv * r6inv);
// Force
float fx = gr * dx;
float fy = gr * dy;
float fz = gr * dz;
// Sum forces.
force[i * 3] += fx;
force[i * 3 + 1] += fy;
force[i * 3 + 2] += fz;
force[j * 3] -= fx;
force[j * 3 + 1] -= fy;
force[j * 3 + 2] -= fz;
```

Figure 11. Code Snippet of compute engine force calculation

Atom Visualizer (Author: Carlos Buenconsejo)

Inputs: ● Position of all atoms in XYZ file format from the Atom Manager
Outputs: ● Visualization of the atoms at each timestep to user
Function: The atom visualizer provides a graphical output of all the atoms in the molecular system. It does this by reading the .XYZ file from the atom managers and displaying them. This is done at the end of simulation when the position of each atom at each timestep is known. An example of the visualization is shown in Figure 12 and a detailed description in appendix G.

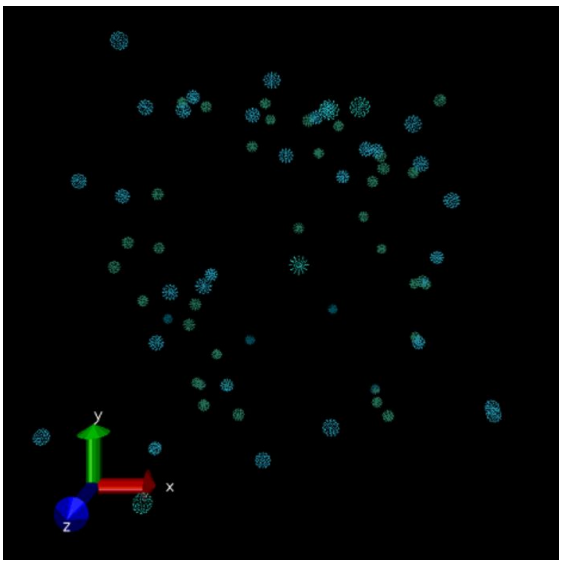


Figure 12. Example of VMD visualization. The blue dots are carbon atoms.

Assessment of Final Design

Performance Simulator (Author: Ihor Stempen)

While our Performance Simulator was unable to meet some of the objectives that we initially planned, it was ultimately able to provide the basic functionality that our supervisor envisioned. Unfortunately, our task to prove the accuracy of the performance simulator is currently blocked by our inability to use our supervisor's FPGA platform; this will be discussed in more detail in the *Assessment of Final Design: Molecular Dynamics* section. Consequently, it is difficult to discuss whether or not our simulator was ultimately successful because we cannot at this point concretely prove that it can accurately estimate the performance of an MPI program on an arbitrary platform. However, we can show that the simulator does provide results that are realistic, and can detect the existence of bottlenecks.

Although the exact details regarding the the status of our project requirements will be discussed in the *Testing and Verification: Final Test Results* section, this section will provide a summary of which functional requirements are currently met by our simulator design, and which ones are not:

Successes:

- With respect to functionality, our simulator successfully completes the flow envisioned by us in the project proposal (now refined and described in the *System Level Overview* section). From start to finish, our simulator can take a user's MPI source code and platform characteristics, and output a visual estimate of the performance of their program on the defined platform. This simulator supports the essential MPI_Send and MPI_Recv functions and runs flawlessly on our supervisor's virtual environment. (Functional Requirement IDs 1,8,10,16)
- Our simulator is supported by a sleek, easy-to-use graphical user interface. For input, a user can easily define their platform characteristics. In order to facilitate quick use, we added the option of loading the platform definition from a configuration file; this is particularly useful since our supervisor's platform specifications are unlikely to change very frequently. For output, the user can visually view where bottlenecks in their program occur, and which MPI processes are causing the most slowdown. (Functional Requirements IDs 2,3,12,14,15)

Known Deficiencies:

- Our chosen simulator design is inadequate to meet Functional Requirement ID #4. We had originally intended for the simulator to not add any human-perceivable overhead time to the runtime of the user's MPI application. Currently, the simulation takes much longer to complete than most complicated MPI programs. For an MPI program with thousands of messages being sent between processes, the simulator can take minutes to complete. This slowness can be attributed to the fact that the Back Propagator module uses recursion to propagate changes through the graph model. However, even an iterative algorithm would likely struggle for programs with thousands of messages, since the overall complexity of the backpropagation algorithm is proportional to $O(n^2)$, where n is the number of of MPI events in the program. This conclusion is reached as follows: for every MPI event in the program, an average of half of all MPI events must be looked at to propagate a performance change. Fortunately, most applications running on our supervisor's platform are very repetitive in nature, with most ranks performing a loop of repetitive events. Thus, only a small fraction of most programs needs to be simulated, thus **reducing the simulator runtime to meet our goal.**

Unknowns:

- As mentioned previously, issues related to using our supervisors platform prevented us from running our Molecular Dynamics benchmark application in hardware. Therefore, we are unable to determine the accuracy of our simulator. However, an analysis in Appendix F demonstrates that our simulator provides results that appear reasonable, and the simulator is able to accurately detect a bottleneck in the user's program. We are hopeful that we will have complete proof of accuracy ready for the final demonstration.

Molecular Dynamics (Author: Carlos Buenconsejo)

Unfortunately, we were unable to leverage our supervisor's platform in order to create the hardware accelerated version of our Molecular Dynamics Program. However, we do have a working software implementation that works as described in the System-Level Overview and Module-Level Description sections which can be seen in Appendix H. The only difference from our intended final solution is that the Compute Engines are implemented in software rather than synthesized as hardware on FPGAs. As a result, we are able to create a .XYZ file and view it using VMD where we see movements of atoms.

Our inability to port the software version of our Molecular Dynamics program to our supervisor's platform can be attributed to the platform being new and unpolished. Although there was documentation, it was outdated along with the code required to compile various files. Combined with the fact that we were also given a new model of FPGA board that had never been used before, we ran into many software issues that were out of our control. We had many efforts in trying to debug these issues along with lengthy email threads and meetings with the PHD student who developed the platform but ultimately we could not get it to work. An example of our communication efforts can be seen in Appendix L which represents about 3 weeks of debugging and development and Appendix M documents and shows a majority of the issues that we had encountered. In the end, we were able to compile everything and load it onto the FPGA but were unable to get the Atom Manager and Compute Engines to communicate over the network.

Successes:

- With our debugging and development efforts put into trying to port our software, we were able to test and provide feedback on our supervisor's FPGA cloud platform. We hope that as a result, future individuals interested in using this platform will be able to go through the workflow required to successfully use the platform. Also as mentioned earlier, we do have a working software implementation of MD that does what we designed for. This aligns with ID 5 in the requirements table, but more importantly we are able to run this program in our Simulator which is described in the next section.

Deficiencies:

- Since we were unable to port our software MD application to our supervisor's platform, this directly fails the primary requirement with ID 6 in the requirements table. As a result, we also don't meet the objectives referenced by IDs 7 and 9. These are that the hardware version of MD should be faster than software, and creating a benchmark program to automatically determine the platform's characteristics, respectively. Since there is no working version of the MD on our supervisor's platform, we can't measure its performance to compare it to software, and since we are not able to get the platform working we can't create a benchmark program.

Testing and Verification

Verification Table (Author: Everyone)

Note: See Appendix C for explanation of verification tests

Table 2: Verification Results for verification tests

Change?	ID	Project Requirement	Verification Results And Proof	Requirement Verification Method			
				Similarity	Review of Design	Analysis	Test
	1.0	Create MPI FPGA cloud simulator for performance measurements	**PASS**. See Appendix F for simulation running on ping-ping program with different acceleration values				**X**
	2.0	Create user interface for simulator	**PASS**. See *Final Design: Module level descriptions: User Interface*.		**X**		
	3.0	Minimize user time spent using UI	**PASS**. See Appendix O for graduate student's approval.				**X**
	4.0	Simulation needs to complete quickly	**FAIL**. Explained in *Final Design: Assessment of Final Design* section pertaining to performance simulator				**X**
MODIFIED	5.0	Create software-based MPI application	**PASS**. Atom Visualizations shown in Appendix G, and code snippet shown in appendix H.				**X**
	6.0	Port software-based MPI application to use FPGA hardware acceleration (HLS) and the MPI FPGA cloud	**FAIL**. Reasoning discussed in detail in the *Final Design: Assessment of Final Design* section pertaining to Molecular Dynamics.				**X**
	7.0	FPGA MPI cloud application should	**FAIL**. Since Project Requirement 6.0 could not				

		be faster than equivalent software implementation	be completed, this project requirement can not be completed.				**X**
	8.0	Simulator should be able to support MPI applications using the MPI_send and MPI_recv functions	**PASS**. MPI application with MPI_send and MPI_Recv calls shown in Appendix F. The source code for the program is shown with Appendix I.				**X**
	9.0	Create benchmarking tool for determining network parameters of MPI FPGA platform	**FAIL**. Since we were not able to achieve success for Project Requirement 6, this related optional requirement was not attempted.				**X**
	10.0	Simulator tool must function in the MPI FPGA cloud server environment	**PASS**. Images of Simulator working in cloud server environment shown in Appendix F.		**X**		
	11.0	Simulator should be accurate to a certain range	**PARTIAL FAIL**. Since we could not get our MD design to work in HW, there is no way to prove that our simulator is able to accurately estimate the performance of designs on our supervisor's platform. This is discussed in detail in *Final Design: Assessment of Final Design section.* However, a partial proof of correctness is shown in Appendix F.				**X**
	12.0	GUI should be able to save its existing state to a config file	**PASS**. See Appendix P.		**X**		
	13.0	GUI should be able to load a	**PASS**. See Appendix P.				

		saved state from a config file			**X**		
	14.0	Workflow Manager should save user input from GUI as .cfg files	**PASS.** Once the user clicks save parameters to config file with a given name, a .cfg file is created in the directory. See Appendix J.				**X**
	15.0	Workflow Manager should pass data from the .cfg files to the GUI in the same format for loading	**PASS.** When a load is initiated from the GUI, the settings are read from disk and passed to the GUI using the given software interface in Appendix K.				**X**
	16.0	.slog to text log module should work	**PASS**. Images of unreadable .slog and readable .textlog shown in the *Final Design: Module Level Descriptions: Text Log Generator* section.				**X**

Final Test Results (Author: Ihor Stempen)

Ultimately, we were able to run our software MD application through our Performance Simulator, and collect performance results for a hypothetical hardware platform configuration. If we are eventually able to run MD in hardware, than it would be trivial to simply time the runtime of the application and compare it to our simulation prediction to check for accuracy and conclude our capstone. Figure 13 below shows a software MD application with a single compute engine rank (process 0) and single atom manager rank (process 1). The simulation runs only for three time steps in order to save on time, but since MD is very repetitive, the runtime shown here can be interpolated for runs with more timesteps. The simulation contains 72 atoms. Figure 13 shows the application exactly as it ran in software. On the other hand, figure 14 shows our simulator attempting to predict how long it would take for this same application to run on a hypothetical hardware platform similar to our supervisor's platform. For this simulation, we assumed the following hardware configuration:

- A network 5 times slower than software communication over shared memory. This is a reasonable assumption since shared memory communication is very fast.
- The atom manager runs in software, and so, does not get any speedup/slowdown
- A compute engine that is 3 times faster since it is being accelerated on an FPGA

The results show that the software variant runs for approximately 1.46 milliseconds, while the hardware variant is estimated to run for approximately 5 milliseconds. This shows that the hypothetical hardware platform that we are estimating for would be **slower than software** for MD. The

reason for this is likely that the speedup gained by running the compute engine in hardware is offset by the sheer quantity of data (atom forces) that need to be sent over a much slower network. A partial-proof of the correctness of these results is shown in Appendix F. **If we ultimately do create a hardware version of our MD application, the final step is to time the application's performance, and compare against these results.**

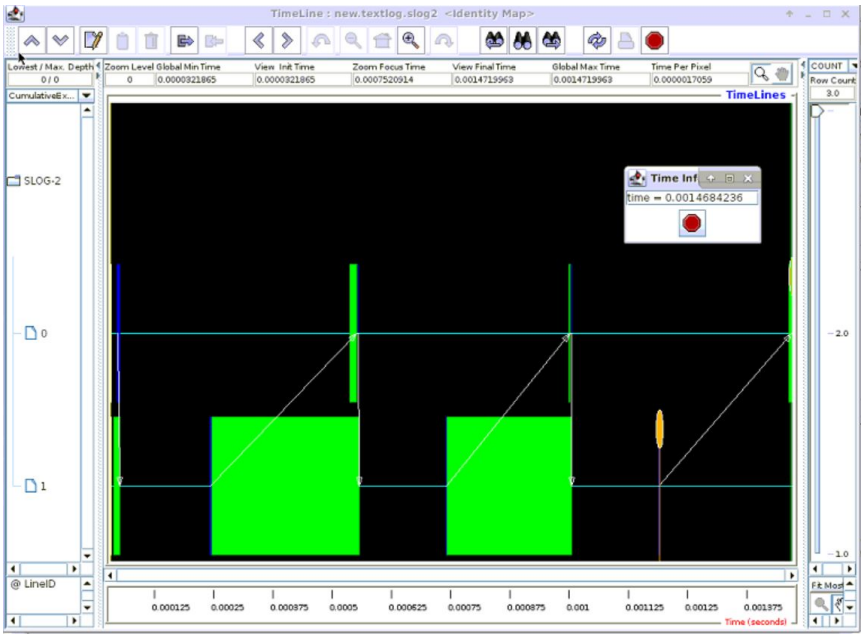


Figure 13. Simulation of 3 steps of software MD with all acceleration factors set to 1 (ie. this shows the execution of software MD in software). Runtime of 1.46ms.

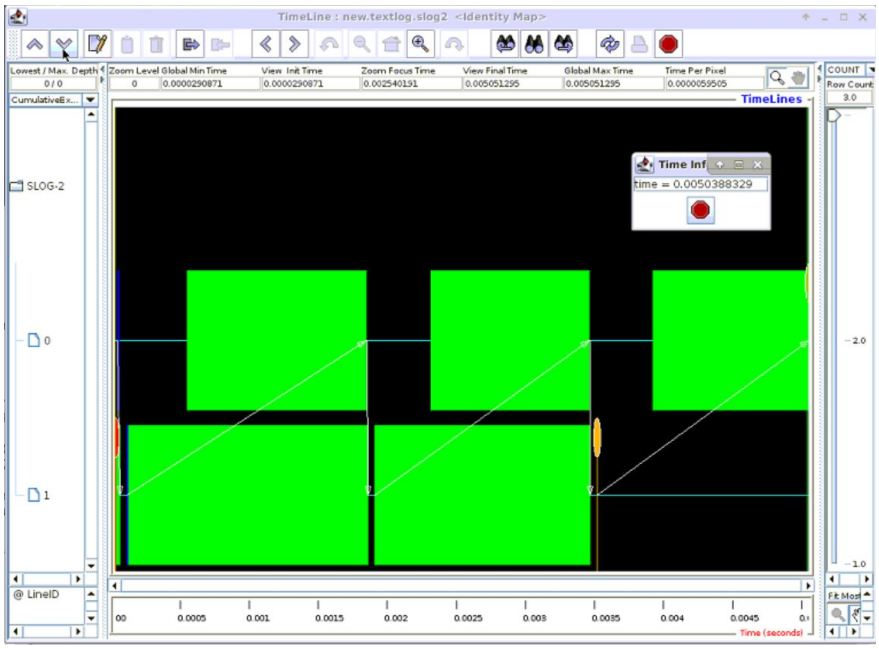


Figure 14. Simulation of 3 steps of software MD with the network set to be 5 times slower, and the compute engine kernel set to be 3 times faster (ie. this shows the execution of software MD in a hypothetical hardware environment). Runtime of 5ms.

Summary and Conclusions (Author: Zhaotong Zheng)

Driven by increasing bottlenecks in homogeneous CPU only systems, state of the art research is currently being done in industry on heterogeneous cloud computing, in which general purpose CPUs are being paired with FPGAs to automatically accelerate software workflows in hardware. Our supervisor Professor Paul Chow of UofT is one of such groups researching this technology.

Heterogeneous computing however is still in its early phases and is an immature technology. As a result, our supervisor is facing a number issues in the development of their heterogeneous computing FPGA platform. Their primary concern is the inability for users to know beforehand the potential extent of their performance gains on a software program when using a heterogeneous FPGA computing platform. In other words, it is difficult for users to determine what parts of their code is bottlenecking their potential for distributed hardware acceleration. Thus, our supervisor tasked us with developing a simulator for their platform to solve this user problem, and to then develop an MPI application to be run on their FPGA platform to prove the functionality and accuracy of our simulator and their platform.

In summary of the design and verification work we accomplished, we managed to develop a fully functional simulator program, as well as a fully functional software implementation of an MPI application based on Molecular Dynamics (MD). However, we were unable to meet all project requirements. Specifically, due to our inability to implement a hardware implementation of the MPI application (MD) to be run on the FPGA platform, we were unable to prove the accuracy of our simulator; which was a key requirement for our project. Implementing an MPI application in hardware was also a project requirement in itself, and thus we failed to meet that as well.

Our failure to meet requirements was not from a lack of effort on anyone's part though. The issues we faced in trying to get the FPGA platform to work (compilation, loading programs onto FPGA, network communications, etc) were simply outside the scope of our project and knowledge, and can be attributed to the novelty of our supervisor's FPGA platform. To try and solve these issues, we worked hard with our supervisor's team of graduate students, and they provided immense guidance and help but we still could not resolve the problems by the time of this final report.

As a result of not meeting all project requirements, we cannot call our project a success. However, given the functional programs that we have delivered, it would be shortsighted to call our project a complete failure. From this more optimistic point of view, we were successful in delivering a functional simulator that passed most of our tests, and outputs reasonably accurate albeit unofficially tested results. Additionally, we successfully implemented software MD with full visualization, and thus, we have delivered 2 out of 3 deliverables that are ready for usage and demonstration purposes.

Despite our shortcomings in developing a hardware capable MPI application, our testing and verification was at least exhaustive enough to prove our initial design ideas in terms of the simulator. As mentioned in the Final Test Results section and Appendix F, we were able to prove in a roundabout way through software that our simulator is capable of demonstrating an MPI application's overall

performance profile. Then, by analyzing the performance profiles, the user is capable of determining bottlenecks in their program in regards to distributed FPGA hardware acceleration potential. This was demonstrated through our software MD application being run through the simulator.

In conclusion, we were unable to realize all of our initial design goals and requirements. However, we have laid the foundation of what can be a useful tool for helping users to write software applications with high potential for hardware acceleration on a distributed FPGA computing platform. Further work in the future can be done to prove (or disprove) the accuracy of our simulator by resolving the issues in running our MD application on the Galapagos platform. Our simulator can also be further extended to include automated analysis of performance profiles to help the user in finding bottlenecks in their program. All things considered, we may not have perfectly succeeded in our project, but we have provided our supervisor a foundation that may prove to be useful in his future work, and valuable debugging and development experience on his state of the art distributed FPGA computing platform.

References

[1] NVIDIA, “NVIDIA GPU CLOUD TENSORFLOW,” *nvidia.com*. [Online]. Available: https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/tensorflow/. [Accessed: 04-Oct-2018].

[2] Intel, “Intel® FPGA SDK for OpenCL™,” *intel.com*. [Online]. Available: https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html. [Accessed: 04-Oct-2018].

[3] “Field-programmable gate array,” *Wikipedia.com*, 02-Oct-2018. [Online]. Available: https://en.wikipedia.org/wiki/Field-programmable_gate_array. [Accessed: 04-Oct-2018].

[4] Amazon, “Amazon EC2 F1 Instances,” *Amazon.com*. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/. [Accessed: 04-Oct-2018].

[5] W. Gropp, E. L. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. Cambridge (Mass.): MIT Press, 2014.

[6] Xilinx, “Vivado High-Level Synthesis,” *Xilinx.com*. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html. [Accessed: 04-Oct-2018].

[7] “High-level synthesis,” *Wikipedia*, 25-Sep-2018. [Online]. Available: https://en.wikipedia.org/wiki/High-level_synthesis. [Accessed: 04-Oct-2018].

[8] W. Gropp, E. L. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. Cambridge (Mass.): MIT Press, 2014.

[9] C. Madill, A. Patel, M. Saldaña, P. Chow and R. Pomès, "A Heterogeneous, Purpose Built Computer Architecture for Accelerating Biomolecular Simulation", *Biophysical Journal*, 02-Feb-2011. [Online]. Available: https://www.cell.com/biophysj/fulltext/S0006-3495(10)02524-5. [Accessed: 15-Oct-2018]

[10]“VMD - Visual Molecular Dynamics,” Theoretical Biophysics Group. [Online]. Available: https://www.ks.uiuc.edu/Research/vmd/. [Accessed: 21-Mar-2019].

Appendices

Appendix A - Gantt Chart History

Blue = Scheduled **Yellow** = Behind Schedule **Green** = Completed

Red = Behind Schedule - At Major Risk **Orange** = Scheduled - At Risk **Black** = Cancelled

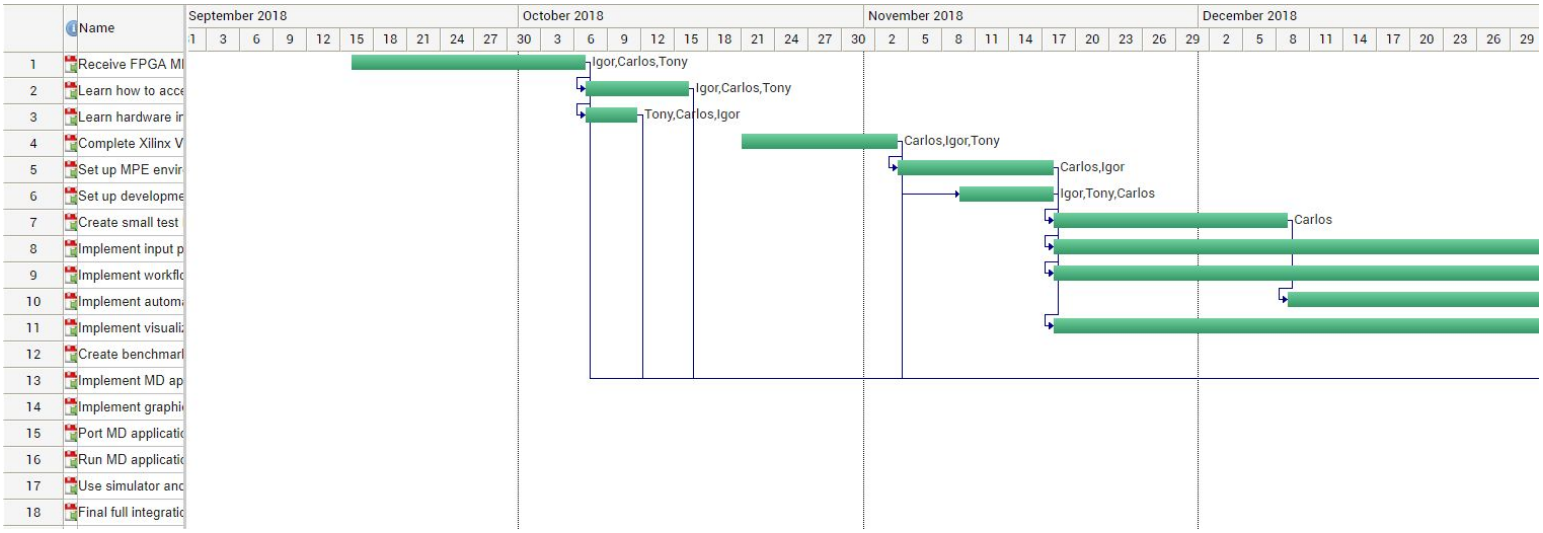


Fig. A.1 Final Report Gantt Chart - Fall Semester

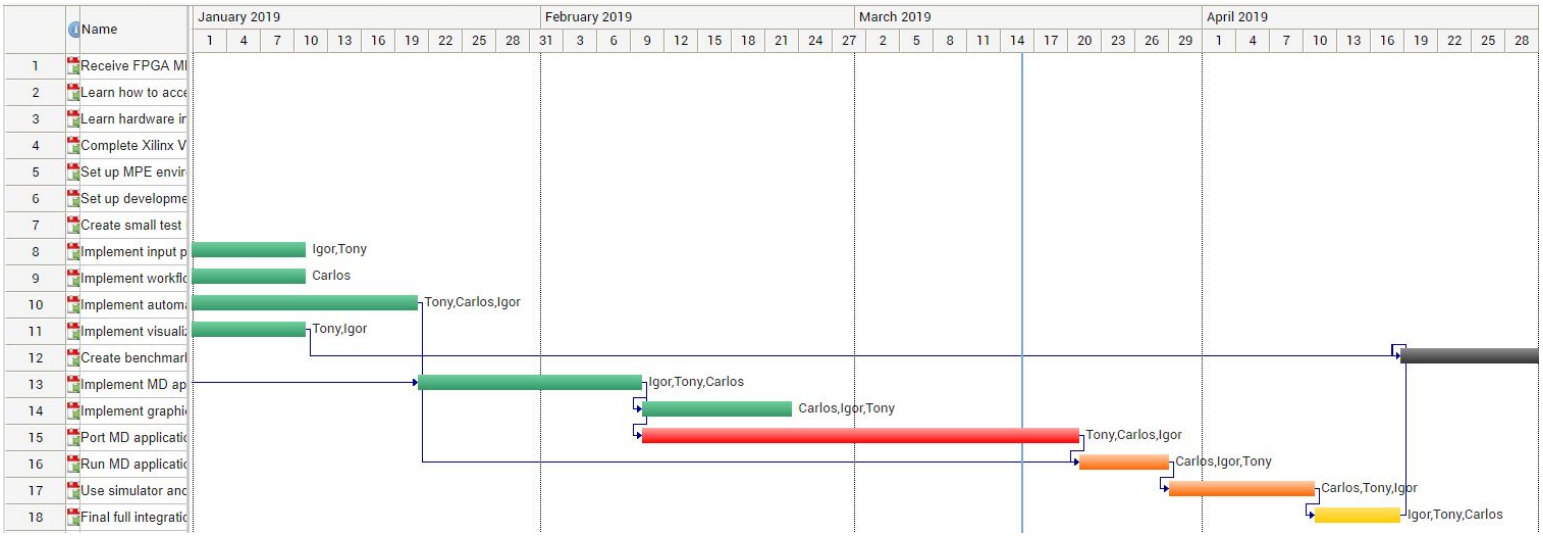


Fig. A.2 Final Report Gantt Chart - Winter Semester

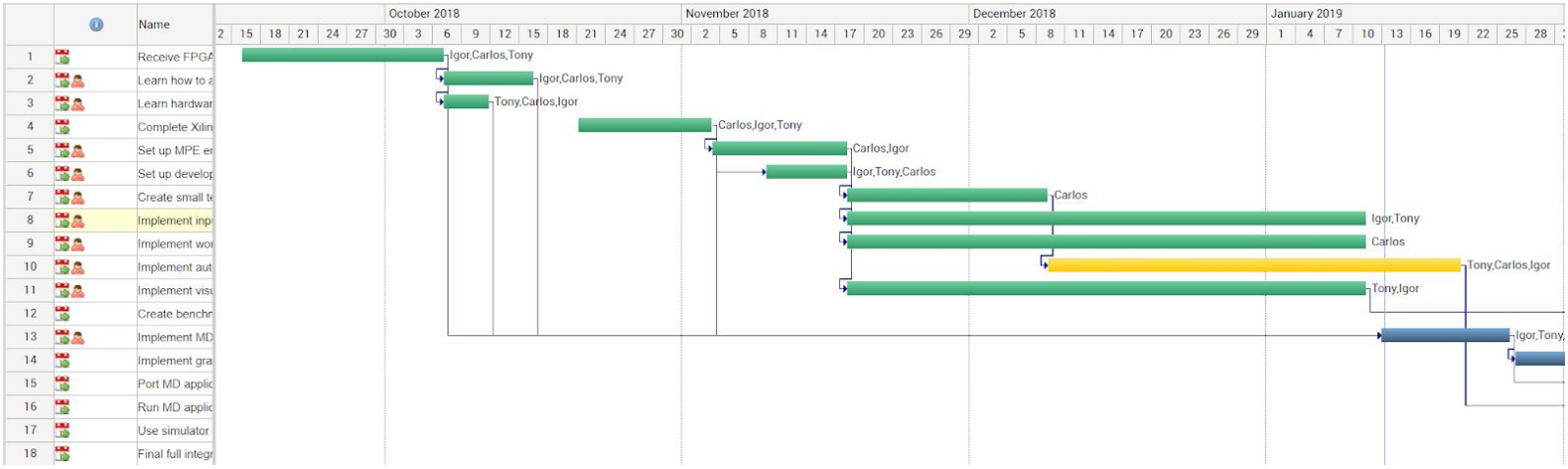


Fig. A.3 Progress Report Gantt Chart - Fall Semester

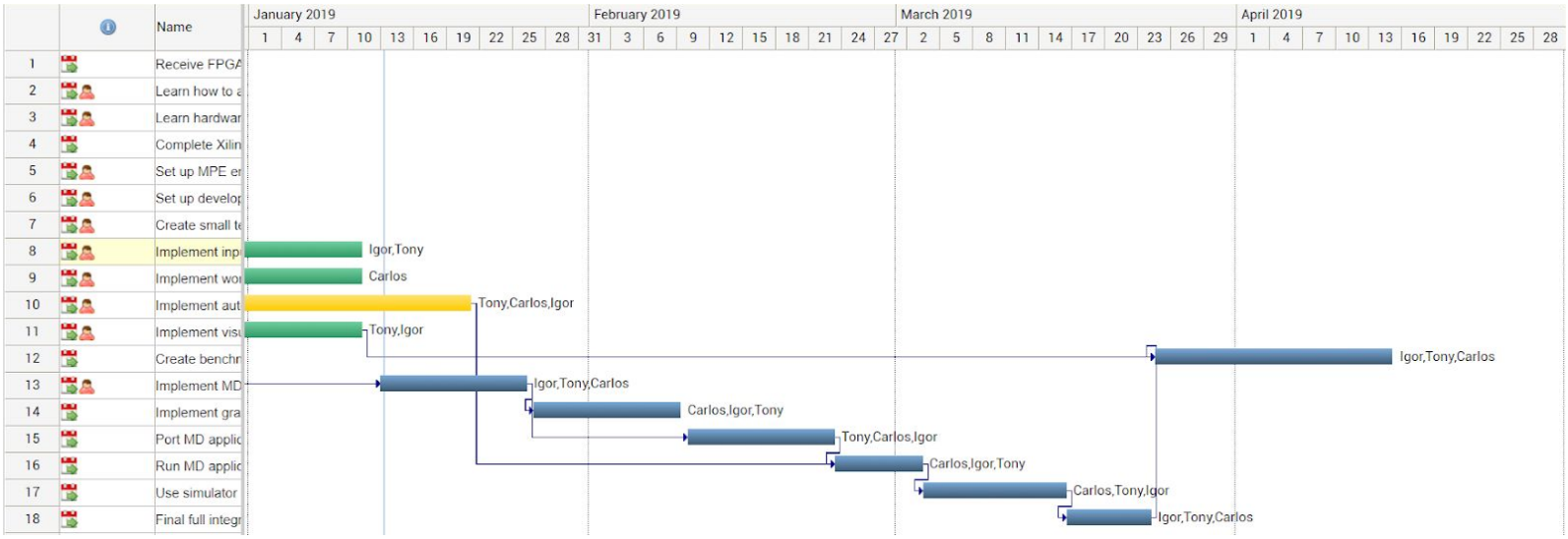


Fig. A.4 Progress Report Gantt Chart - Winter Semester

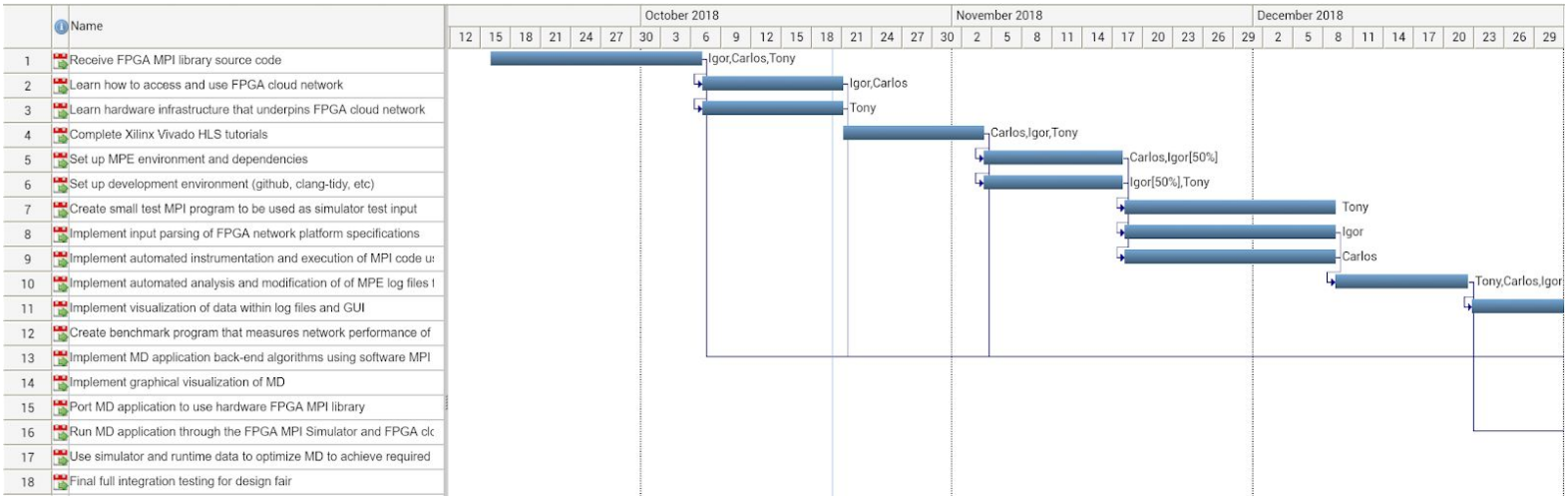


Fig. A.5 Project Proposal Gantt Chart - Fall Semester

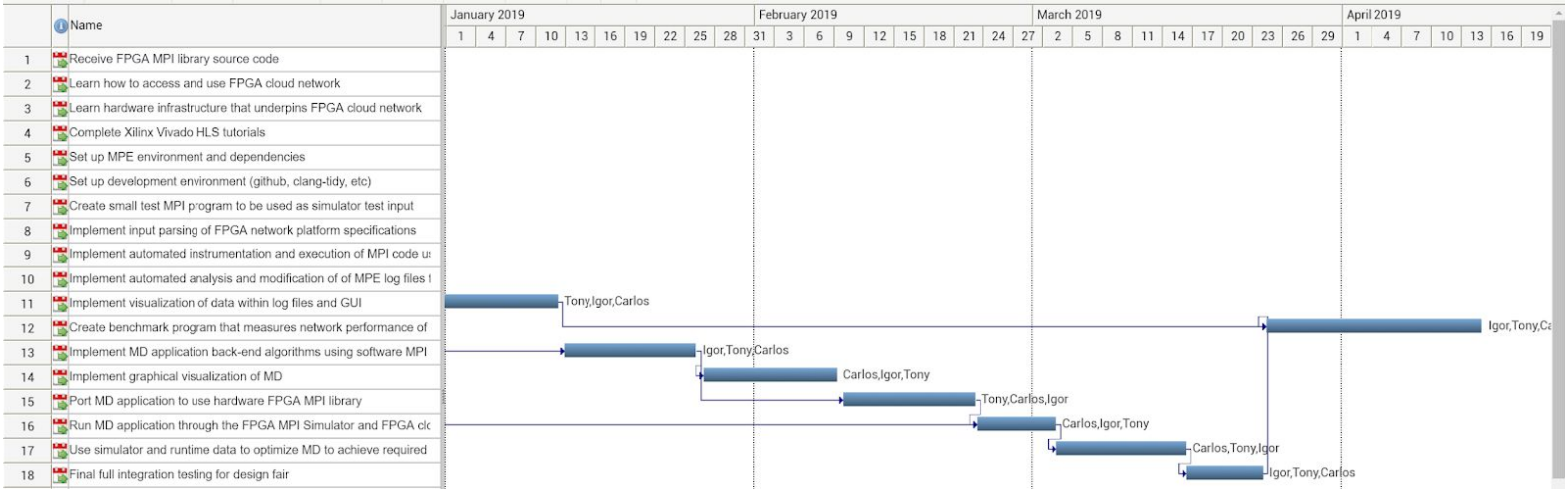


Fig. A.6 Project Proposal Gantt Chart - Winter Semester

Appendix B: Financial Plan

The financial plan estimates are detailed below. Nothing new was added since the project proposal. The full financial summary can be found in figure B1:

1. Source Control Software: Private Git repository price listed as 7$ per month.
2. Xilinx Virtex Ultra Scale Plus FPGAs: FPGA price listed as $7000].
3. Xilinx Vivado Design Suite: Software price listed as $4295.
4. Student hours: 8 month capstone; around 4 weeks per month; 5 working days in a week; 2.5 hours of work per day.
5. Student hourly salary: 20$ per hour is close to our salaries during PEY.

The items listed below are of great importance for the success of our project. Contingency plans for these items are summarized below:

1. *(Priority 1) Xilinx Virtex Ultra Scale Plus FPGAs: In case we do not get these particular FPGAs, we may be able to request less advanced and cheaper FPGAs from the university. In the worst-case scenario, we would be able to buy our own. This would greatly increase the amount of actual funding that we require. Theoretically, our project should work with any FPGAs, but our supervisor already has access to the FPGAs mentioned in the financial plan.*
2. *(Priority 1) Xilinx Vivado Design Suite: The university has a license to this software, and it is readily available on many of the Undergraduate Lab computers. There is no foreseeable way that we would not be able to gain access to it. However, in the non-existent chance that we cannot access this software, we would be able to buy it ourselves, greatly increasing our required funding.*

Consumables/Services											
Item	**Priority**	**Cost/Unit**	**Quantity**	**Total Cost**	**Requires Funding**						
Source Control sc	2	$7/month	8 months	$56	Y					**Funding**	
Total Consumables/Services				$56						Students (3)	$19
Total Requiring Funding				$56						Supervisor	$0
										Request From De	$0
										Total Funding	$56
Capital Equipment											
Item	**Priority**	**Cost/Unit**	**Quantity**	**Total Cost**	**Requires Fundin**	**Kept/Paid for by Students**					
Xilinx Virtex Ultra	1	$7,000	3	$21,000	N	N					
Xilinx Vivado Des	1	$4,295	1	$4,295	N	N					
Total Capital Equipment				$25,295							
Total Requiring Funding				$0							
Student Labour											
Item	**Cost/Unit**	**Quantity (# of hc**	**Total Cost**								
Student 1	$20	400	$8,000								
Student 2	$20	400	$8,000								
Student 3	$20	400	$8,000								
Total Student Labour			$24,000								
Total Cost of Project			$49,351								
Total Cost Requiring Funding			$56								

Figure B1. Financial Summary

Appendix C: Validation and Acceptance Tests

Change?	ID	Project Requirement	Verification Method
	1.0	Create MPI FPGA cloud simulator for performance measurements	**Test:** A simple test MPI software program can be written. We can run our simulator on this MPI program to see if a performance result is computed.
	2.0	Create user interface for simulator	**Review of Design:** True/false check for existence of user interface.
	3.0	Minimize user time spent using UI	**Test:** Ask one of our supervisor's graduate students to use our simulator. Measure their time interacting with the user interface (configuring inputs and interpreting outputs).
	4.0	Simulation needs to complete quickly	**Test:** Measure standalone runtime of the user's MPI software program and compare it to the runtime of the simulator. These times should be virtually identical.
MODIFIED	5.0	Create software-based MPI application	**Test:** ~~Assuming proposed solution from "Assessment of Proposed Solution" section. There are many existing implementations of MD that we can compare our implementation against to check for correctness. The most exhaustive approach would be to check every time step of the MD simulation and compare the position of all molecules.~~ Due to the issues faced by Tony towards the end of the year, we spoke to our supervisor Paul Chow about re-scoping our project. We agreed that it would be reasonable to implement a scaled-down version of MD with a limited number of forces. Thus, it is no longer possible to verify correctness against different implementations of MD. However, our supervisor Paul Chow is content with us simply showing a visualization of MD that has moving atoms. Therefore, the test is to show a visualization of MD with moving atoms driven by short-range electrostatic forces.
	6.0	Port software-based MPI application to use FPGA hardware acceleration (HLS) and the MPI FPGA cloud	**Test:** Can compare our MPI FPGA cloud implementation against our pure software implementation. Can verify the position of all molecules at every timestep.
	7.0	FPGA MPI cloud application should be	**Test:** We can measure and compare the latencies of the pure software implementation and the MPI FPGA cloud

		faster than equivalent software implementation	implementation.
	8.0	Simulator should be able to support MPI applications using the MPI_send and MPI_recv functions	**Test:** Can run our simulator on an MPI application with MPI_send and MPI_recv functions and check if the simulator completes and produces a result.
	9.0	Create benchmarking tool for determining network parameters of MPI FPGA platform	**Test:** The parameters of current version of MPI FPGA cloud platform are known by our supervisor. We can run our benchmark and confirm that we report the same parameters.
	10.0	Simulator tool must function in the MPI FPGA cloud server environment	**Review of Design:** Implementation will be done entirely in the server environment. Any verification of functional requirements will be done in the server environment.
	11.0	Simulator should be accurate to a certain range	**Test:** We can directly measure the runtime of our simple test MPI program from ID 1.0 and compare those results to our simulator results for the same program.
	12.0	GUI should be able to save its existing state to a config file	**Review of Design:** True/false check if functionality is working.
	13.0	GUI should be able to load a saved state from a config file	**Review of Design:** True/false check if functionality is working.
	14.0	Workflow Manager should save user input from GUI as .cfg files	**Test:** Configuration files are generated when the user clicks save.
	15.0	Workflow Manager should pass data from the .cfg files to the GUI in the same format for loading	**Test:** The GUI gets the values from the configuration files when load is clicked.
	16.0	.slog to text log module should work	**Test:** An .slog converted to text log then converted back to .slog is identical.

Appendix D

Figure D2 is a simplified example of the graph model that a textlog (such as the one in Figure D1) is converted to in the MPE backend.

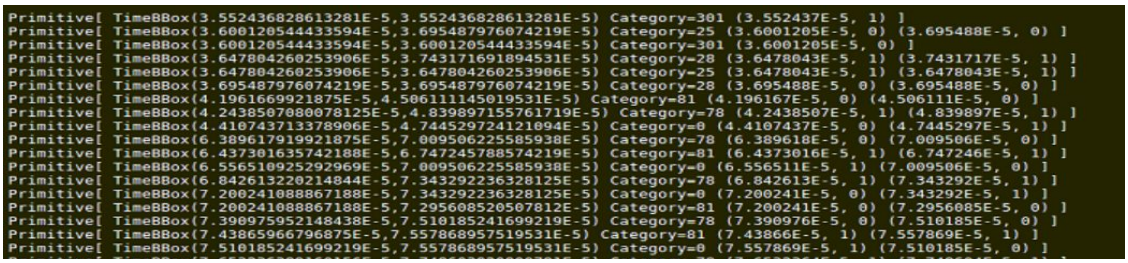


Figure D2. Example of a textlog

Each node in the graph represents an event that was logged by MPE, and the edges between the nodes represent causal dependencies between events. Each node has a start and end time. Each node's start time is always the maximum of the end times of all ancestors nodes (nodes pointing at that discussed node). For example, nodes B and E point at node C. Node C's start time is the maximum of 3s and 3.2s, the end times of nodes B and E. Generally all nodes within a rank (an MPI process) point exclusively to the the next chronological node in the same rank. The only exception is when an MPI message (green node below) is passed from one rank to another.

Assuming this graph has been constructed for an example MPI process, the rest of the MPE backend flow would work as follows: depending on the platform parameters that the user has selected in the frontend GUI, the MPE backend would invoke backpropagation on any nodes that are encompassed by the platform characteristics change. For example, if the user specifies that the network connection between rank 0 and rank 1 is twice as slow, then node E will now complete at 3.9s. This change has to be propagated to all descendent nodes of E. Node C will start at 3.9s and finish at 4.7s. Node D will start at 4.7s and finish at 6.7s. Although this example is contrived, and only contains one message node, in a real scenario, this process would need to be repeated multiple times.

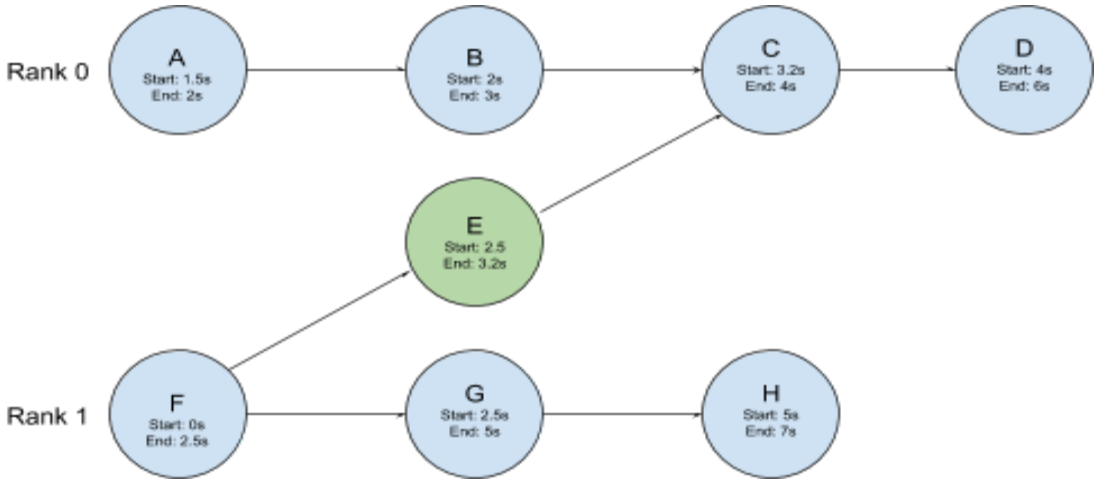


Figure D2. An example of a textlog converted to a graph by the Back Propagator

Referring back to figure D1, the textlog file only contains information information related to MPE events. These include MPI_Send calls, MPI_Recv calls, and MPI messages. An example sequence of log events could be as follows:

- Process #0 initiates an MPI_Send call. This call lasts in the timeframe (2.3s, 2,4s)
- The MPI messages between process #0 and process #1 travels in the timeframe (2.35s to 2.49s)
- Process #1 initiates an MPI_Recv call. This call lasts in the timeframe (2.1s to 2.51s). This process started the call before the message was sent, but ended it immediately after receiving the message.

Although the textlog provides very clear information regarding when MPI events occur, it is not obvious how we can determine how the details of the rest of the computational work of a process. We solve this simply by assuming that everything in between two MPI events for a process is a section of 'computational work'. In the graph in figure D2, these 'computational work' sections would correspond to their own work type, which are affected temporally by the hardware acceleration factors platform characteristics in figure D3 below. For example, if a user specifies that HW Rank (process) 0 is 2 times faster, then the backpropagation algorithm will target 'computational work' nodes with a speedup of 2:

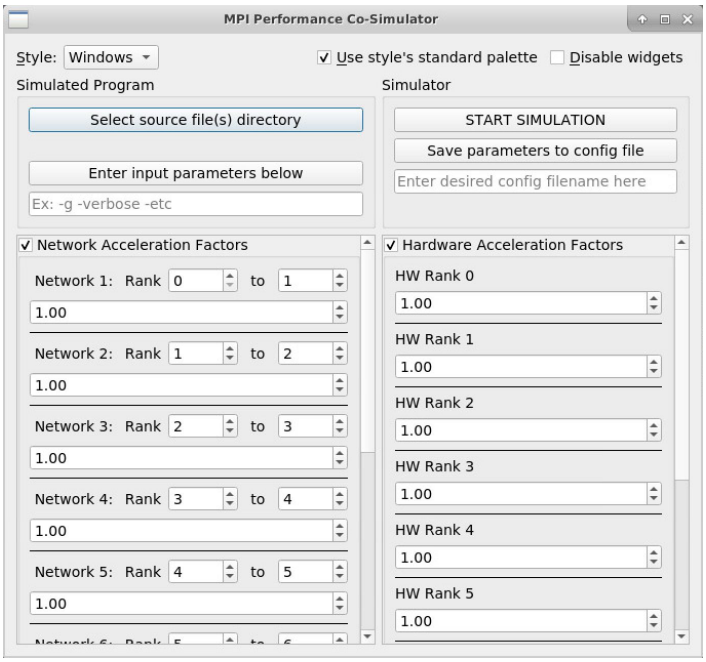


Figure D3, Example of Hardware Acceleration Factors which target the 'computational work' sections of the figure below.

Appendix E

```python
#propogate the startTime change to the successor
def propagate(self, node, newEndTime):
    for successor in self.graph.successors(node):
        index = 0
        if (node.getType() == "Arrow"):
            index = 1
            #self.propagate(node.getArrowEarliestArrivalRef(), node.getArrowEarliestArrivalRef().updateStartTime(newEndTime, index))
        #else:
        self.propagate(successor, successor.updateStartTime(newEndTime, index))

#if the node matches the target type, change its length, and propogate the change
def backPropagate(self, targetType, speedupFactor):
    for node in self.graph.nodes(data=False):
        if (node.isTargetType(targetType)):
            newEndTime = node.speedupReverse(speedupFactor)
            self.propagate(node, newEndTime)
            self.markAllNodesUnvisited()

#if the node matches the target type and rank, change its length, and propogate the change
#Used for doing backpropagation on HW rank speedup
def backPropagateHW(self, speedupFactor, rank):
    for node in self.graph.nodes(data=False):
        if (node.isTargetType("Worker")):
            if (node.getFromRank() == rank):
                newEndTime = node.speedup(speedupFactor)
                self.propagate(node, newEndTime)
                self.markAllNodesUnvisited()

#used for back propogation for network speedup
#need to do it for two types of nodes: messages, and the second half of MPI_Send
#for the latter, we need to know its associated rank (ie which rank does the MPI_send send to)
def backPropagateNetwork(self, speedupFactor, fromRank, toRank):
    for node in self.graph.nodes(data=False):
        if (node.isTargetType("Arrow")):
            if ((node.getFromRank() == fromRank) and (node.getToRank() == toRank)):
                newEndTime = node.speedup(speedupFactor)
                self.propagate(node, newEndTime)
                self.markAllNodesUnvisited()
        elif (node.isTargetType("MPI_Send_End")):
            if ((node.getFromRank() == fromRank) and (node.getAssociatedRank() == toRank)):
                newEndTime = node.speedup(speedupFactor)
                self.propagate(node, newEndTime)
                self.markAllNodesUnvisited()


def markAllNodesUnvisited(self):
    for node in self.graph:
        node.markNotVisited()
```

Figure E1. Example of Back Propagator Code in Python

```java
public static final void main( String[] args )
{

    InputLog         slog_ins;
    TreeDir          treedir;
    TimeBoundingBox  timeframe;
    String           err_msg;

    parseCmdLineArgs( args );


    slog_ins   = new InputLog( in_filename );
    if ( slog_ins == null ) {
        System.err.println( "Null input logfile!" );
        System.exit( 1 );
    }
    if ( ! slog_ins.isSLOG2() ) {
        System.err.println( in_filename + " is NOT SLOG-2 file!." );
        System.exit( 1 );
    }
    if ( (err_msg = slog_ins.getCompatibleHeader()) != null ) {
        System.err.print( err_msg );
        InputLog.stdoutConfirmation();
    }

    slog_ins.initialize();
    System.out.println( slog_ins.printCategoryMap() );
    if ( !printDrawables ) {
        slog_ins.close();
        System.exit( 0 );
    }

    treedir = slog_ins.getTreeDir();
    // System.out.println( treedir );

    TreeDirValue  root_dir;
    root_dir  = (TreeDirValue) treedir.get( treedir.firstKey() );
    timeframe = new TimeBoundingBox( root_dir.getTimeBoundingBox() );
    scaleTimeBounds( timeframe );

    boolean   isStartTimeOrdered;
    boolean   isIncreTimeOrdered;
    double    prev_bordertime;

    Iterator  dobj_itr;
    Drawable  dobj;
    double    dobj_bordertime;
    int       dobj_count;
```

Figure E2. Example of extended MPE code in Java to support Slog to Textlog conversion and back

Appendix F

The following is a justification on the accuracy of our simulator. While originally we were supposed to verify the accuracy of our simulator using our Molecular Dynamics Hardware benchmark, this could not come to fruition as discussed in the *FInal Design* section of our document. The following is proof that our simulator provides results that are not unrealistic (ie. if a component of the platform is sped up, the overall runtime of the program should naturally decrease). Additionally, the example below demonstrates that the simulator accurately responds to bottlenecks in the program. The first two images show the simulator running with default GUI acceleration values. *NOTE: default GUI acceleration values are not default platform characteristics. The default GUI values are simply values of 1 for all acceleration factors. The default platform characteristics are the actual acceleration factors of our supervisor's platform, which are almost certainly different than the default GUI values.*
Note: the source code of the program being simulated in shown in Appendix I.

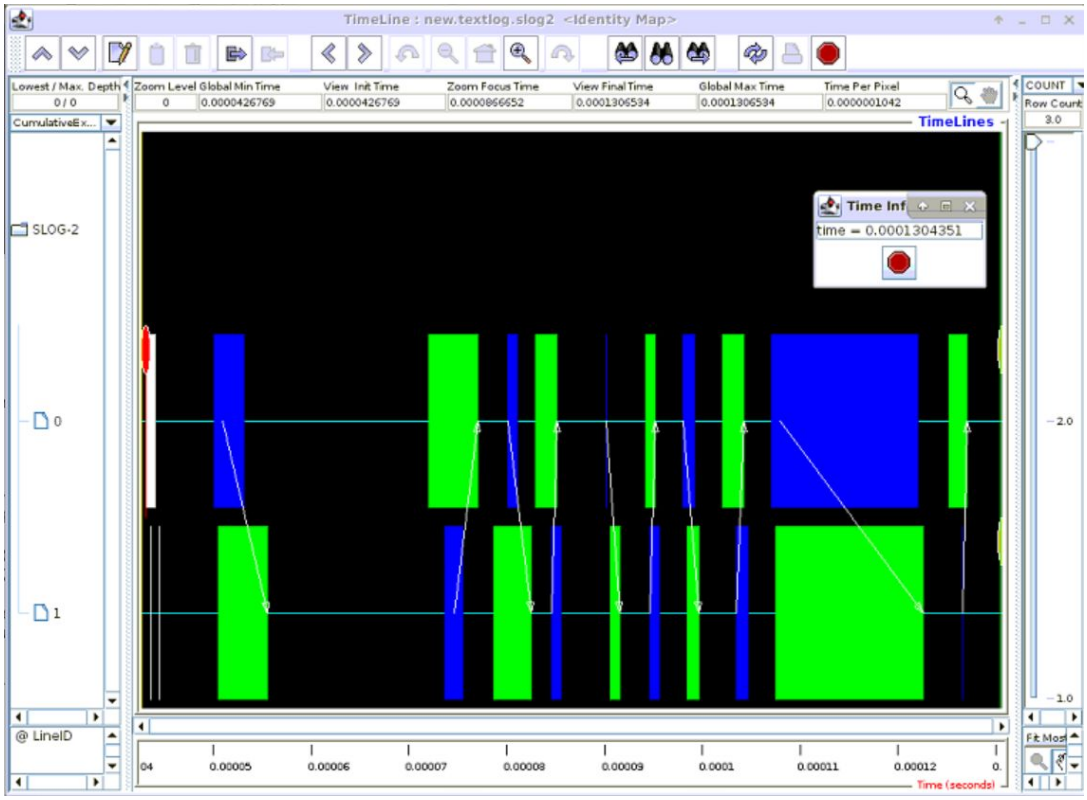


Figure F1. Simulation Visualization with default GUI values; runtime of 0.13 ms

Figure F2. Default GUI acceleration values corresponding to figure F1

Now, we accelerate process (rank) 0 by a factor of 3. The runtime drops from 0.13ms to 0.084 ms, which is a realistic value. Any slowdown that could be attributed to process 0 has likely been removed. Visually, this is seen by row 0 in Figure 3 having very little black regions. Black regions in the visualization tool represent any work that is not an MPI event. These regions are targeted when we set the Hardware Acceleration factors in the GUI. This is described in more detail in Appendix D.

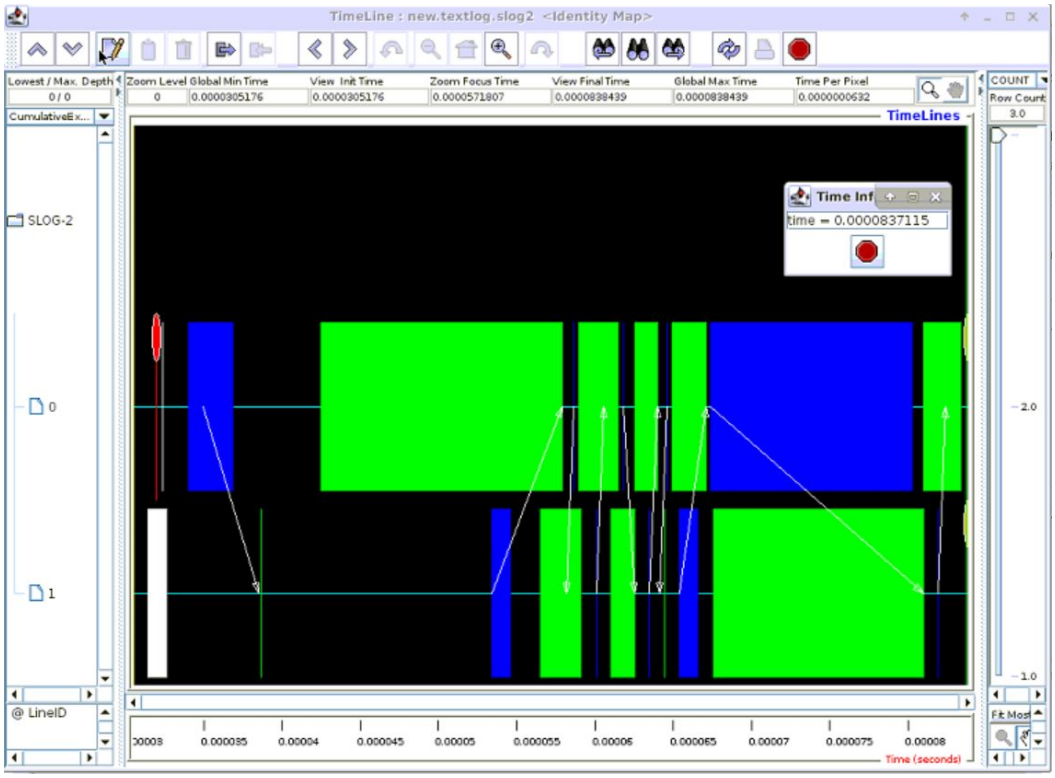


Figure F3. Simulation Visualization with process 0 accelerated by factor of 3; runtime of 0.084 ms

Figure F4. GUI acceleration values corresponding to figure F3

Now we accelerate process 0 by a factor of 5 instead of 3. The runtime only decreases to 0.083 ms down from 0.084 ms. This shows that our simulator adequately adheres to the fact that the program cannot be sped up any more. No matter how much faster we make process 0, process 0 still needs to wait on process 1 to finish its work. This 'dependency rule' is most fundamental factor that can influence accuracy. Anything else can be fine tuned and tweaked to accurately match our supervisor's platform, and provide accuracy.

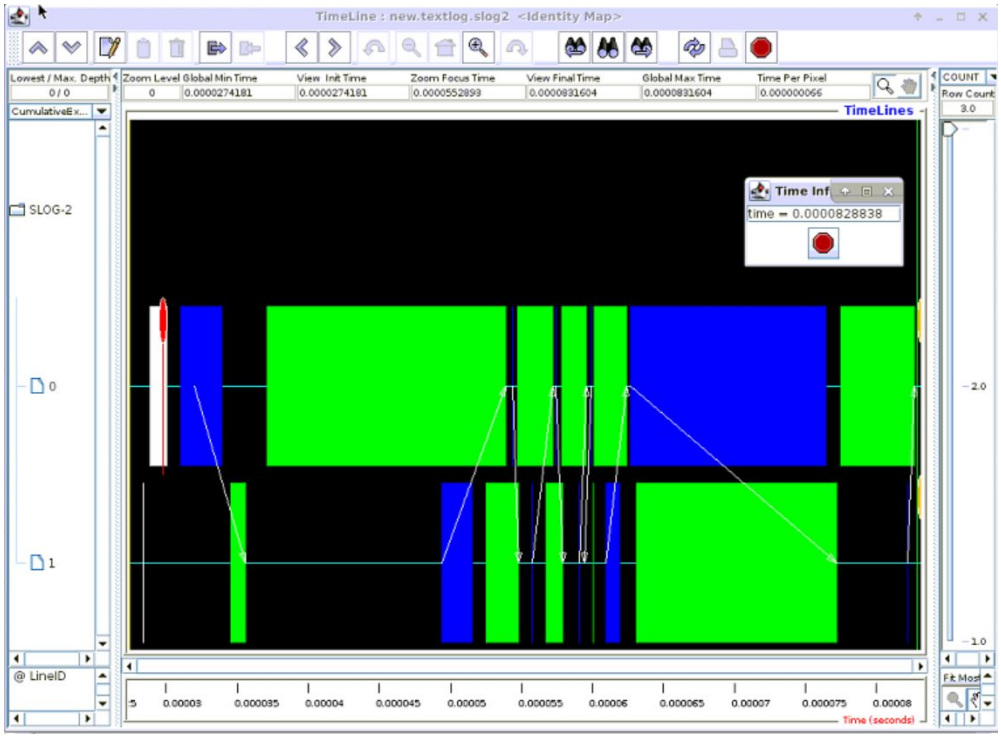


Figure F5. Simulation Visualization with process 0 accelerated by factor of 5; runtime of 0.083 ms

Figure F6. GUI acceleration values corresponding to figure F5

If we now also accelerate process 1 by a factor of 5, we can see that the runtime drops significantly down to 0.061 ms, as expected.

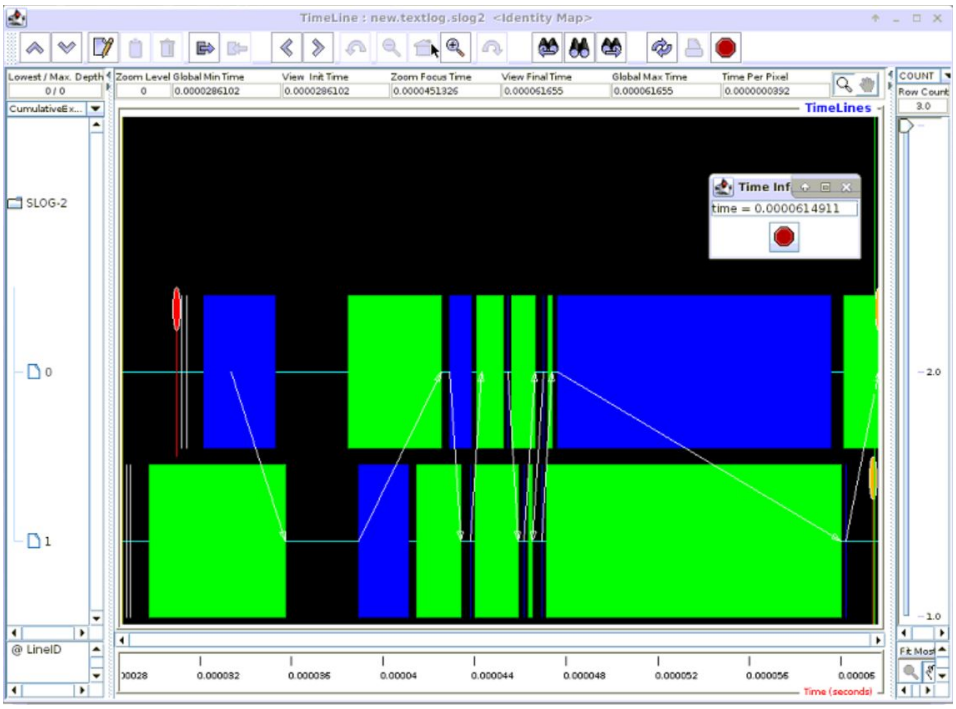


Figure F7. Simulation Visualization with processes 0 and 1 accelerated by factor of 5; runtime of 0.061 ms

Figure F8. GUI acceleration values corresponding to figure F7

NOTE: A very important issue that can affect the accuracy of the simulator is the fact that for each session of the simulation, the user's program is compiled and re-executed. Since the execution of a program is not guaranteed to be the same for every execution, the MPI logs are also not guaranteed to be the same. Thus, for programs with a short execution time, such as the example program above, the results will not be consistent between runs. This is not a big problem for longer running programs, since the small fluctuations in runtime will be negligible compared to the length of the problem. For shorter running programs, this problem can be mitigated by isolating our container to ensure that the host machine for the virtual environment is unloaded (no computational work being done other than our simulator).

Appendix G

The following appendix shows the final output of our Software Molecular Dynamics MPI application. The output is visualized using an industry standard molecular dynamics tool called VMD. Figures G1, G2, and G3 show the MD visualization 150 time frames (10^{-15} seconds) apart.

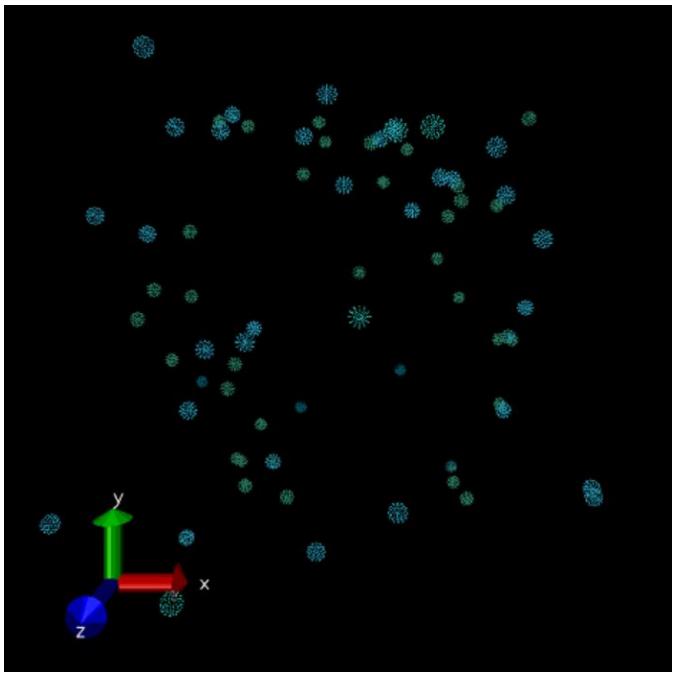


Figure G1. MD simulation at time step 0

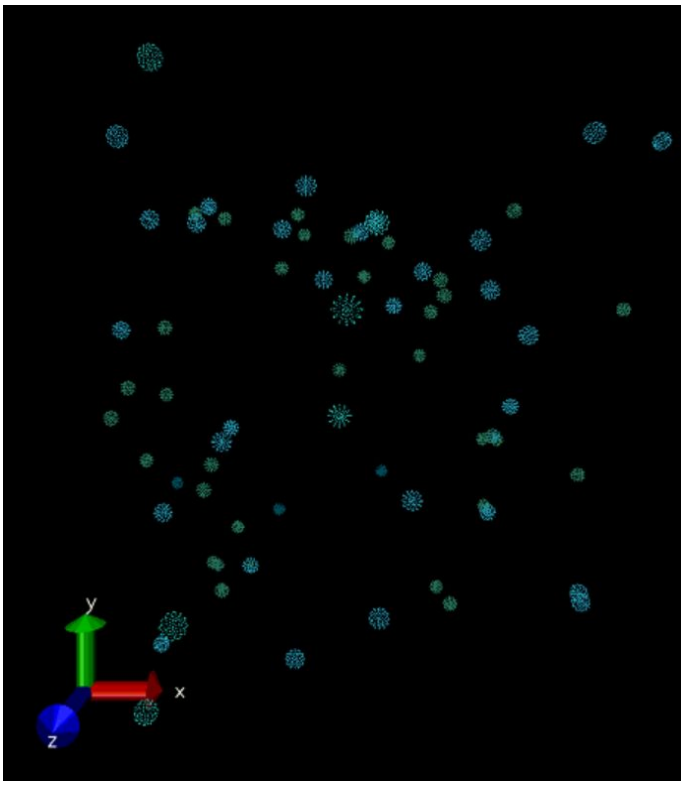


Figure G2. MD simulation at time step 150

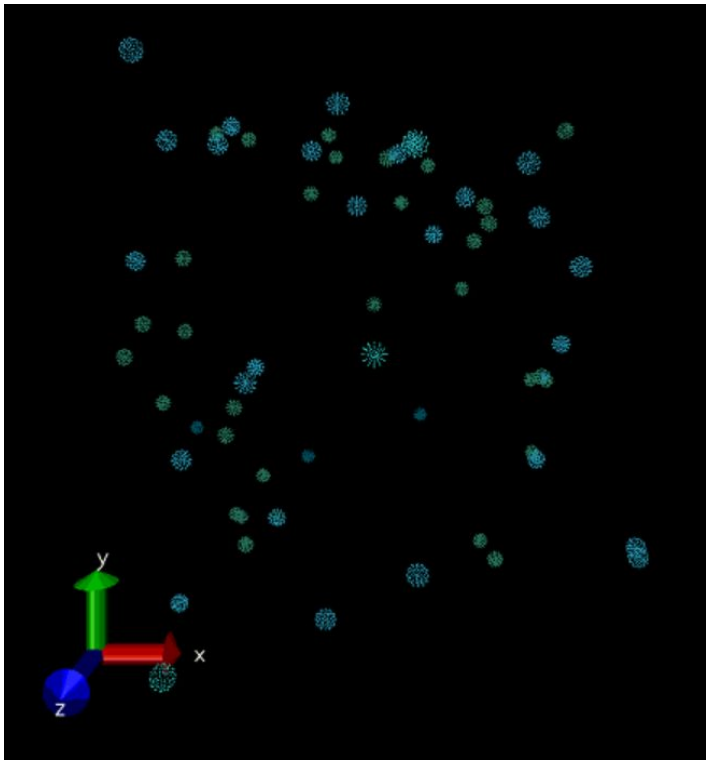


Figure G3. MD simulation at time step 300

Appendix H

The following is a code snippet of software MD.

```
void
doMD(int atomCount, int stepCount)
{
    int i, j, r, q;
    int stepIndex = 0;
    //float *forceSum = NULL;
    //float *vel = NULL;

    //float *pos = (float*) malloc(atomCount * 3 * sizeof(float));
    float pos[atomCount * 3 ];

    //float *force = (float*) malloc(atomCount * 3 * sizeof(float));
    float force[atomCount * 3 ];

    //forceSum = (float*) malloc(atomCount * 3 * sizeof(float));
    float forceSum[atomCount * 3 ];

    //vel = (float*) malloc(atomCount * 3 * sizeof(float));
    float vel[atomCount * 3 ];


    if (!world_rank)
    {

        float boltzmanTemp = 298. * BOLTZMAN;
        float tempOverMass = sqrt(boltzmanTemp / 40.0);


        for (i=0; i != atomCount; ++i)
        {
            pos[i * 3] = (float)rand() / (float)RAND_MAX * CUBELENGTH;
            pos[i * 3 + 1] = (float)rand() / (float)RAND_MAX * CUBELENGTH;
            pos[i * 3 + 2] = (float)rand() / (float)RAND_MAX * CUBELENGTH;

            double rnd = (double)rand() / (double)RAND_MAX;

            rnd *= 12.;
            rnd -= 6.;
            vel[i * 3] = (rnd * tempOverMass); // 1000;

            rnd = (double)rand() / (double)RAND_MAX;
            rnd *= 12.;
            rnd -= 6.;
            vel[i * 3 + 1] = (rnd * tempOverMass); // 1000;

            rnd = (double)rand() / (double)RAND_MAX;
            rnd *= 12.;
            rnd -= 6.;
            vel[i * 3 + 2] = (rnd * tempOverMass); // 1000;

            #ifdef VERBOSE
            PRINT("%3d) V: %8.3f %8.3f %8.3f\n", i, vel[i * 3], vel[i * 3 + 1], vel[i * 3 + 2]);
            #endif
        }
```

Figure H1. Code snippet of software MD benchmark application

Appendix I

The following is a code snippet of the simple MPI program shown in appendix F. Both the MPI_Send and MPI_Recv functions are used.

```
int main(int argc, char** argv) {
  const int PING_PONG_LIMIT = 10;

  // Initialize the MPI environment
  MPI_Init(NULL, NULL);
  // Find out rank, size
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // We are assuming at least 2 processes for this task
  if (world_size != 2) {
    fprintf(stderr, "World size must be two for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }

  int ping_pong_count = 0;
  int partner_rank = (world_rank + 1) % 2;
  while (ping_pong_count < PING_PONG_LIMIT) {
    if (world_rank == ping_pong_count % 2) {
      // Increment the ping pong count before you send it
      ping_pong_count++;
      MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
      printf("%d sent and incremented ping_pong_count %d to %d\n",
             world_rank, ping_pong_count, partner_rank);
    } else {
      MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,
               MPI_STATUS_IGNORE);
      printf("%d received ping_pong_count %d from %d\n",
             world_rank, ping_pong_count, partner_rank);
    }
  }
  MPI_Finalize();
}
```

Figure I1. Code of the ping pong MPI application.

Appendix J

This is an example of a .cfg file that is created by the workflow manager in order to save the user's parameters from the Frontend GUI. This .cfg file is created with the given name and has descriptive names for each setting. This allows a user to modify them by hand.

Figure J1: Image of example configuration file and settings

Appendix K

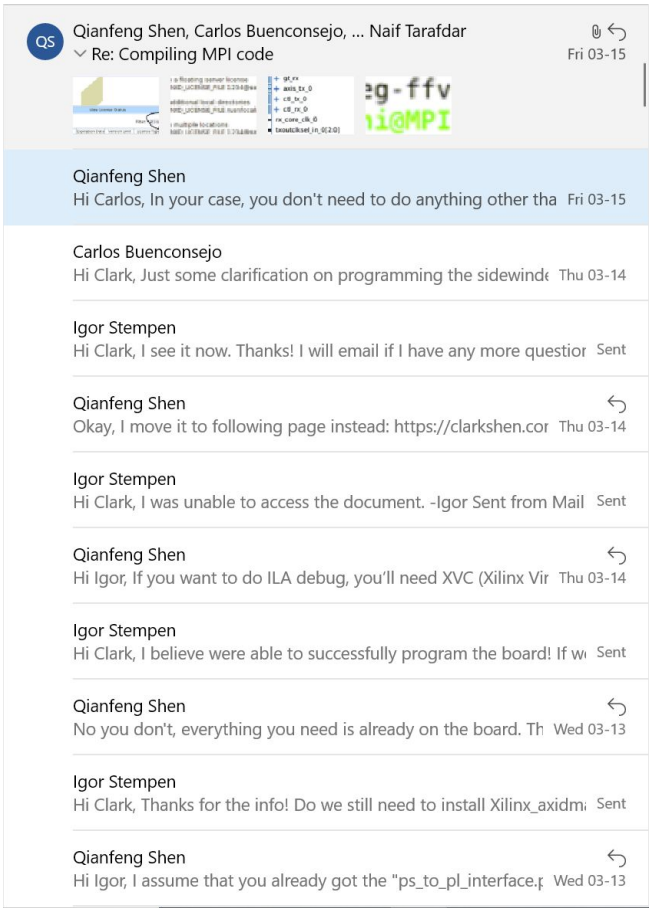
This is an image of the software interface used to pass data to the Frontend GUI when loading a .cfg file. The .cfg file is read and parsed by the Workflow Manager in order to present the data the same way it was given when a save was initiated. This makes it easy for the Frontend GUI to reload the settings.

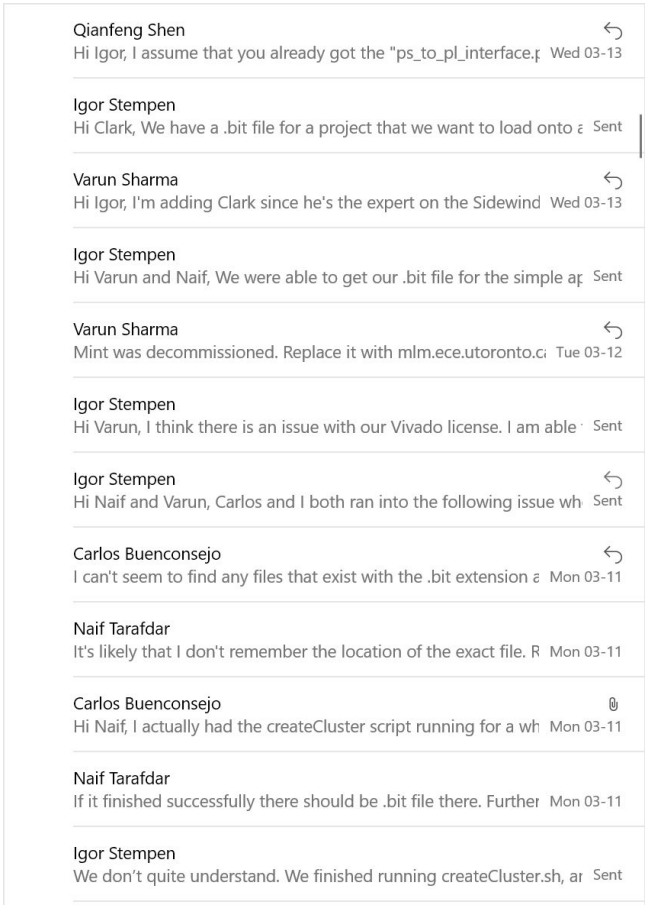
```
network_acceleration_factors = [[source_1, dest_1, acceleration_factor_1, name_1],
                                [source_2, dest_2, acceleration_factor_2, name_2],
                                ...,
                                [source_n, dest_n, acceleration_factor_n, name_n]]



hardware_acceleration_factors = [acceleration_factor_1,
                                 acceleration_factor_2,
                                 ...,
                                 acceleration_factor_n]
```

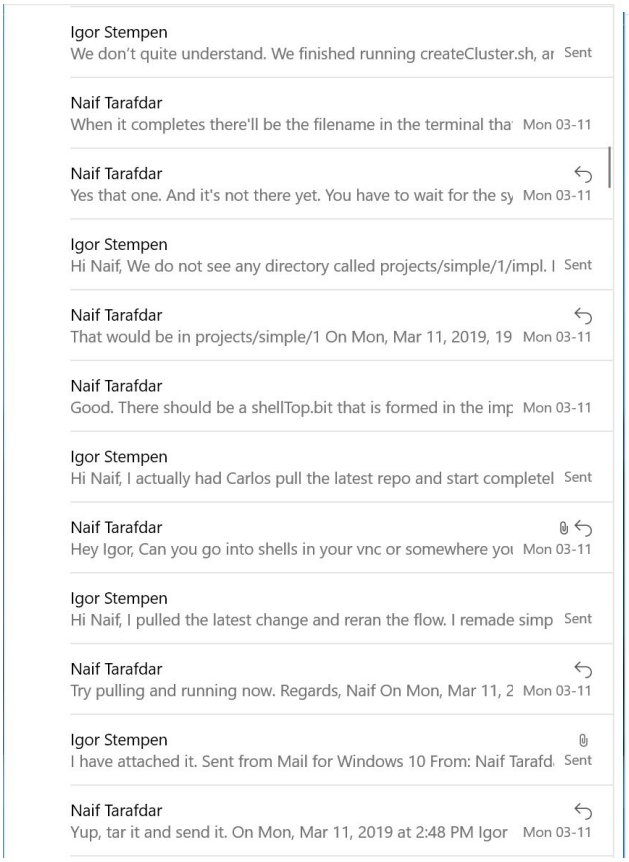
Figure K1: Image of interface between workflow manager and frontend GUI

Appendix L

The images below show three email threads with our supervisor's graduate students pertaining to the issues we faced with the FPGA MPI Platform (Galapagos). This accounts for several weeks worth of debugging and troubleshooting for us and for the graduate students. In total, there are 106 emails in the threads below.







Igor Stempen
We don't quite understand. We finished running createCluster.sh, ar Sent

Naif Tarafdar
When it completes there'll be the filename in the terminal tha Mon 03-11

Naif Tarafdar
Yes that one. And it's not there yet. You have to wait for the sy Mon 03-11

Igor Stempen
Hi Naif, We do not see any directory called projects/simple/1/impl. I Sent

Naif Tarafdar
That would be in projects/simple/1 On Mon, Mar 11, 2019, 19 Mon 03-11

Naif Tarafdar
Good. There should be a shellTop.bit that is formed in the imp Mon 03-11

Igor Stempen
Hi Naif, I actually had Carlos pull the latest repo and start completel Sent

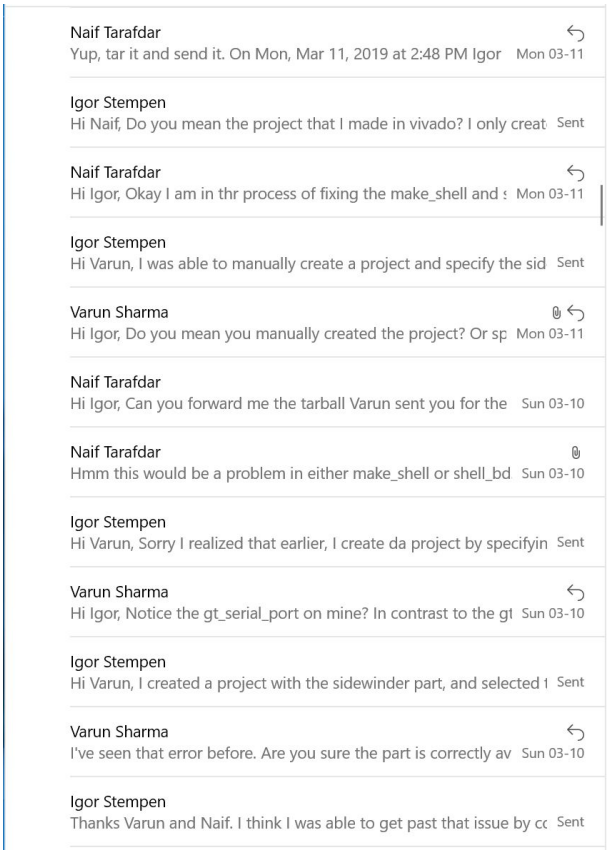
Naif Tarafdar
Hey Igor, Can you go into shells in your vnc or somewhere you Mon 03-11

Igor Stempen
Hi Naif, I pulled the latest change and reran the flow. I remade simp Sent

Naif Tarafdar
Try pulling and running now. Regards, Naif On Mon, Mar 11, 2 Mon 03-11

Igor Stempen
I have attached it. Sent from Mail for Windows 10 From: Naif Tarafd Sent

Naif Tarafdar
Yup, tar it and send it. On Mon, Mar 11, 2019 at 2:48 PM Igor Mon 03-11



Naif Tarafdar
Yup, tar it and send it. On Mon, Mar 11, 2019 at 2:48 PM Igor Mon 03-11

Igor Stempen
Hi Naif, Do you mean the project that I made in vivado? I only creat Sent

Naif Tarafdar
Hi Igor, Okay I am in thr process of fixing the make_shell and s Mon 03-11

Igor Stempen
Hi Varun, I was able to manually create a project and specify the sid Sent

Varun Sharma
Hi Igor, Do you mean you manually created the project? Or sp Mon 03-11

Naif Tarafdar
Hi Igor, Can you forward me the tarball Varun sent you for the Sun 03-10

Naif Tarafdar
Hmm this would be a problem in either make_shell or shell_bd Sun 03-10

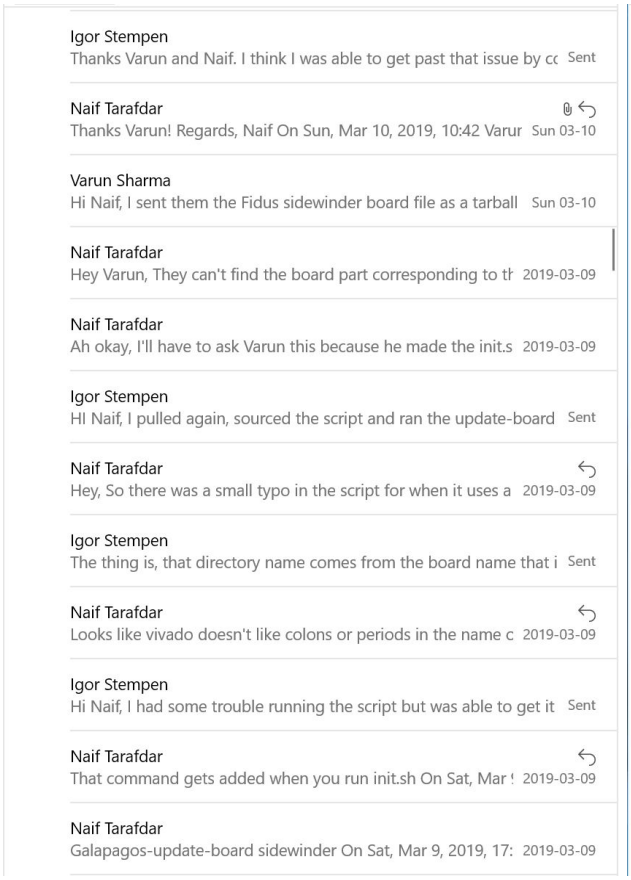
Igor Stempen
Hi Varun, Sorry I realized that earlier, I create da project by specifyin Sent

Varun Sharma
Hi Igor, Notice the gt_serial_port on mine? In contrast to the gt Sun 03-10

Igor Stempen
Hi Varun, I created a project with the sidewinder part, and selected t Sent

Varun Sharma
I've seen that error before. Are you sure the part is correctly av Sun 03-10

Igor Stempen
Thanks Varun and Naif. I think I was able to get past that issue by cc Sent



Igor Stempen
Thanks Varun and Naif. I think I was able to get past that issue by cc Sent

Naif Tarafdar
Thanks Varun! Regards, Naif On Sun, Mar 10, 2019, 10:42 Varur Sun 03-10

Varun Sharma
Hi Naif, I sent them the Fidus sidewinder board file as a tarball Sun 03-10

Naif Tarafdar
Hey Varun, They can't find the board part corresponding to th 2019-03-09

Naif Tarafdar
Ah okay, I'll have to ask Varun this because he made the init.s 2019-03-09

Igor Stempen
HI Naif, I pulled again, sourced the script and ran the update-board Sent

Naif Tarafdar
Hey, So there was a small typo in the script for when it uses a 2019-03-09

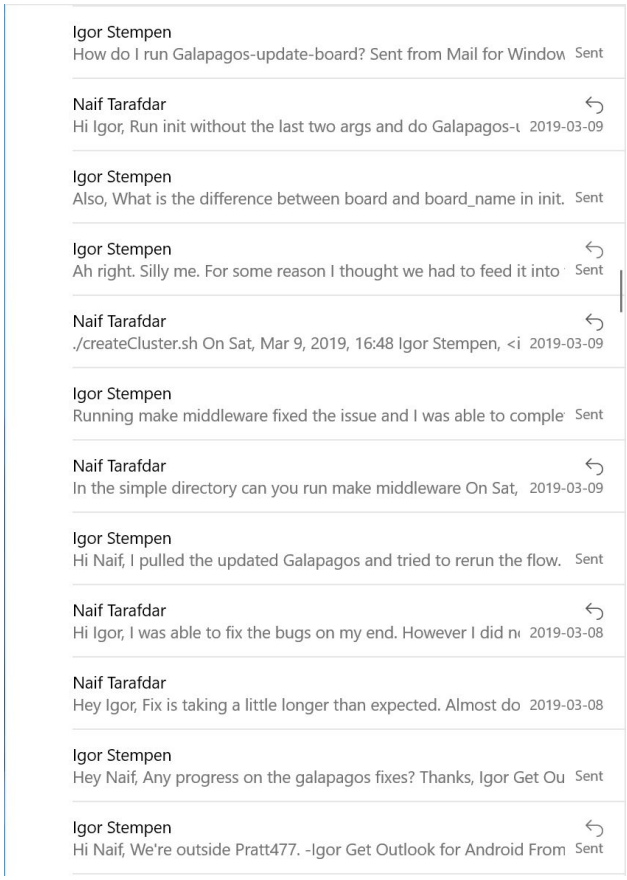
Igor Stempen
The thing is, that directory name comes from the board name that i Sent

Naif Tarafdar
Looks like vivado doesn't like colons or periods in the name c 2019-03-09

Igor Stempen
Hi Naif, I had some trouble running the script but was able to get it Sent

Naif Tarafdar
That command gets added when you run init.sh On Sat, Mar 2019-03-09

Naif Tarafdar
Galapagos-update-board sidewinder On Sat, Mar 9, 2019, 17: 2019-03-09



Igor Stempen
How do I run Galapagos-update-board? Sent from Mail for Windov Sent

Naif Tarafdar
Hi Igor, Run init without the last two args and do Galapagos-u 2019-03-09

Igor Stempen
Also, What is the difference between board and board_name in init. Sent

Igor Stempen
Ah right. Silly me. For some reason I thought we had to feed it into Sent

Naif Tarafdar
./createCluster.sh On Sat, Mar 9, 2019, 16:48 Igor Stempen, <i 2019-03-09

Igor Stempen
Running make middleware fixed the issue and I was able to comple Sent

Naif Tarafdar
In the simple directory can you run make middleware On Sat, 2019-03-09

Igor Stempen
Hi Naif, I pulled the updated Galapagos and tried to rerun the flow. Sent

Naif Tarafdar
Hi Igor, I was able to fix the bugs on my end. However I did n 2019-03-08

Naif Tarafdar
Hey Igor, Fix is taking a little longer than expected. Almost do 2019-03-08

Igor Stempen
Hey Naif, Any progress on the galapagos fixes? Thanks, Igor Get Ou Sent

Igor Stempen
Hi Naif, We're outside Pratt477. -Igor Get Outlook for Android From Sent

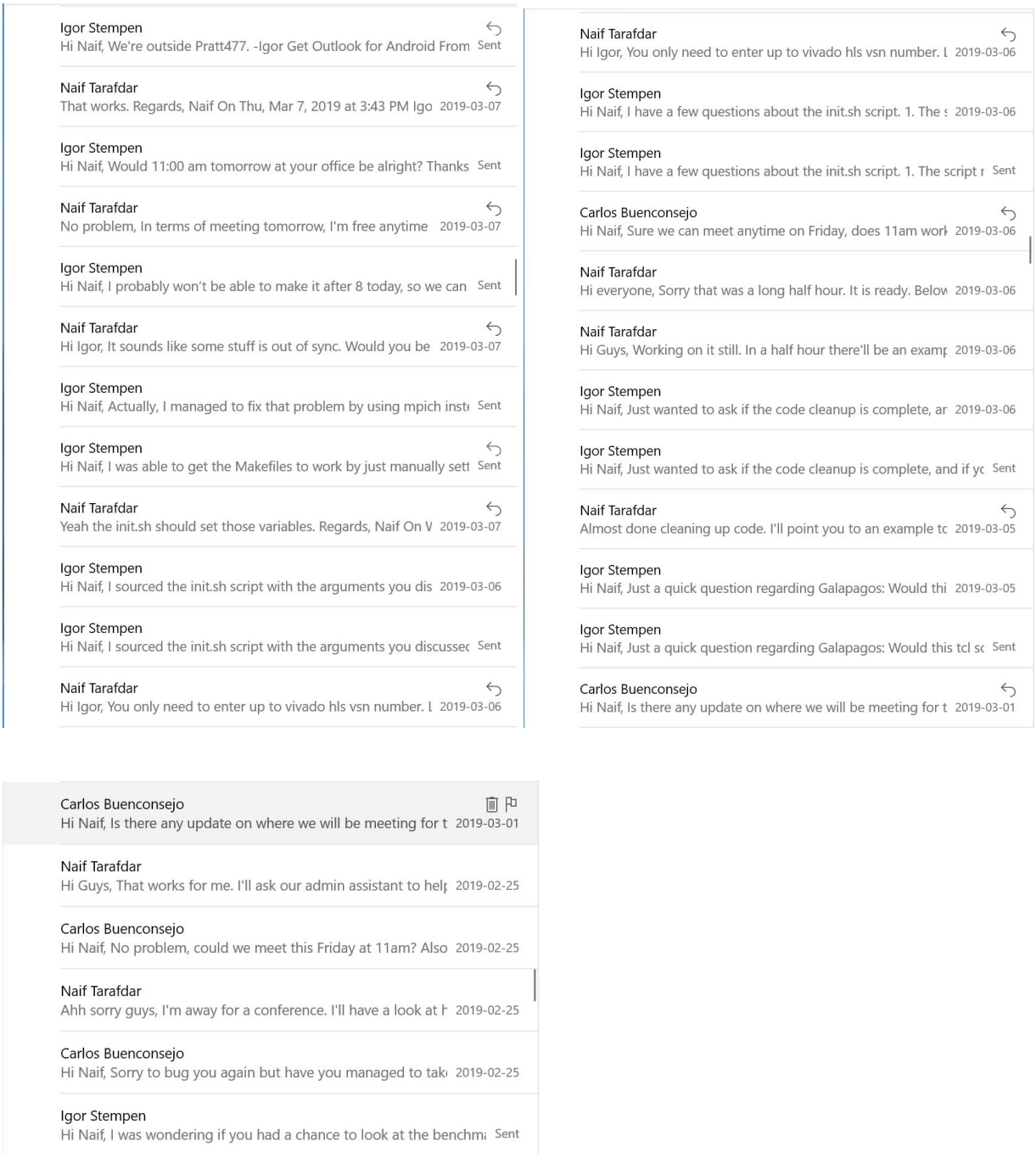


Figure set L1. Email thread #1 regarding issues faced with development on our supervisor's platform

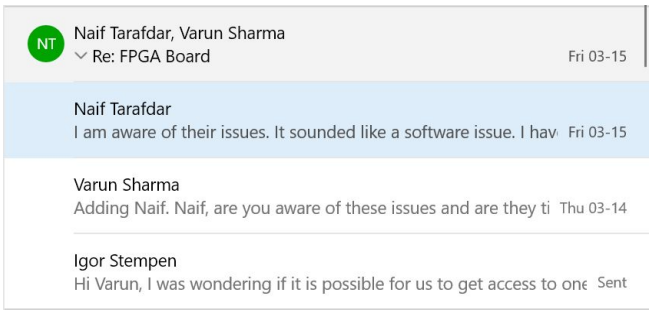


Figure L2. Email thread #2 regarding issues faced with FPGA development

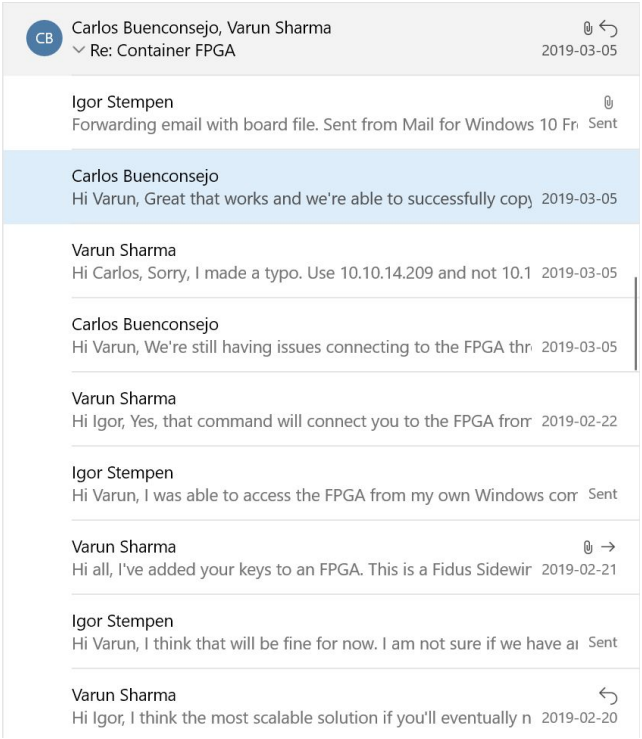


Figure L3. Email thread #3 regarding issues faced with FPGA development

Appendix M

This appendix documents the issues we encountered when trying to use our supervisor's platform.

```
int xml_parser(){

    ifstream myfile;
    vector<string> mac_addresses;
    vector<string> ip_addresses;
    string line;
    // myfile.open ("mpiMacAddresses");
    // while ( getline (myfile,line) )
    // {
    //   //cout << line << '\n';
    //   mac_addresses.push_back(line);
    // }
    // myfile.close();

    TiXmlDocument doc( "../configuration_files/map.xml" );
    doc.LoadFile();
    TiXmlHandle docHandle( &doc );
    TiXmlElement * fpga = docHandle.FirstChild( "cluster" ).FirstChild("node").ToElement();
    TiXmlElement * fpga2 = fpga;
    TiXmlElement * fpga3 = fpga;
    vector<string> kernel_to_mac_ptrs(256);

    for(fpga;fpga;fpga = fpga->NextSiblingElement()){
        TiXmlElement * mac_addr = fpga->FirstChild("mac")->ToElement();
        string temp_str =  mac_addr->GetText();
        //cout << temp_str <<endl;
        mac_addresses.push_back(temp_str);
    }
    for(fpga3;fpga3;fpga3 = fpga3->NextSiblingElement()){
        TiXmlElement * ip_addr = fpga3->FirstChild("ip")->ToElement();
        string temp_str =  ip_addr->GetText();
        //cout << temp_str <<endl;
        ip_addresses.push_back(temp_str);
    }
```

Figure M1: This is a small code snippet of a part of the software infrastructure of our supervisor's platform. The highlighted text shows the code we had to edit in order to progress in the workflow.

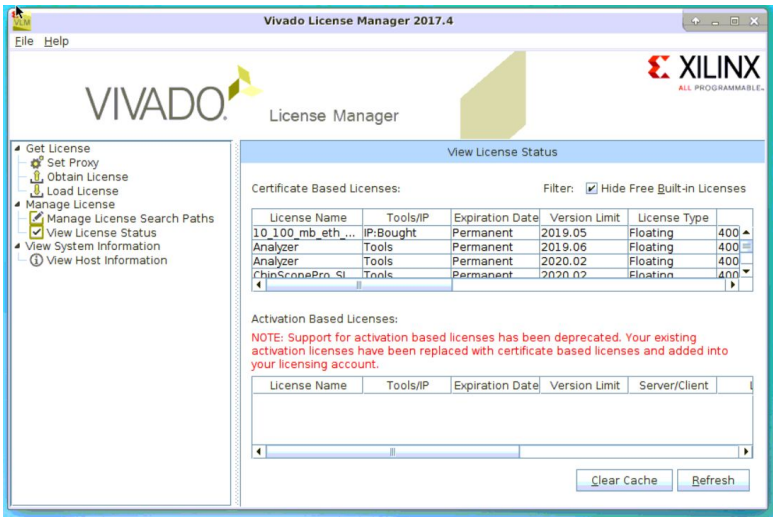


Figure M2: Image of the Vivado License Manager. We had issues with not having the required license for synthesizing the software into hardware kernels for the FPGA.

```
#define C_SYNC_ENV_PACKET 0
#define C_CLR2SND_PACKET 1
#define C_DATA_PACKET 2
#define C_ASYNC_ENV_PACKET 3
#define C_RECV_ERROR 4
#define C_DATA_TRANSMISSION_DONE 5
#define ETHERNET_INTERFACE_NAME "eth0"
#define ETH_PROTO 0x7400
#define BUFFER_SIZE 1024
#define HEADER_OFFSET 14
```

Figure M3: Bug in HUMboldt.h define. Originally the highlighted define was set to “eth3” but had to be “eth0” to work on our system.

```
buencons@MPI-FPGA-simulator:~/Documents/new_test_galapagos/galapagos-master/HUMb
oldt/apps/simple/sw$ ./example
rank: 0
sw
hw
Error: could not open socket
bind failed: Permission denied
buencons@MPI-FPGA-simulator:/bin$ ./example
rank: 0
Error: could not get interface index
start listening to port 7
Init is done!
Fatal error in MPI_Send: Invalid datatype, error stack:
MPI_Send(174): MPI_Send(buf=0x7ffe4627be90, count=1, INVALID DATATYPE, dest=0, t
ag=0, MPI_COMM_WORLD) failed
MPI_Send(104): Invalid datatype
[unset]: write_line error; fd=-1 buf=:cmd=abort exitcode=403302915
:
system msg for write_line failure : Bad file descriptor
terminate called without an active exception
Aborted (core dumped)
```

Figure M4: Permission issues in our Linux Container did not permit us from running the Atom Manager. This was solved by moving it to a folder with root permissions, but we still encountered bugs.

Appendix N

This appendix provides more detailed implementation details on the simulator GUI. And a sample section of the GUI code.

The GUI is implemented using an API called PyQT, which is a Python3 wrapper over the cross-platform C++ Qt API. We primarily made use of the QApplication and QWidget classes extended from the PyQt Core library to generate GUI elements that we needed such as the main window, buttons, dialog boxes, and file browser windows.

Our GUI component was designed with full modularity in mind, and works independent of the rest of the modules, meaning changes to the GUI will not affect the rest of the project, and vice versa. This has the added benefit of each module not being intrinsically linked to the GUI, which makes it easier for the rest of the modules to be integrated into other projects and automated scripts.

Communication between the GUI and the workflow manager is based on a data format protocol that we have defined.

```
13  ##############################################################################################
14  #                                                                                            #
15  #                                IMPORTANT CLASS ATTRIBUTES                                  #
16  #                                ___________________________                                 #
17  #                                                                                            #
18  # 1. programFile[0] //Global variable that holds string of path to executable               #
19  #                   //of program to be simulated                                            #
20  #                                                                                            #
21  # 2. InputParameters.text() //Global variable that holds str of executable parameters       #
22  #                                                                                            #
23  # 3. networkValueBoxList //List of network value boxes that hold network acceleration       #
24  #                        //factors. Accessed by networkValueBoxList[n].value()              #
25  #                                                                                            #
26  # 4. hwValueBoxList //List of hardware value boxes that hold hardware Acceleration          #
27  #                    //factors. Accessed by hwValueBoxList[n].value()                       #
28  #                                                                                            #
29  # 5. SavedConfigFileName.text() //class attribute that holds config filename                #
30  ##############################################################################################
31  class MPI_Simulator_GUI(QDialog):
32
33      #CONSTRUCTOR FOR MAIN GUI WINDOW
34      def __init__(self, numNetworkFactors, numHWFactors, network_connections=None, hardware_nodes=None):
35          super().__init__()
36
37          if network_connections is not None:
38              self.network_connections = network_connections
39          if hardware_nodes is not None:
40              self.hardware_nodes = hardware_nodes
41
42          self.numNetworkFactors = numNetworkFactors
43          self.numHWFactors = numHWFactors
44
45
46          self.originalPalette = QApplication.palette()
47
48          styleComboBox = QComboBox()
49          styleComboBox.addItems(QStyleFactory.keys())
50
51          styleLabel = QLabel("&Style:")
52          styleLabel.setBuddy(styleComboBox)
53
54          self.useStylePaletteCheckBox = QCheckBox("&Use style's standard palette")
55          self.useStylePaletteCheckBox.setChecked(True)
56
57          disableWidgetsCheckBox = QCheckBox("&Disable widgets")
58
59          #Call functions that create GUI elements in each quadrant of the main window
```

Fig. N1 - GUI sample code, main class definition

Appendix O

This appendix provides screenshotted proof of Varun's (our supervisor's graduate student) approval of our GUI design.

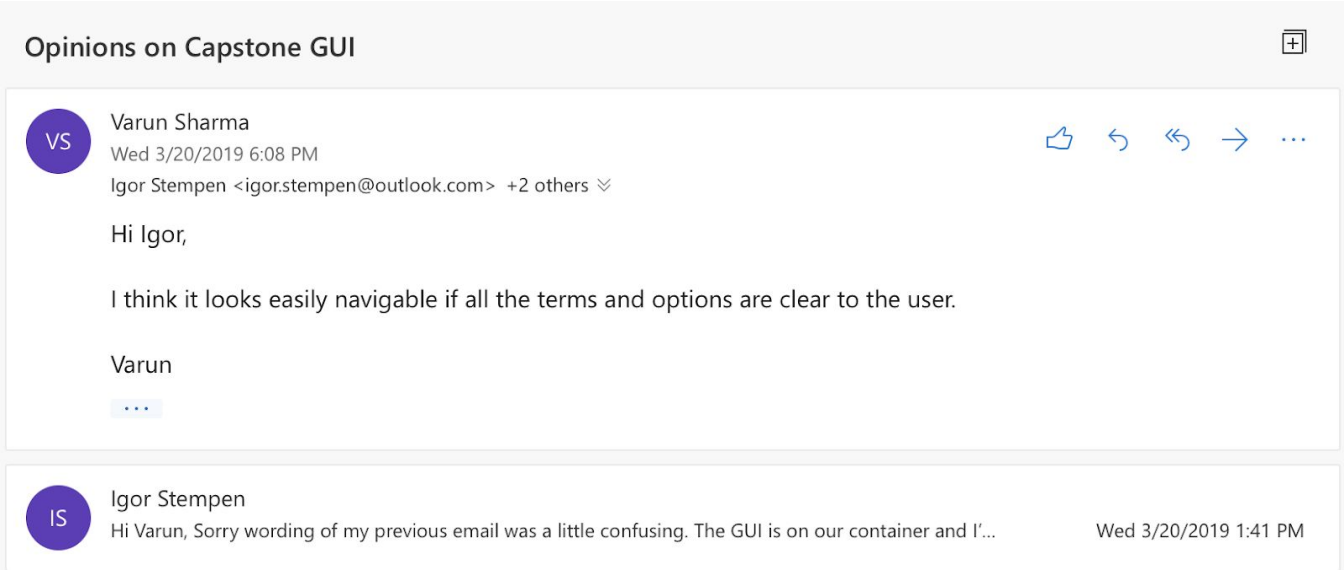


Fig. O1 - Varun's approval email

Appendix P

This appendix provides proof of our GUI's ability to save the GUI state to a config file, and then load prior saved GUI states from a config file. Since proof of this cannot be provided in static screenshots of our GUI, code samples are provided instead.

```
    def saveToConfig(self):
        print("Saving user values to config file")

        self.toConfigNetworkParamList = []

        print("SAVING NETWORK ACCEL FACTORS")

        for i in range(0, self.numNetworkFactors):
            entry = []

            entry.append(self.netSourceBoxList[i].value())
            #print(self.netSourceBoxList[i].value())
            entry.append(self.netDestBoxList[i].value())
            #print(self.netDestBoxList[i].value())
            entry.append(self.networkValueBoxList[i].value())
            #print(self.networkValueBoxList[i].value())
            entry.append(self.networkLabelList[i].text())
            #print(self.networkLabelList[i].text())

            self.toConfigNetworkParamList.append(entry)

        for i in range(0, self.numNetworkFactors):
            print(self.toConfigNetworkParamList[i])

        self.toConfigHWParamList = []

        print("SAVING HARDWARE ACCEL FACTORS")

        for i in range(0, self.numHWFactors):
            self.toConfigHWParamList.append(self.hwValueBoxList[i].value())

        for i in range(0, self.numHWFactors):
            print(self.toConfigHWParamList[i])

        print("CALLING SAVE TO PARAMETERS FUNCTION")

        workflowScript.saveParameters(self.toConfigNetworkParamList, self.toConfigHWParamList, self.savedConfigFileNam
```

Fig. P1 - GUI saveToConfig code snippet

```
def openFromConfig(self):
    print("trying to load config file")
    self.configFile = QFileDialog.getOpenFileName(self, 'Open File')
    print(self.configFile[0])
    self.network_connections, self.hardware_nodes = workflowScript.loadParameters(self.configFile[0])
    print("Read following values from config file")
    print(self.network_connections)
    print(self.hardware_nodes)

    print("starting simulator with loaded config values")
    self.simulator_gui = MPI_Simulator_GUI(len(self.network_connections), len(self.hardware_nodes), self.network_connections, self.hardware_node
    self.simulator_gui.show()
```

Fig. P2 - GUI openFromConfig code snippet