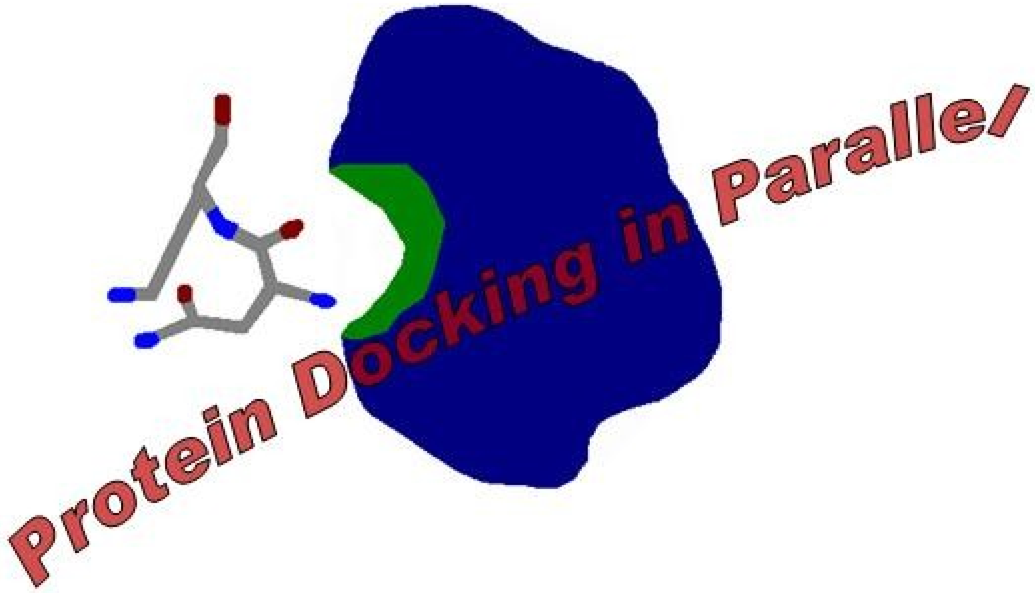
Improving Protein Docking Efficiency with General Purpose Computation for Graphics Processing Units – Formal Proposal



Project Team E-51:
Timothy Blattner
David Hartman
Andrew Kiel
Adam Mallen
Andrew St. Jean

Faculty Advisor:
Dr. Craig Struble

Table of Contents

I. Executive Summary

Dr. Struble has commissioned our team to increase the efficiency of protein docking experiments by implementing parallel processing techniques in the existing AutoDock docking software. Currently AutoDock uses serial processing only and a significant increase in speed is expected with the implementation of parallel computation. Parallelization is facilitated by the CUDA API designed by NVIDIA which allows a program written in C++ to be automatically compiled in a suitable manner to execute on a GPU. GPUs are inherently suited to the task of parallel processing due to their many individual cores which act independently.

GPUs are an attractive solution to the problem of cost and time effective parallel processing because they are common, thus cheap, and the CUDA API allows for an easy transition into learning to program for them. Upon completion of this project any current AutoDock user with capable hardware (many workstations already exist that meet minimum specifications) will be able to download our new code and use AutoDock as they always have with no apparent change in functionality to the user except for faster run time for experiments. The value of this optimization becomes apparent when one considers the fact that users typically run huge numbers of docking simulations within one experiment and time on large, powerful supercomputers is expensive.

The main advantages of AutoDock are the fact that it is a free, open-source docking simulation tool and that it uses a genetic algorithm to exhaustively search the solution space. As of this point we have shown our goal is realistically achievable by May of 2009 through the generation of a simple genetic algorithm modified to run on GPUs. This implementation of parallel processing on the GPU is proof that parallelization can be applied to the type of algorithm that AutoDock uses successfully. With the release of our final product, as well as our testing results, we are confident that users whom already have supported hardware will have no hesitation in adopting our version. While we hope our improvements will be significant enough to warrant hardware upgrades, the average user is expected to be able to take advantage of our product with no additional cost.

II. Introduction

The Customer Needs and Target Specification Document provides a layout of the problem, the needs provided by the customers for improving the AutoDock software, and our specific quantifiable goals for satisfying these needs.

The following sections are included in this document:

- Project Background
- Project Objective Statement
- Customer Needs Classification
- Target Specifications
- Problem Decomposition
- Final Concepts
- Schedule
- Economic Analysis
- Prototype Analysis

III. Background

AutoDock

AutoDock can perform a docking simulation in as little as a minute. Although this is quite fast, a drug company can go through millions of individual simulations before having the requisite data to start physically synthesizing a new drug compound [1]. We will be using parallel computing techniques to speed up AutoDock.

Genetic Algorithm

AutoDock uses a genetic algorithm to perform a docking simulation [1]. This type of algorithm uses crossover and mutation of a parent population to make a new, random population. This provides a fast, efficient way to exhaustively search the solution space. The first step is to select or generate a starting population of which the best individuals can be calculated. These best individuals are altered by genetic crossover combination and also randomly mutated to make new individuals which can then be ranked for fitness. Each new set is altered again and the process is repeated until a best solution is obtained.

Parallel Processing

Parallel processing uses multiple processors acting simultaneously to solve a computational problem [2]. By running a process in parallel, problems are solved faster because different pieces of the problem can be solved at the same time [2]. The algorithms used in the AutoDock software never take advantage of this strategy, though examination of these algorithms shows potential for speed up using parallel processing techniques.

Graphics Processing Units

Graphics Processing Units (GPUs) provide parallel processing over numerous processor cores [3]. When programming for the GPUs, it is useful to use their multi-core architecture by modifying pieces of the code to run in parallel over their multiple processors. Unfortunately, GPUs only support single-precision floating point values [4] in comparison to programming on double-precision floating point CPUs.

CUDA

One can send instructions directly to NVIDIA graphic cards using the CUDA Application Programming Interface (API) [5]. The CUDA API will be used to access the parallel processing power of the GPUs. In this way, we can modify the existing AutoDock C++ code to harness the power of parallel computing and improve the efficiency of the program.

IV. Project Objective Statement

Problem Statement

The popular AutoDock software uses serial processing techniques to simulate protein docking. AutoDock does not take advantage of the parallel processing power of modern computer architecture such as multi-core processor chips and graphical processing units.

Project Objective

Reduce docking time in the AutoDock simulation by using parallel processing on GPUs, which will be tested and implemented by working 20 man hours per week until April 29, 2009 without funding.

V. Customer Needs Classification

Customer Needs Categories and Details

1. Accuracy/Precision
    - AutoDock software should not cause a significant loss of accuracy in the program's output.
    - Customers should be satisfied with the output despite the loss of precision from performing computations on single precision GPUs instead of double precision CPUs.
    - Provide results of CUDA AutoDock test data and differences in comparison to old AutoDock results for customer review.
2. Compatibility
    - Customers without the necessary CUDA capable GPU hardware will be able to compile and run AutoDock the same as before our changes to the code.
    - CUDA AutoDock should make changes transparent to the user.
    - Command line tools of AutoDock are consistent with prior version.
    - Code that we add/modify in the software should follow the same styles as the existing code and provide clear comments for the customer to understand.
    - Released as open source software.
3. Cost
    - We should provide a low cost mechanism for parallel processing in order to speed up the runtime of an AutoDock docking simulation. We plan on using both high-end and low-end GPUs to meet this need. So, since this design decision has already been made, the real customer need is making sure that the cost of the CUDA capable GPU hardware is worth the speed up trade-off.
    - Our modified AutoDock code should be released as open-source software.
4. Speed
    - CUDA AutoDock will provide a speedup of at least 10 times.
    - We should provide measurements of speedups achieved for customer review.

Prioritized Customer Needs

1. Has minimal loss of accuracy in results
2. Is precise enough for customer to use
3. Cost of CUDA capable hardware is worth the speed up our modifications provide
4. Runs as usual if customer does not have CUDA capable hardware
5. User interaction with AutoDock commands when running our modified program is identical (or at least similar) to previous AutoDock interface
6. Results in a speed up of an order of magnitude or more with high-end hardware
7. Released as open source software
8. Coding style remains consistent with previous AutoDock code and provides clear comments
9. Provides differences in accuracy and precision for customer review
10. Provide measurements of speed tests for customer review

VI. Target Specifications

The following metrics table contains a list of all specifications formulated by our list of different customer needs. Each entry contains the metric number, the associated customer need, the name of the specification, the type of units used to measure the specification, a pair of marginal and ideal values which represent our project's goal, and a relative weight of the importance of meeting each goal (1 is the least important and 5 is the most important).

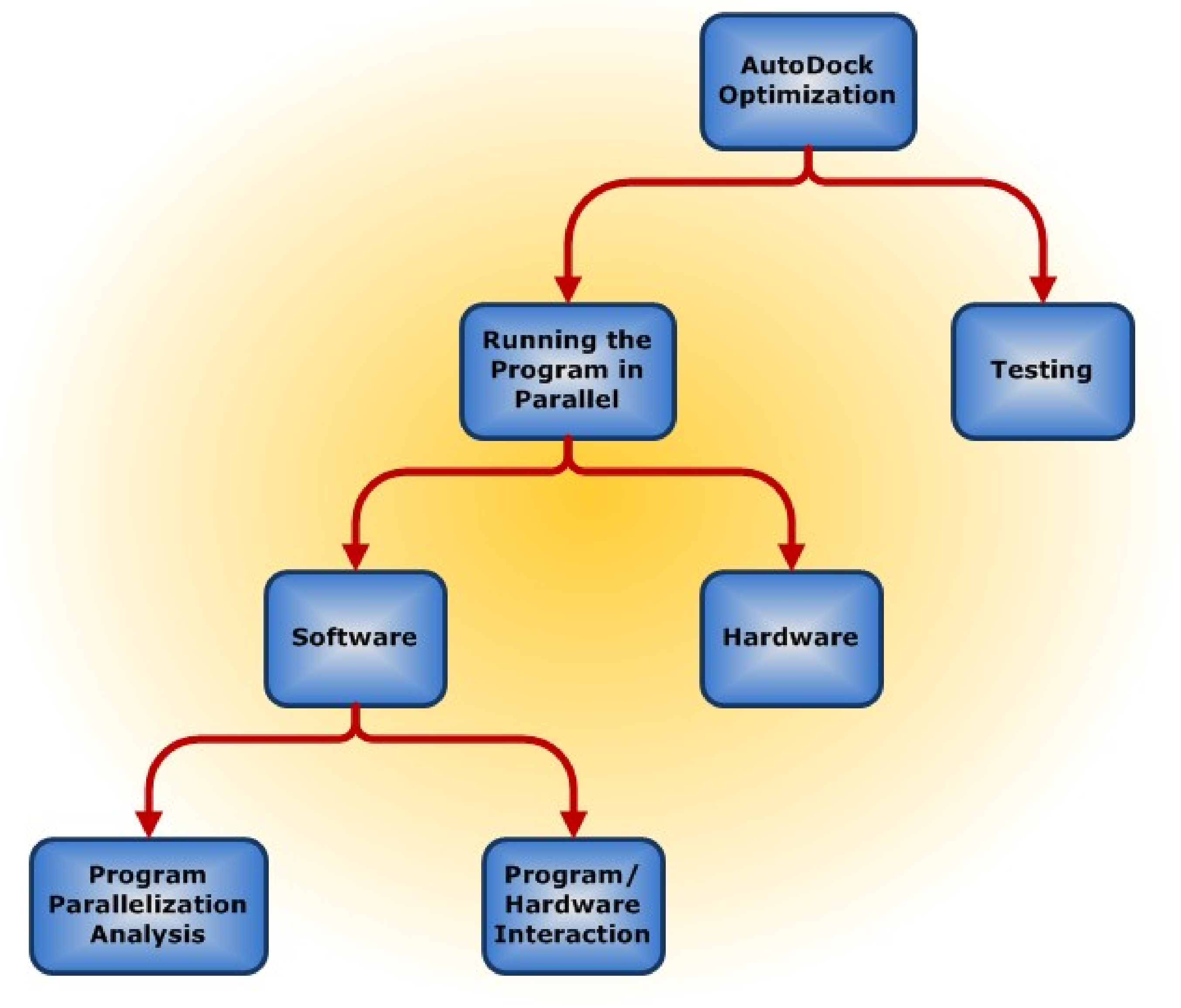
Metric No.	Need	Metric	Units	Marginal Value	Ideal Value	Weight
1	1	Difference in mean value between our AutoDock results and original AutoDock results	P-value of T-statistic	(1-P) < 0.05	(1-P) < 0.01	5
2	1	Difference in variance between our AutoDock results and original AutoDock results	P-value of F-statistic	(1-P) < 0.05	(1-P) < 0.01	5
3	2	Precision of our AutoDock results, measured as significant digits	Significant Digits	>= 5	>= 10	5
4	3,6	On high-end hardware, factor of speed up	Percentage	> 1000%	> 1500%	4
5	3	On low-end hardware, factor of speed up	Percentage	> 200%	> 1000%	1
6	4	Runs old AutoDock as normal on machines without supported hardware	Binary	Yes	Yes	5
7	5	Interface is identical (or similar)	Binary	Yes	Yes	4
8	7	Released as open source	Binary	Yes	Yes	5
9	8	Coding style is consistent with proper comments	Binary	Yes	Yes	2
10	9	Provide measurements of speed tests for customer review	Binary	Yes	Yes	2
11	10	Provides differences in accuracy and precision for customer review	Binary	Yes	Yes	2

VII. Problem Decomposition

Introduction

Before generating concepts and deciding on a final solution, we need to decompose our problem. Once the sub-problems have been identified, the following section will address the sub-problems for which solution methods have already been decided upon. The next section will discuss the generated concepts the team has come up with as possible solutions to the remaining, unsolved, sub-problems.

The following diagram outlines the basic subdivisions of our problem.


AutoDock Optimization
Running the Program in Parallel
Testing
Software
Hardware
Program Parallelization Analysis
Program/ Hardware Interaction

AutoDock Optimization

At the most basic level our project requires solving two distinct problems. One is the actual parallelization of the program, and the other is the methods and analysis of testing the differences in speed, accuracy, and precision between our program and the original.

Running the Program in Parallel

There are two aspects of the process of parallelizing the program which we have made into separate sub-problems. The first problem is deciding what hardware to use. The other sub-problem involves the software characteristics of parallelizing the program. This software aspect includes both the problem of where to run code in parallel and how to run that code on the chosen hardware.

Hardware

The basic hardware problem this project faces is the decision of a hardware medium which allows for the running of (and possibly optimizes for) parallelized programs.

Program/Hardware Interaction

Running parallelized programs is in essence different from running serial programs because it requires a fundamentally different interaction between software and hardware. In order to run a parallelized program, special code is needed to actually run the software on hardware which is capable of running such programs in parallel. Determining how the parallelized software will run on the hardware has been identified as one of our critical sub-problems.

Program Parallelization Analysis

An essential problem facing the project is identifying which pieces of the program can be run in parallel without ruining its functionality. Deciding what methods of code analysis will allow the team to efficiently discover the sections of the program which can be run in parallel is the basic aspect of this sub-problem.

Testing

Besides the actual modifications to the program, the other problem facing the project is decided on methods of testing the differences between our modified AutoDock program and the original. Because of the stochastic nature of the algorithms used by the program, intense testing is required to convince customers that the results of our program are not statistically different from those of the original program. The problem is basically defined as the decision of an appropriate testing method which will yield sufficient results to convince customers that there is no significant difference between the programs.

VIII. Final Concept

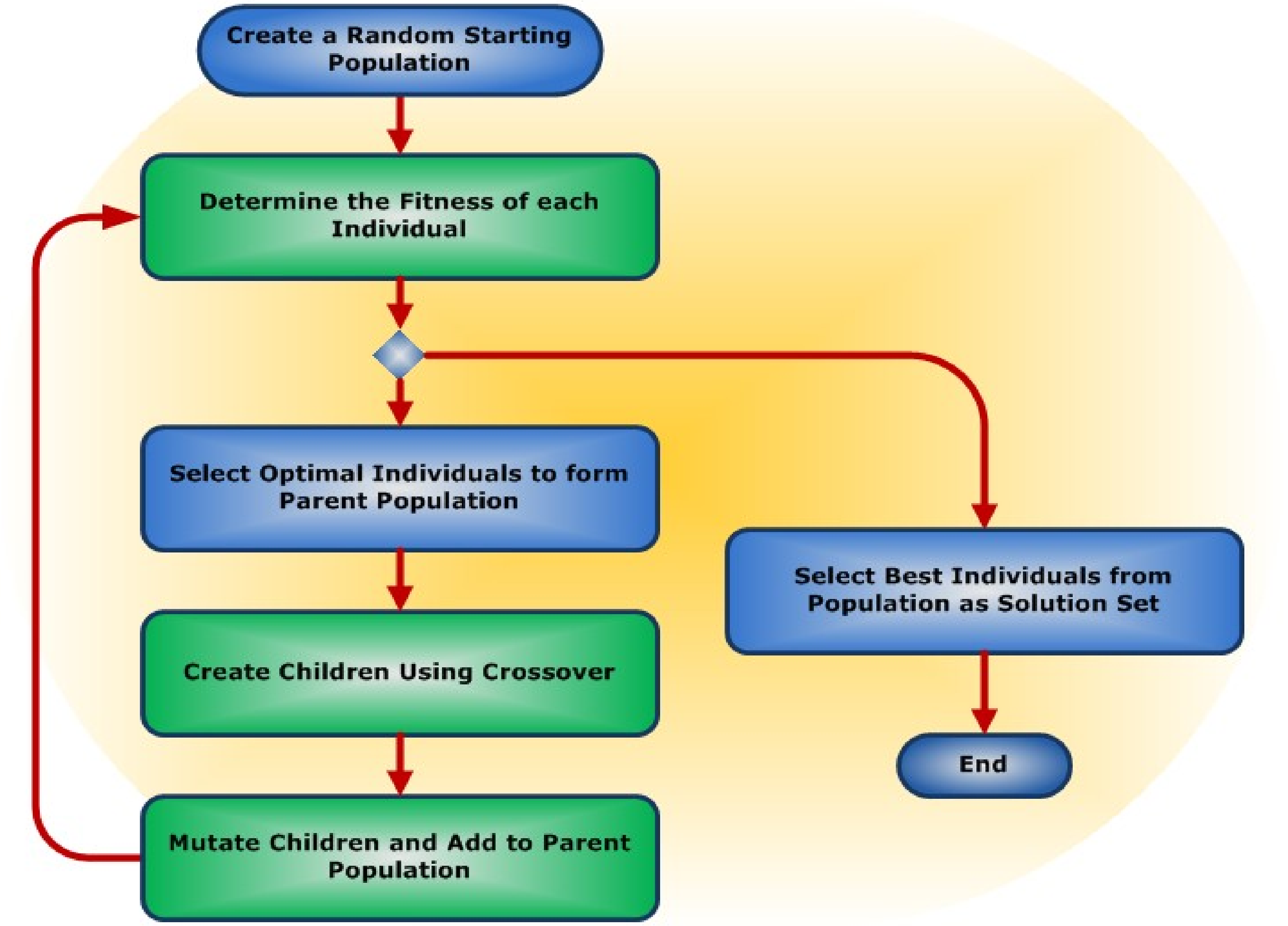
After considering the needs of our customers and decomposing our problem, we have selected a final concept that will be implemented in the coming semester. The following outlines the design decisions we have made for our process of program analysis, our methods of parallelization, our process for modifying the program, and our method of testing the results of our modified program against those of the original.

Program Analysis

Because we found no simple tool which could adequately reverse engineer the AutoDock code into decipherable diagrams, we decided to use Dr. Struble's overview of AutoDock's genetic algorithm to identify which parts of the AutoDock program we will modify to run in parallel [10]. This means that we will have to use simple code review to find the actual code in the program that represents the parts of the algorithm which we have identified to be safe places to run in parallel.

Genetic Algorithm Analysis

The following diagram outlines the genetic algorithm used by AutoDock:


Create a Random Starting Population
Determine the Fitness of each Individual
Select Optimal Individuals to form Parent Population
Create Children Using Crossover
Mutate Children and Add to Parent Population
Select Best Individuals from Population as Solution Set
End

The sections highlighted in green are those identified as steps which can be safely run in parallel. Notice that these sections perform operations with individual members of the population which are independent of the results of operations performed on other members. Specifically, the fitness (or energy) calculations of each member of the population is independent of fitness calculations of other members, the crossover reproduction process to form children does not depend on the creation of any other children, and the mutation of an individual member of the population does not depend on the result of the mutations of any other member. This means that our code review will focus on finding the implementation of these three steps of the algorithm.

Implicit Parallel Programming

Because we have chosen to run our code on NVIDIA GPUs that are CUDA capable, which uses implicit parallel programming when compiling code, our final selection will also use implicit parallel programing. Explicit parallel programing does not follow in the constraints of the hardware that is defined in out project.

Data Parallelism

Data parallelism is ideal because CUDA forces programmers to mark off sections of sequential code that will be run in parallel. Due to the nature of the CUDA API, we are bound to using data parallelism in our implementation.

Domain Decomposition

We have concluded that using functional decomposition could cause problems in parallel computing. These problems might include conflicts in the AutoDock algorithm. By using domain decomposition, we are able to send independent portions of the docking procedure to be run in parallel on the hardware. Running independent portions is more ideal rather than portions that might be dependent on each other.

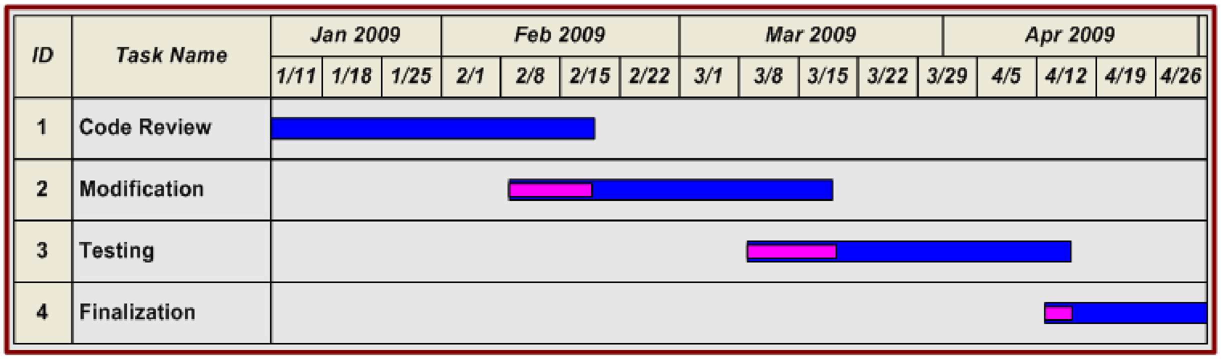
CUDA Translation

CUDA is the API of choice for NVIDIA GPUs. The group has studied CUDA and its interaction with the hardware and has discovered optimizations that can be implemented. One optimization that can greatly increase performance in memory interaction on GPUs in memory coalescing. As we continue to work with CUDA and learnings its complex system, we will further discover more optimizations that we can use.

Testing

After modifying the program we need to test that the results from our modified AutoDock do not statistically differ from those of the original AutoDock. We plan on running multiple trial runs with the same input on both the original and modified versions of AutoDock and statistically comparing the distributions of results. Our ideal tolerance of difference is 1% and our marginal tolerance is 5%. We will also run the same experiments on a machine with low-end GPU hardware so we can compare the gain in speed on low-end and high-end GPU hardware. For our project we will use Dr. Struble's gpgpu machine for tests on high-end GPU hardware and the Systems Lab's MicroWulf cluster for tests on low-end GPU hardware (specifications for both of these machines can be found in the appendix). For sample input we will be using two data sets from researchers here at Marquette which include sample ligands and proteins from Dr. Struble's Bio-Informatics lab and from Dr. Sem and his assistant Andrew Olson's research.

IX. Schedule



We have divided our schedule for the Spring into four tasks. The first three tasks are all around a month in length. The pink portions on the above waterfall model of our schedule represent our hard and soft deadlines for our tasks.

Code Review

Our first task will consist of code review. We will have two weekly meetings on Tuesdays and Thursdays from 2:00pm to 4:00pm. We will divide our group into two teams. One team will have two programmers, and the other team will have two programmers and one biomedical engineer. We will start this task by searching in main.cc for any genetic algorithm function calls. Once found, we will analyze these function calls throughout the code to determine where there are steps that can be run in parallel. We are also going to search the AutoDock files for genetic keywords, such as population, mutate, parent, etc. The files containing these keywords may also contain steps of the genetic algorithm that can be run in parallel.

Modification

Our second task will be program modification. We will keep the same meeting times as code review, however we will not be split into teams. Since code review pinpointed the locations for running the code in parallel, we will now be able to make necessary changes to the program. We will perform domain decomposition on each step of the algorithm that can be run in parallel, as well as implementing CUDA's ability to provide memory coalescing and shared memory management.

Testing

Our third task will be our testing phase. This task will include creating a normal probability plot to determine if the results of trial experiments are distributed normally. If the results are normal, we will perform standard T and F-Tests to measure the difference in mean and variance respectively. If the results are not normal, then we will use non-parametric statistics to analyze our results.

Finalization

Our fourth task is the finalization process. This task will include release of our testing results from our third phase, as well as release of the code to our customers. Upon release we will have our customers run our code in order to make sure our customer needs are satisfied.

X. Economic Analysis

Hardware Cost

We are testing our code on two machines (GPGPU and MicroWulf) in order to analyze the difference in speed up between high and low end GPUs. The GPGPU machine contains high end NVIDIA GPUs and has a total cost of $2350. The MicroWulf contains low end NVIDIA GPUs and has a total cost of $1800 [10].

Work Cost

We estimate that in addition to our scheduled meetings we will each put in six hours of work in between meetings. This time will be devoted to both our scheduled tasks and our deliverables for the Spring. This results in ten man hours per week per person or fifty total man hours per week for fifteen weeks. This equals 750 total man hours at a proposed salary of $25 per hour. This results in a total work cost of $18,750 and a total project cost of $22,900.

XI. Prototype Anaylsis

Introduction

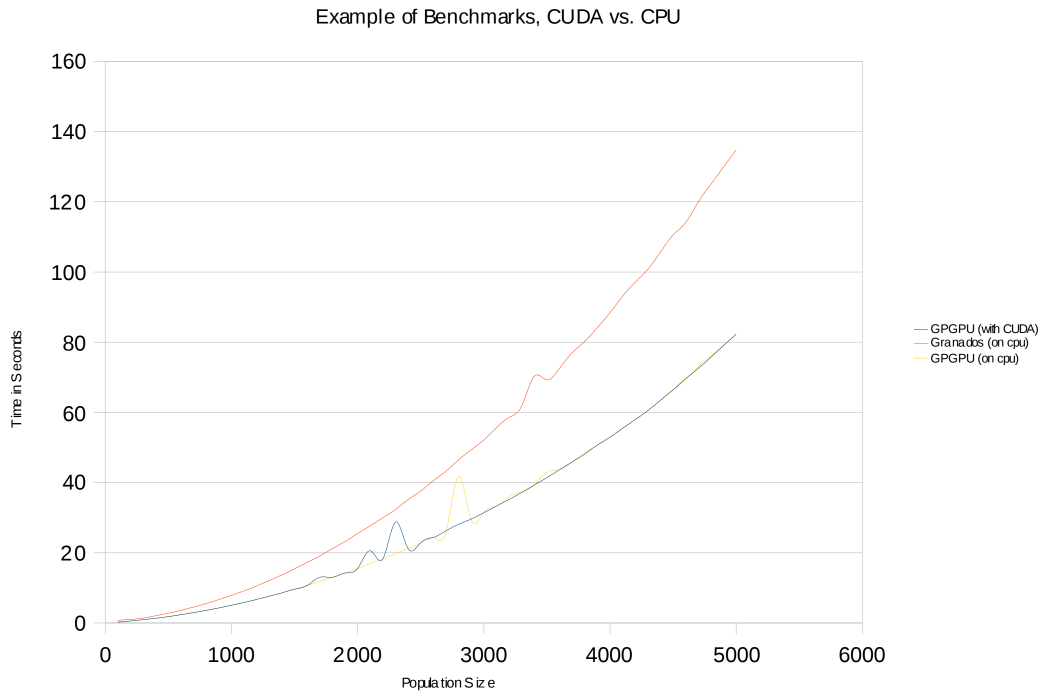
Autodock has a few charactistics that must be analyzed for our prototype. Autodock is implemented using C++ and uses a genetic algorithm. For our prototype we decided to identify a program that is coded in C++ and uses a genetic algorithm, which mirrors the autodock functionality. By creating the code that runs on GPUs from C++ code, we exemplify the skills needed to be successful in our project.

Prototype

Our prototype is a solution to the traveling salesman problem using a genetic alrogithm. The original code was implemented by Dr. Thomas Pederson from the Department of Computer Science in Umea University. The traveling salesman is a classic example of a computationally difficult problem. The problem consists of N number of cities with distances between each city. The salesman must travel to every city in the shortest amount of distance possible, and return back to the original city from which the salesman started. The algorithm used by Dr. Pederson uses similar concepts to the genetic algorithm used in Autodock with the exception of mutations. The portion of the traveling salesman code that we decided to parallelize is the parent selection. Although this section is not computationally intense, it is simple enough so that we can show our abilities with CUDA.

Results

We ran three tests. One on a standard CPU on granados.mscs.mu.edu, the second on the high end CPU on gpgpu.mscs.mu.edu, and the third on the high end GPU on gpgpu.mscs.mu.edu. We ran fifty test runs with population increments of 100. We compared the tests as shown in the graph below.

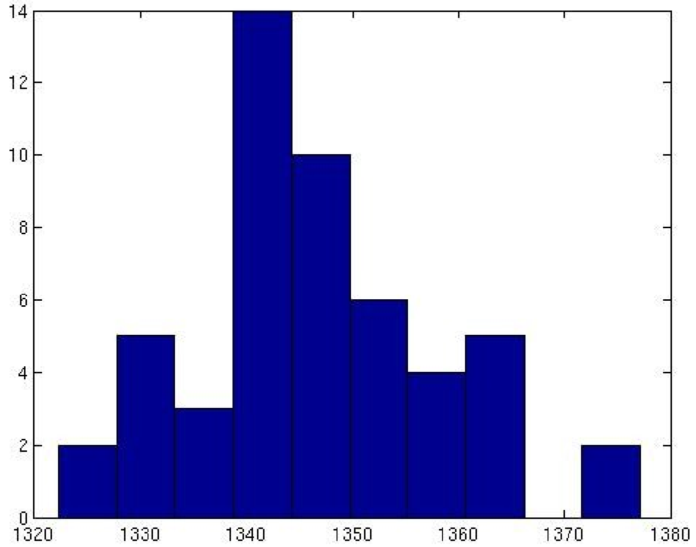


Although there is no speed improvement in our prototype, we want to show our strategies for analyzing the AutoDock modifications.

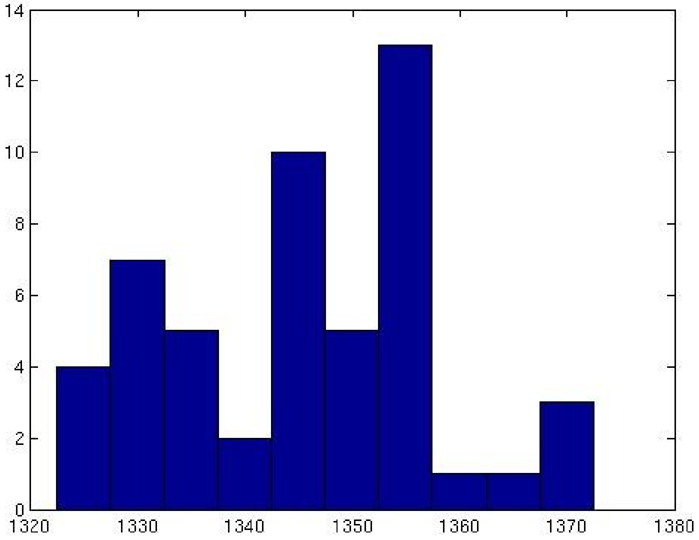
Testing

As an example of the tests we will run after making modifications to AutoDock, we have used similar analysis on the results of the prototype trial runs. We ran 51 trial runs of the normal CPU version and 51 trial runs of our modified CUDA version, both with the same graph as input and on the same GPGPU machine. The following histograms show the distributions of results from the two experiments.

This histogram shows the distribution of results of the CPU experiment:



This histogram shows the distribution of results of the GPU experiment:



We ran a two sample T-Test on the two distributions in MATLAB and got a P-value of 0.5448. Though this is far under the 0.99 we were aiming for or even the 0.95 we identified as our marginal error tolerance for the P-value, we are not as concerned with the accuracy of these tests because this is just the prototype. However, it does show that we are capable of running the sort of statistical analysis that we plan on using for our actual modifications to AutoDock.

XII. Appendix

System Specifications for the gpgpu.mscs.mu.edu Machine

Processor	Dell Precision T3400 Convertible MiniTower Processor E4500, 2.20GHz, 800 2MB L2, 525W
Video Card	Dual nVidia Quadro FX3700 512MB dual DVI
Ram	2GB, 667MHz, DDR2 NECC SDRAM Memory, 2X1GB
Hard Drive	80GB SATA 3.0Gb/s with NCQ and 8MB DataBurst Cache

System Specifications for MicroWulf

Processor	Four AMD Phenom 64 9500 Quad-Core 2.2 GHz
Video Card	Four MSI NX8400GS nVidia GeForce
Ram	Four 1GB, 800MHz, DDR2 x2
Hard Drive	500GB Western Digital 7200 RPM

XIII. References

[1] AutoDock. 24 September 2008. <http://AutoDock.scripps.edu>

[2] Blaise Barney., "Introduction to Parallel Computing." 2007 https://computing.llnl.gov/tutorials/parallel_comp/#Whatis

[3] "What is CUDA?" CUDA Zone. NVIDIA. 14 October 2008. <http://www.nvidia.com/object/cuda_what_is.html>

[4] Halfhill, Tom R. "Parallel Processing with CUDA, Nvidia's High-Performance Computing Platform Uses Massive Multithreading." 1 January 2008. NVIDIA. 14 October 2008. <http://www.nvidia.com/docs/IO/55972/220401_Reprint.pdf>

[5] "CUDA education." CUDA Zone. NVIDIA. 14 October 2008. <http://www.nvidia.com/object/cuda_education.html>

[6] "Random seed." Wikipedia.org. 14 October 2008. <http://en.wikipedia.org/wiki/Random_seed>

[7] Kumar, Vipin, Ananth Grama, Anshul Gupta, and George Karypis. Introduction to Parallel Computing: Design and Analysis of Algorithms. Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc. 1994.

[8] Chandy, K. Mani and Stephen Taylor. An Introduction to Parallel Programming. Boston: Jones and Bartlett Publishers, Inc. 1992.

[9] Struble, Craig Ph.D. Personal interview. 31 Oct. 2008.

[10] Struble, Craig Ph.D. Personal interview. 14 Nov. 2008.