UNDERGRADUATE THESIS PROJECT FINAL REPORT
School of Engineering and Applied Science
University of Virginia

Multithreaded Implementation of Leukocyte Identification Algorithm

Submitted by

Donald Clay Carter

Computer Engineering

STS 402

Section 5 (2:00 p.m.)

April 5, 2007

On my honor as a University student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in Science, Technology, and Society Courses.

Signed _____

Approved _____ Date _____
Technical Advisor – Kevin Skadron

Approved _____ Date _____
Science, Technology, and Society Advisor – Bryan Pfaffenberger

Table of Contents

Glossary of Technical Terms

algorithm – a set of instructions that when complete will accomplish a specific task

algorithm parallelization – application of parallel programming techniques to an existing algorithm to create a version that can execute at least partially in parallel

concurrent programming – see parallel programming

microscopy – method of image capture using microscope probes; in this case probes are inserted into blood vessels of live patients - in vivo (Lach et al, 2006)

multithreaded program – a parallel program implemented as a series of shared memory execution threads that emanate from a single main traditional process

parallel programming - the process of splitting a problem into several sub problems, solving the sub problems simultaneously, and combining the solutions of sub problems to get the solution to the original problem (Xavier and Iyengar, 1998)

throughput – measure of processing capacity in terms of amount of data processed over an interval of time

uniprocessor computer – standard single processor Von Neumann machine; in this case a traditional personal computer

Abstract

Millions of people worldwide suffer from conditions related to deficiency in inflammatory response. Review of microscopy video allows for analysis of the rolling, arrest, and adhesion of leukocytes. Studying the motion of leukocytes will assist researchers in designing new treatments for inflammatory disorders. Toward this end, researchers have designed leukocyte detection and tracking algorithms that allow microscopy video to be analyzed by computer and the results to be presented to physicians. These techniques, while effective, currently operate at a throughput level that hampers effectiveness due to the processing time involved. To ease this difficulty, it is proposed that the current detection and tracking algorithms be parallelized. The student will design a new parallel form of the detection algorithm and implement prototypes of the new algorithm on a GPU architecture. These efforts resulted in an increase of throughput by two orders of magnitude and correspondingly allowed for a reduction in program execution time of two orders of magnitude.

I. Introduction

Understanding of white blood cell behavior is critical to learning more about medical conditions resulting from malfunction in inflammatory response. Researchers in the University of Virginia departments of Electrical and Computer Engineering and Biomedical Engineering have developed algorithms for identifying, counting, and tracking white blood cells (leukocytes) during *in vivo* video microscopy (Lach, Acton, & Skadron, 2006). Currently implemented versions of the algorithm achieve a processing throughput level that only allows for processing of microscopy imagery after data collection is complete. This project aimed to increase computational throughput by three orders of magnitude and allow real-time processing of imagery by designing a multithreaded implementation of the detection algorithm.

The student individually accomplished the project as a continuation of research into detection algorithm throughput increase conducted by members of the departments of Electrical and Computer Engineering and Computer Science (Wolpert, 2006). The student's project was initiated in September 2006 and is currently in progress with completion anticipated in April 2007. The scope of the project is to implement the most processing intensive sections of the detection algorithm in a parallel architecture and time permitting to design an end-to-end application that incorporates these parallelized sections into the overall detection algorithm.

As of this writing, the project is still in progress with completion anticipated in June 2007. The student has designed multithreaded prototypes of the most computationally intensive sections of the detection algorithm for the Nvidia GPU. Of the two prototypes designed by the student, one is fully functional yet produces results that do not fully

coincide with the results produced by the uniprocessor algorithm. Preliminary timing results for this prototypes suggest that GPU processing requires approximately 85 ms. Comparing this value to the 1.01 s or 1010 ms processing time on the uniprocessor yields a two order of magnitude decrease in processing time. This prototype will be revised to generate fully accurate results and the remaining prototype will be implemented and verified. Further steps to achieve the three order of magnitude processing time reduction hypothesis require correct results from the prototypes and will be achieved within the project time period specified above.

Discussion of the project requires review of relevant technical literature, examination of the social and ethical context, and in-depth examination of efforts made by the student to accomplish the project. Full understanding of the need for the project requires recognition of the crossroads that computer science faces regarding increase in processing power and the resulting efforts to drive new parallel architectures (Lach, Acton, & Skadron, 2006). As with any engineering area, the project retains unique social and ethical context and the student has considered this context while completing the project to act as a responsible engineer. Finally, continuation of the research performed by the student necessitates in depth discussion of the student's efforts in completing the project and the results achieved. This analysis will allow future research to build upon the conclusions gathered from this project and further the research accomplished by the student.

II. Social and Ethical Context

Primary social contributions made by the project are in the area of medical research regarding inflammatory response. Inflammatory disease is a direct result of leukocytes rolling along the internal surface lining of small blood vessels known as postcapillary venules. By gathering data on the number and velocity of these rolling leukocytes it is possible to greatly increase understanding and treatment options for inflammatory diseases (Ray, Acton, & Ley, 2002). The rolling and eventual adhesion of the leukocytes immediately precedes inflammation (Kunkel, Dunne, & Ley, 2001; Ley, 2001). Researchers can potentially advance their understanding of inflammatory response based on the results of the project.

The quality of microscopy imagery leaves much to be desired, particularly the resolution and depth perception of the produced imagery. Innovative new imagery technology such as infrared, optical, and microwave imagery techniques is needed (Johnson, Turnbull, & Fitzsimons, 1999). Increased image resolution and depth perception will ultimately require more processing time due to the increased size of the data set to be processed, in this case the image. However, due to the expected three order of magnitude increase in processing throughput of the parallelized algorithm, this is not expected to be a problem (Lach et al, 2006). These contributions to inflammatory research and imagery technology have the potential to benefit society.

Development of parallel architectures plays an important role in the future of software development in both a social and economic sense. The project makes a contribution to research in the field of concurrent programming on next generation hardware that is beginning to enforce a paradigm shift in software development (Pancake, 1991;

Metropolis & Rota, 1993). The idea that processing power will double every 18 months, known as Moore's Law, has defined advances in computer architecture for the past 30 years (Twist, 2005). In a 1997 article in *Wired* magazine, Gordon Moore expressed that "in about a decade, we're going to see a distinct slowing in the rate at which the doubling occurs" (Leyden, 1997, p. 1). Moore would find his prediction for the industry accurate once again.

As predicted by Gordon Moore, increasingly inadequate heat dissipation has led the processing throughput of computer chips to a plateau. Moore himself acknowledged the reality and claimed his law was dead in an interview with *Techworld* in April 2005 (Dubash, 2005). This new reality impacts both the computer industry and academia. To counter the stall of processing power increase, more processors are added to continue the doubling effect. The symbiotic relationship of hardware and software will emerge as multiprocessor hardware development drives new software practices to utilize the hardware. In this case, computer scientists knowledgeable in parallelizing algorithms are needed. Given this need for software developers with new abilities, university programs of study must include instruction in the art of parallel programming (Kurtz, 1998; Howland, 2006). Analysis of the results of this project and its effectiveness with the chosen parallel architecture will provide development insights to the developers of new parallel multi-processor architectures in terms of what applications are suited for various parallel architectures. The project also identifies the new wave of parallel computing requirements and communicates the necessity of learning algorithm parallelization to computer science students.

Ethical concerns that the project raises are also worthy of examination. Frequently examined ethical issues regarding testing on live mice apply to the collection of test microscopy data (UVa Health System, 2003). However, the student only encountered previously captured microscopy imagery and in no way dealt with any animal testing. Issues regarding testing with live mice aside, the issue of personal privacy can be raised when discussing the leukocyte detection and tracking algorithms. Microscopy imagery analysis provided by the leukocyte detection algorithm can be misused if placed in the wrong hands. Health care providers and marketing firms can target individuals with known inflammatory conditions (Regan, 2006). The security of systems that store such personal information is always in question (Strassberg, 1996). No safety issues are encountered by the student during the project. The student is fully aware and will comply with University of Virginia standards regarding recognition of the work of others; the student is a firm believer in the University honor system and holds firmly to the standards of conduct outlined therein. In addition to UVa standards, the student will comply with Association of Computing Machinery guidelines, particularly regarding intellectual property and contributing to society (Association of Computing Machinery Council, 1992). Aside from the potential privacy concern, the leukocyte detection program does not present notable ethical concerns.

Efforts to complete the project have few potential problems, yet these concerns are of great magnitude and worthy of examination. Though data on tracking of leukocyte motion will lead to improved treatment, dependence upon this method of advancing inflammatory research is risky and not recommended. Analyses of the detection and tracking algorithms and discussions with the author have led to an intimate understanding

of the code and awareness of the need for further development to determine more concrete characteristics of leukocytes (Cvijetic, 2006). Another important issue of note regards the uncertain future of parallel architecture development. As mentioned previously there is and will continue to be a great need for computer scientists trained in the art of algorithm parallelization. Without efforts such as this project, the push for parallelization will not gain sufficient strength and the computer industry will suffer. The members of the academic and industrial areas of computer science must embrace this paradigm shift for parallel architectures to succeed (Talia, 1997). The student is fully aware of these potential pitfalls and has proceeded as planned while remaining aware of the progress made on the issues during the scope of the project.

Economic and social impacts of new parallel architectures, social contributions in the area of inflammatory research, ethical privacy concerns, and the potential problems described above are all directly relevant to this project. In completing this project, the student has increased his awareness of these external issues and improved his abilities as a software engineer. Awareness of the underlying issues in technology is crucial to the student's continued successful development as an engineer. Careful consideration of all the effects of the project has allowed the student to complete the project and make a contribution to society.

III. Review of Technical Literature

This project serves the primary purpose of furthering research on inflammatory conditions while at the same time demonstrating strengths and weaknesses of cutting edge parallelization techniques. Millions of Americans suffer from medical conditions resulting from malfunction in inflammatory response, including rheumatoid arthritis, asthma, multiple sclerosis, and colitis (Lach, Acton, & Skadron, 2006). Studying the molecular mechanisms of leukocyte rolling, arrest, and adhesion allows for greater understanding of the methodology of inflammatory disease (Acton, Wethmar, & Ley, 2002). Early versions of leukocyte tracking software required ten minutes of processing per video frame; processing an entire 100,000 frame microscopy tape, recorded over an hour in clinic, would require nearly two years of processing time on a standard personal computer. The project encompasses three major technical issues, specifically algorithm parallelization, cutting edge parallel architectures, and image processing techniques used to detect and track leukocytes in microscopy video. All of these issues are directly related to the project content and understanding of each is essential to the successful completion of the project.

Algorithm parallelization is involved in the design and implementation of a multithreaded version of the leukocyte detection algorithm. Designing the detection algorithm implementation for the parallel architectures requires knowledge of the involved parallel architectures. Determining parallelizable sections of the detection algorithm and modifying the implementation to a multithreaded version directly involves using intimate knowledge of the image processing techniques utilized in the algorithm (Cvijetic, 2006). All of the described activities are critical to the successful completion

of the project, and therefore, the knowledge requisite for each activity is equally critical to the successful completion of the project.

The first major technical issue critical to the project is algorithm parallelization. Creation of a parallel algorithm allows for parallel computations to be executed. Xavier and Iyengar (1998) define parallel computation as, "the process of splitting the problem into several sub problems, solving the sub problems simultaneously, and combining the solutions of sub problems to get the solution to the original problem" (p. 3). Programs that accomplish tasks in this way utilize multiple threads of control; independent threads with shared memory require certain sections of code, called atomic or critical sections, to be executed sequentially without interruption. To achieve this synchronization methods such as semaphores, condition variables, and barriers are used to guarantee that sequential execution occurs without interruption (Andrews, 2000). Many strategies exist for designing parallel algorithms, including data, task, and pipeline parallel models. Particularly applicable to the leukocyte detection algorithm is the data-parallel model, a type of parallelism that is accomplished by executing the same operation concurrently on different data items (Grama, Gupta, Karypis, & Kumar, 2003). Identifying other possibilities for parallelization in the leukocyte detection algorithm can be achieved by using the profiling tool gprof. Gprof measures the performance of an executing program and identifies bottlenecks that reduce the overall computational efficiency and increase runtime of the program (Fenlason & Stallman, 1988). Utilizing these models and techniques assisted the student in designing the multithreaded implementation of the leukocyte detection algorithm.

The cutting edge architectures used to execute the multithreaded design are extremely important to the success of the project. Within the past several years, the Graphics Processing Unit has emerged as a potent parallel architecture (Atanasov, 2005). The design of the GPU as a massive arithmetic computational device allows for new uses, particularly computing large sets of operations found in research (Dokken, Hagen, & Hjelmervik, 2005). ATI has developed a direct programming interface called CTM that allows the user to directly utilize the processors contained in the ATI GPU processing array (ATI, 2006). NVIDIA released the Compute Unified Driver Architecture (CUDA) in November of 2006 as a software interface for issuing and managing computations on the GPU; this software is notable particularly for its ability to execute general programs on NVIDIA GPUs without mapping the program onto a graphics API (NVIDIA, 2006). Researchers can use CUDA to execute existing scientific programs on the GPU without reformulating them as traditional graphics problems and easily utilize the significant parallel processing abilities of the GPU (NVIDIA, 2006). IBM has also released a parallel architecture known as the Cell Broadband Engine that utilizes a controller processor and a series of slave data processors (IBM, 2006). The Cell chip, as it is commonly known, implements an innovative master-slave relationship between processors and demonstrates significant potential towards task level parallelism (Gschwind, 2006). A considerable understanding of the abilities and potential of these architectures has been fundamental to the success of the student's multithreaded leukocyte detection implementation.

The final essential area of study is the specific techniques applied in the current implementations, including the image processing functions. The key functionality of the

leukocyte detection algorithm is the construction of ellipses that signify potential leukocytes in the microscopy imagery. The ellipses are evaluated by comparison against the gradient inverse coefficient of variation, or GICOV, explained by Lach as "the ratio of mean derivative in image intensity in the normal direction (with respect to the contour) over the standard deviation" (Lach et al, 2006). The leukocyte tracking algorithm is significantly more complex, with the key component being the calculation of a motion gradient vector field, or MGVF, based on the apparent contours of the blood vessel in which the imagery takes place (Ray, Acton, & Ley, 2002). The MGVF biases the expected motion of the leukocytes in the direction of blood flow (Lach et al, 2006). An illustration of this process is found in Figure 1, Appendix A. Two external libraries are utilized in the leukocyte detection uniprocessor implementation. The Meschach library of matrix functions is utilized to compute various matrix values such as determinants (Stewart & Leyk, 1994). Also used by the detection algorithm is avilib, a library containing AVI video file manipulation functions such as reading, writing, and otherwise manipulating files (Johanni, 2001). Complete understanding and the ability to modify or replace these techniques with those deemed more optimal has been necessary for the student to complete the project.

IV. Materials and Methods

Discussion of the efforts made to complete the project requires specific delineation of the resources used by the student in project efforts as well as a detailed description of those efforts. The student utilized resources provided by the University of Virginia Department of Computer Science and purchased by Assistant Professor Kevin Skadron's research group as well as knowledge and techniques taught by a variety of Computer Science faculty. Efforts to complete the project were accomplished in according to the revised gantt chart constructed by the student (see Appendix A, Figure 3).

A. Materials

Completion of the project requires a significant amount of highly specialized resources. The student has been provided with access to computing resources such as a Linux environment and web space by the UVa Department of Computer Science. These resources were utilized to analyze and execute code on the uniprocessor computer and provided the student with a project website for storing a variety of data. Most important to the project are the new Dell Pentium D multi-processor personal computers with cutting edge NVIDIA GeForce 8800 graphics cards purchased by Kevin Skadron's research group. These machines provided an environment to execute any developed GPU code. As of this writing, the student has conducted all development in a Linux environment. Early Linux drivers available for the GeForce 8800 cards were unstable and prompted the student to investigate transitioning to the Windows environment. The development of new Linux drivers requires additional evaluation by the student to determine the optimum development environment moving forward with the project. This

determination will be primarily made by considering the ease of use of the environment given its consistent successful operation with the GeForce 8800 graphics cards. In addition the student has drawn upon experiences gained in CS414, Operating Systems, to facilitate the multithreaded development. Use of these resources has enabled the student to progress towards completing the project.

B. Methods

The time required to accomplish each objective is noted in Appendix A Figure 3, Revised Gantt Chart, reflecting changes from the student's expectations regarding time required to accomplish project objectives as seen in Figure 2.

1 - Analyze existing leukocyte detection and tracking source code

The student acquired current C source code and conducted algorithm analysis using this version. This version was developed and managed by Leo Wolpert, a graduate student in the University of Virginia Department of Computer Science. The most recent version of the detection and tracking source code is under the management of Marija Cvijetic, a graduate student in the University of Virginia Department of Electrical and Computer Engineering; meetings with Marija were held to discuss her insights into development of the algorithms. No intellectual property issues exist regarding this project; the algorithms are the property of Scott Acton and the implementations accomplished by students are the property of their advisor and respective departments. The student furthered his understanding of the source by diagramming the control flow of the algorithms in flow chart form. Analysis of the algorithms in this way allowed for visual representation and eased the study of program control flow.

2 - Execute and profile code base on uniprocessor computer

In completing this objective the student generated valid, executable object code for the provided source code and executed it on a standard single processor (uniprocessor) personal computer. The student has compiled and executed the source on a test microscopy video file. The student next utilized the profiling tool gprof to analyze potential bottlenecks in the execution flow of the algorithm implementation. Gprof analyzed the source execution and pinpointed locations in the code that required the most time or resources to execute (Fenlason & Stallman, 1988). These identified computationally intensive sections of the detection algorithm were chosen for execution on the GPU. Specifically identified by gprof were the individual functions find_ellipse and dilate_image. Within find_ellipse, potential leukocytes are identified and corresponding gradient inverse coefficients of variation are identified for later use in narrowing the potential leukocytes (see Appendix C – Find_ellipse Function). Dilate_image is a narrowing step accomplished on the source image to remove outlier values within the image; this allows more accurate distinction between actual leukocytes and false positives (see Appendix D – Dilate_image Function). A copy of the gprof analysis is provided in Appendix B.

3 - Choose GPU architecture and create simple program

Both the ATI and NVIDIA GPU architecture interfaces provide the ability for users to run standard computations on the GPU hardware without requiring that the operations be simulated as a type of standard GPU computation such as ray tracing. The student selected the GPU architecture to be used in this project based on comparisons of programming manuals from both vendors. Immediately apparent from study of the ATI

CTM interface manual was the difficulty presented in managing the GPU memory; ATI interface programming strongly resembles assembly programming and requires the programmer to directly manage GPU memory (ATI, 2006). The NVIDIA CUDA programming guide revealed that CUDA manages GPU memory internally and does not require programmer involvement; the CUDA compiler allows programmers to write code for CUDA in the C language and compile them into instructions for the GPU (NVIDIA, 2006). The advantages of CUDA over ATI CTM with regard to programming syntax and memory management allow the student to focus on developing the multithreaded prototypes rather than struggling with understanding difficult syntax and memory management issues. These advantages led the student to select the NVIDIA CUDA architecture for this project. After selecting the architecture, the student examined provided sample programs in CUDA and gained insight into their construction. The nature of the NVIDIA GPU architecture requires computations to be uploaded to the GPU, executed, and then the results of the execution downloaded from the GPU (ATI, 2006).

4 - Design parallel detection algorithm for GPU architecture

Utilizing the understanding of the source gained in the previous objectives, the student designed and implemented multithreaded prototypes of the computation intensive functions described above in the NVIDIA GPU architecture. As of this writing, the prototype for find_ellipse has been designed and implemented, yet requires additional refinement to provide comparable results to the base uniprocessor version. Due to difficulties involved with implementing find_ellipse, the prototype for dilate_image has been designed and only partially implemented. To design the prototypes, the student

identified sections of the functions that could be parallelized and determined which sections must be executed sequentially. Required sequential sections are known as the critical sections of the algorithm. The discovered parallelizable sections each received their own thread. These efforts essentially provided data level parallelism; data sets were processed using the same computations in parallel to reduce processing time.

As mentioned above, substantial problems were encountered in implementation and testing of the find_ellipse prototype. The student was able to pattern the prototype design after a provided CUDA example program that executed matrix multiplications; errors made by the student in providing input data contributed to erroneous output from the prototype. After many iterative revisions of the prototype the student was able to reproduce the base uniprocessor output within CUDA using a single threaded execution model. Translating this to a multithreaded execution model has to this point led to one-third of produced GICOV values matching to the baseline GICOV values generated by the uniprocessor code. Significant findings from these efforts include design difficulties encountered, a precision difference, and timing data from executing the prototype. These findings are discussed in detail in the discussion of results chapter. Next steps in completing the project include revision of the multithreaded model of find_ellipse to reproduce the base results, full implementation of the dilate_image prototype, and verification of the dilate_image prototype through reproduction of base results. Another significant factor involves the development environment; as stated in the materials section, the student has accomplished all development in the Linux environment to this point and is in the process of evaluating the merit of continuing in Linux or switching to

Windows. Accomplishing these objectives has led the project to this point and will lead to the completion of the project.

5 - Reduction of project scope

Initial declaration of project scope included several components of the project that have since been removed. The student intended to rewrite the leukocyte tracking code, changing the language used from Matlab to C and preparing it for processing on the GPU. This objective was removed after discussions with Marija Cvijetic indicated that the tracking algorithm was significantly dated and the effort to rewrite the code to C and for the GPU was deemed too costly given the time available. Plans were also made for the student to utilize the multithreaded detection algorithm design to create a multithreaded implementation of the algorithm in an alternate parallel architecture. This objective will be attempted if sufficient time in the academic year remains after completing the previous objectives; given the time required to create the prototype for find_ellipse and remaining work on dilate_image, it is unlikely that this objective can be completed. This reduction of the project scope has allowed the student to concentrate on generating a quality implementation of the multithreaded prototypes and potentially make efforts to publish the research in an academic peer reviewed journal.

V. Results

This chapter presents the project results obtained thus far. The primary focus of the project was in designing and implementing the multithreaded prototypes. Timing data was acquired through execution of the find_ellipse multithreaded prototype and a precision difference between the uniprocessor and multithreaded output was encountered. The results are summarized in a table in Figure 4, Appendix A.

A. Algorithm Parallelization Approach

The design experience of the student can provide insight to future researchers of the necessary time investment to complete such a project. The approach taken by the student in analyzing the code identified bottlenecks in the detection algorithm and allowed the student to take steps to remove them. As described in the methods chapter, executing code on the GPU requires a section of parallelized code for the GPU written using the CUDA API and uploaded to the GPU by a calling program (NVIDIA, 2006). The uploaded section of code is known as a kernel and is executed by several threads simultaneously on the GPU on separate data, a textbook example of data level parallelism. Following execution on the GPU, the results of the kernel execution are downloaded from the GPU and returned to the calling program (NVIDIA, 2006).

The find_ellipse prototype consists of two major components, a driver program and the kernel. The driver program reads in the input file data generated from the uniprocessor implementation, prepares the graphics card for execution, and retrieves the execution results from the graphics card. Input data generated by the uniprocessor detection implementation includes two dimensional image gradient and angular data used

to calculate a gradient score for each pixel. This gradient score per pixel is used to generate the GICOV scores per pixel. After reading in this input data from the input file, the driver program prepares the GPU for execution. The driver program allocates memory on the GPU for each input and copies the inputs to the allocated memory. At this point the kernel is executed on the GPU. Within the kernel each loop iteration is assigned a separate thread in a classic demonstration of data parallelism; the find_ellipse code found in Appendix C illustrates the loop described above. The results of the kernel execution are downloaded by the driver program from the GPU to main memory. Finally, the kernel GICOV results are compared to the GICOV results of the uniprocessor implementation to determine if the kernel generates accurate GICOV scores.

CUDA is uniquely suited to the detection algorithm and similar algorithms containing sections of parallelizable code that can be offloaded onto a GPU and processed. The student designed both prototypes for find_ellipse and dilate_image with minimal difficulty; however, achieving results on the GPU that matched those generated by the base uniprocessor code proved extremely challenging due to unfamiliarity with the CUDA development environment and an unsuspected precision difference discussed below.

B. Precision Difference

While examining data sets produced by the base uniprocessor algorithm and the student-designed CUDA uniprocessor algorithm, a precision difference was discovered between the two data sets. During lengthy attempts by the student to generate data from find_ellipse within CUDA as a single threaded model to match data from the base uniprocessor code, the student constructed a comparison program that compared data sets

between the two programs and tallied the data points that did not match. A chance inspection of the data revealed that data from the two sources differed by values ranging from 0.001 to 0.000001; these differences are negligible when considering that the data typically ranged between 3.0 and 5.0 and can be discarded. When a factor accounting for small variation was incorporated into the comparison program the data sets were found to match up within 0.01%.

C. Timing Data

Timing data acquired through execution of the find_ellipse prototype did not supported the student's hypothesis of a three order of magnitude decrease in processing time and a corresponding increase in throughput. As mentioned in the introduction, preliminary timing results from the find_ellipse multithreaded prototype reflect an approximately 85 ms execution time. Previous execution of the find_ellipse function on the same platform on the uniprocessor resulted in an approximately 1.01 s or 1010 ms execution time. This reflected a two order of magnitude decrease in processing time.

VI. Interpretation of Results

This chapter discusses the significance of the results described in the previous chapter and their relevance to the research area in general. The results interpreted here include the precision difference and timing data discussed in the results chapter.

A. Precision Difference

The existence of varying results is mentioned by the CUDA programming guide. For this reason, comparison computations are written into sample CUDA programs to account for variations between uniprocessor and multithreaded computations. The detection algorithm provides a real-world example of computation that encounters this precision difference (NVIDIA, 2006). The variations experienced between uniprocessor and multithreaded versions are found to be within 0.01%. The variations do not in this instance affect the outcome of the detection algorithm but are notable for their possible affect in other highly precise applications. Programs that require extremely precise calculation to within 0.01% could generate erroneous output in the CUDA environment.

B. Timing Data

Comparing the uniprocessor and multithreaded execution times, it is clear that execution time was reduced by two orders of magnitude. The multithreaded timing data also reflects the time required to upload and download data to the GPU; these operations are the most time intensive required by GPU processing and contribute greatly to the execution time (NVIDIA, 2006). The speedup experienced by the multithreaded prototype does not reach the goal of a three order of magnitude reduction in processing time. However, the two order of magnitude decrease accordingly increases the amount of

data that can be processed over time. Techniques remain that can be used to attempt to generate further speedup in execution time, such as varying the number of execution threads and unrolling loops to generate additional parallelism.

VII. Conclusions

This chapter summarizes the design experience, precision difference, and timing data conclusions obtained to this point. The chapter also discusses remaining in the project and recommendations to for future research.

A. Summary

The design experience of the student, precision difference encountered, and resultant timing data define the results of the project. The design experience reveals the difficulties inherent in a multithreaded design project and the knowledge gained will assist the student in future endeavors. Precision differences measured between uniprocessor and multithreaded implementations of the algorithm provide a concrete real-world example of CUDA precision problems; precision applications requiring computation within 0.01% accuracy could encounter difficulties in the CUDA architecture. Finally, timing data measured supports expectations of application speedup greater than a single order of magnitude. Measured timing reveals a two order of magnitude decrease in processing time for this algorithm with potential for additional decreases.

B. Recommendations for Future Research

As of this writing, the project has returned encouraging results; the project is not yet complete and is expected to reinforce previously returned results along with additional valuable design insights upon completion. The student will complete the project by fully verifying data from the find_ellipse prototype and implementing and verifying the dilate_image prototype. The student will accomplish these objectives by the end of the

project timeline in June 2007. Due to the necessary scope reduction brought about by the student, further research is necessary into alternate parallel architectures to determine if another architecture can produce more promising results than the NVIDIA GPU. Also needed is the translation of the Matlab tracking code into C or C++ and the parallelization of the tracking algorithm once it is translated. Other possibilities for algorithm refinement exist that range from modifying how variance is calculated to the data structures utilized in the algorithms. The student encourages prospective researchers to consider multithreaded design projects, particularly on the NVIDIA GPU, given the emerging trend from uniprocessor computers to new multicore platforms. Ultimately, multithreaded design is the wave of the future and upcoming computer scientists must ride the wave and face the challenges in development accordingly to meet the computing needs of the future.

VIII. Bibliography

Association of Computing Machinery Council. (1992). ACM Code of Ethics and Professional Conduct. ACM Computing and Public Policy Website. Retrieved on May 8, 2007, from http://www.acm.org/constitution/code.html

Acton, S. T., Wethmar, K., & Ley, K. (2002). Automatic tracking of rolling leukocytes in vivo. *Microvascular Research, 63*, 139.

Andrews, G. R. (2000). *Foundations of multithreaded, parallel, and distributed programming*. Reading, MA: Addison-Wesley.

Atanasov, D. (2005). General purpose GPU programming. *Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing, 11,* 1.

ATI. (2006). *ATI CTM Guide Technical Reference Manual*. Markham, Ontario, Canada: ATI Technologies, Inc.

Cvijetic, M. *Leukocyte Detection in Matlab Source*. University of Virginia: Charlottesville, VA.

Dokken, T., Hagen, T. R., & Hjelmervik, J. M. (2005). The GPU as a high performance computational resource. *Proceedings of the 21st Spring Conference on Computer Graphics, 21,* 21.

Dubash, M. (2005, April). *Moore's law is dead, says Gordon Moore*. Retrieved September 24, 2006, from http://www.techworld.com/opsys/news/index.cfm?NewsID=3477

Fenlason, J., & Stallman, R. (1988). *GNU gprof.* Boston, MA: Free Software Foundation, Inc.

Grama, A., Gupta, A., Karypis, G., & Kumar, V. (2003). *Introduction to parallel computing* (2nd ed.). Essex, England: Benjamin/Cummings Publishing Company, Inc.

Gschwind, M. (2006). Chip multiprocessing and the cell broadband engine. *Proceedings of the 3rd Conference on Computing Frontiers, 3,* 1.

Howland, J. E. (2006). Parallelism across the curriculum. *Journal of Computing Sciences in Colleges, 21*(4), 134.

IBM. (2006). *Cell broadband engine programming handbook* (1st ed.). United States of America: IBM.

Johnson, G., Turnbull, D., & Fitzsimons, E. In vivo microscopy: Technologies and applications. (1999). Gaithersburg, MD: National Institute of Health.

Kunkel, Eric J., Dunne, Jessica L., Ley, Klaus. (2001). Leukocyte Arrest During Cytokine-Dependent Inflammation In Vivo. *The Journal of Immunology*. Bethesda, MD: The American Association of Immunologists.

Kurtz, B. L., Kim, C., & Alsabbagh, J. (1998). Parallel computing in the undergraduate curriculum. *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education* (pp. 21-216). Atlanta, GA: ACM Press.

Lach, J., Acton, S. T., & Skadron, K. (2006). *SEI: Hierarchical Dependency Graphs for Col-Space Design with Application to Leukocyte Detection and Tracking*. Charlottesville, VA: University of Virginia.

Ley, K. (2001). Leukocyte recruitment as seen by intravital microscopy. In K. Ley (Ed.), *Physiology of inflammation* (pp. 303). New York: Oxford University Press.

Leyden, P. (1997, Moore's law repealed, sort of. *Wired, 1997*(5.05)

Metropolis, N., & Rota, G. (1993). *A new era in computation*. Cambridge and London: MIT Press.

NVIDIA. (2006). *NVIDIA CUDA Programming Guide*. Santa Clara, CA: NVIDIA Corporation.

Ostreich, T. *avilib.c*. Free Software Foundation: Cambridge, MA.

Pancake, C. M. (1991). Where are we headed? *Communications of the ACM, 34*(11), 52.

Ray, N., Acton, S. T., & Ley, K. (2002). Tracking leukocytes in vivo with shape and size constrained active contours. *IEEE Transactions on Medical Imaging, 21*(10), 1222.

Regan, P. (2006). Preserving privacy. *Issues in Science and Technology*. Dallas: University of Texas at Dallas.

Segal, M., & Peercy, M. (2006). *A performance-oriented data parallel virtual machine for GPUs*. Markham, Ontario, Canada: ATI Technologies, Inc.

Stewart, D. E., & Leyk, Z. *Meschach Library*. School of Mathematical Sciences, Australian National University: Canberra, Australia.

Strassberg, D. (1996). Data security: Key issue in an age of pervasive computing. *EDN, 41*(8), 48-55.

Talia, D. (1997). Parallel computation still not ready for the mainstream. *Communications of the ACM, 40*(7), 98.

Twist, J. (2005, April 18, 2005). Law that has driven digital life. *BBC News*. Retrieved September 24, 2006, from http://news.bbc.co.uk/1/hi/sci/tech/4449711.stm

University of Virginia Health System. (2003). Ethical Considerations in the Use of Laboratory Animals for Research and Testing at the University of Virginia. *University of Virginia Health System Website*. Retrieved May 8, 2007, from http://www.healthsystem.virginia.edu/internet/ccm/ETHICS/ethics.cfm

Wolpert, L. *Leukocyte Detection in C Source*. University of Virginia: Charlottesville, VA.

Xavier, C., & Iyengar, S. S. (1998). Introduction to parallel algorithms. United States of America: John Wiley & Sons, Inc.

Appendix A – Cited Figures

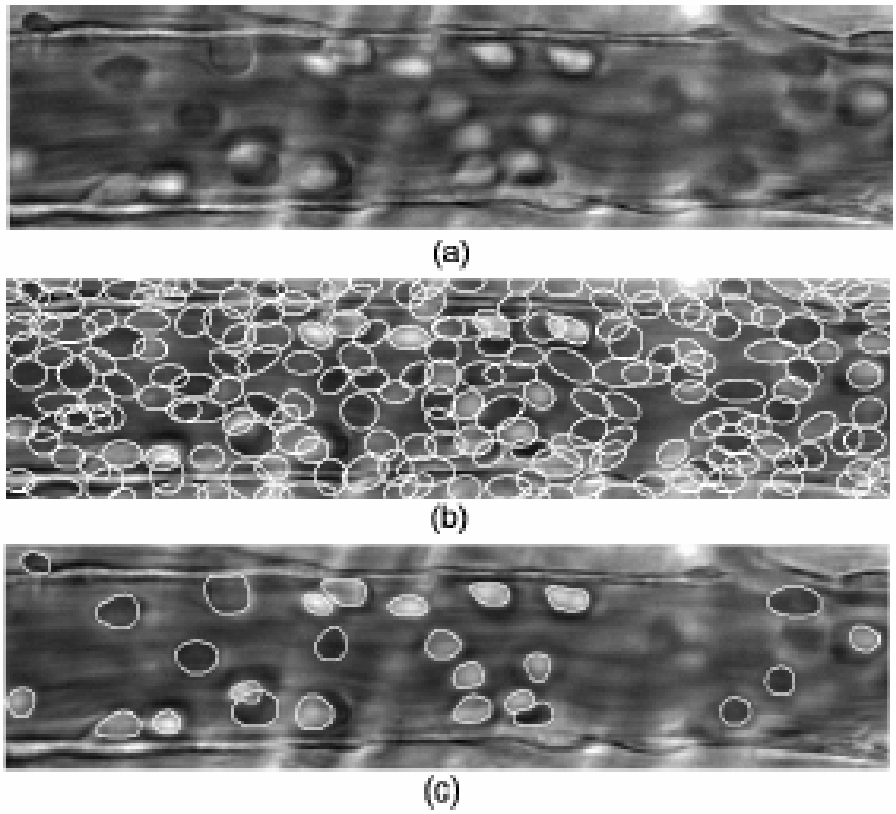

Figure 1. An example of leukocyte detection. (a) An example image in the video sequence showing leukocytes and stationary structures. (b) Result of ellipse matching (in white) on (a). (c) Final result of detection by GICOV thresholding after using B-spline contour. Modified from Lach, Acton, & Skadron, 2006.

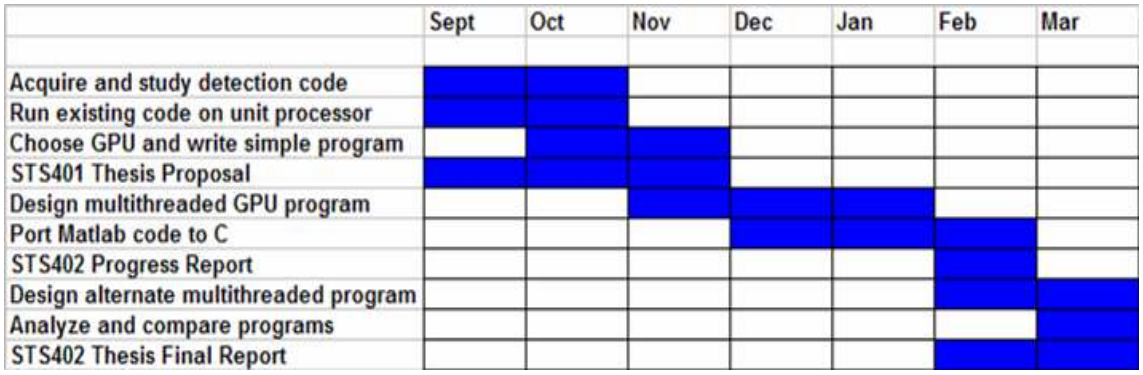

Figure 2. Original Gantt chart. This gantt chart specifies the expected timetable in which the student expected to complete project objectives. Created by student, 2006.

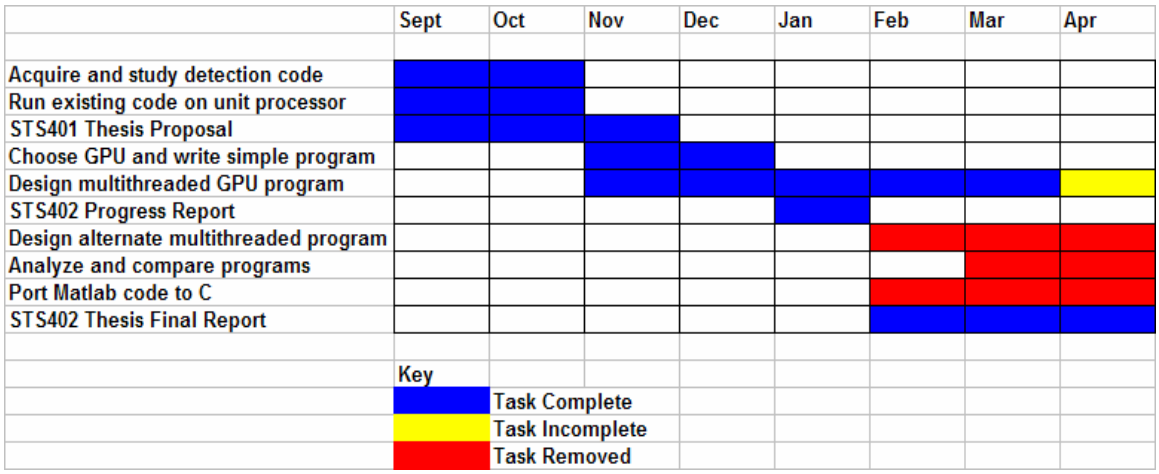

Figure 3. Revised Gantt chart. This gantt chart specifies the final timetable in which the student will complete project objectives. Created by student, 2007.

	Uniprocessor Code	find_ellipse prototype
Precision Difference	baseline	~0.01% difference
Timing Data	1010 ms	85 ms

Figure 4. Summarized Results. This table details the find_ellipse results as compared to the baseline uniprocessor results. Created by student, 2007.

Appendix B – Gprof Analysis of Detection Algorithm

The following are portions of the output provided by gprof analysis of the leukocyte detection C source code:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 57.21      3.65     3.65        1     3.65     3.65  ellipsematching
 40.75      6.25     2.60        1     2.60     2.60  dilate_f
  0.63      6.29     0.04                             internal_mcount
  0.31      6.31     0.02   244460     0.00     0.00  double_eq
  0.31      6.33     0.02      720     0.00     0.00
splineenergyform01
  0.16      6.34     0.01     1512     0.00     0.00  getfdriv
  0.16      6.35     0.01     1512     0.00     0.00  getsampling
  0.16      6.36     0.01        1     0.01     0.01  gradient_x
  0.16      6.37     0.01        1     0.01     0.01  gradient_y
  0.16      6.38     0.01        1     0.01     6.34  main
```

Definitions of column headings:

```
%          the percentage of the total running time of the
time        program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds    for by this function and those listed above it.

 self       the number of seconds accounted for by this
seconds     function alone.  This is the major sort for this
            listing.

calls       the number of times this function was invoked, if
            this function is profiled, else blank.

 self       the average number of milliseconds spent in this
ms/call     function per call, if this function is profiled,
           else blank.

 total      the average number of milliseconds spent in this
ms/call     function and its descendents per call, if this
           function is profiled, else blank.

name        the name of the function.  This is the minor sort
            for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.
```

Appendix C – Find_ellipse Function

```
//Scan from left to right, top to bottom, getting GICOV values
for(i = MaxR; i < width-MaxR; i++)
{
        for(j = MaxR; j < height - MaxR; j++)
        {
                sGicov = 0;

                for(k = 0; k < ncircle; k++)
                {
                        for(n = 0; n< npoints; n++)
                        {
                                y = j + tY[k][n];
                                x = i + tX[k][n];

                                Grad[n] = m_get_val(grad_x, y, x) *
cos_angle[n] + m_get_val(grad_y, y, x) * sin_angle[n];
                        }
                        sum = 0.0;
                        ep = 0.0;

                        for(iIndex = 0; iIndex < npoints; iIndex++)
        sum+=Grad[iIndex];

                        ave = sum/(double)npoints;
                        var = 0.0;

                        for(iIndex = 0; iIndex < npoints; iIndex++)
                        {
                                sum = Grad[iIndex] - ave;
                                var += sum*sum;
                                ep+=sum;
                        }

                        var = (var - ep*ep/(double)npoints) /
(double)(npoints-1);

                        if(ave*ave/var > sGicov)
                        {
                                m_set_val(gicov, j, i, ave/sqrt(var));
                                sGicov = ave*ave/var;
                        }
                }
        }
}
```

Appendix D – Dilate_image Function

```
//Perform grayscale dilation on img_in using the provided sturcturing
element
MAT * dilate_f(MAT * img_in, MAT * strel)
{
     int i, j, el_i, el_j, x, y, el_center_i = strel->m/2;
     int el_center_j = strel->n/2;
     double max, temp;
     MAT * dilated = m_get(img_in->m, img_in->n);

     for(i = 0; i < img_in->m; i++)
     {
        for(j = 0; j < img_in->n; j++)
        {
                max = 0.0;
                for(el_i = 0; el_i < strel->m; el_i++)
                {
                        for(el_j = 0; el_j < strel->n; el_j++)
                        {
                                y = i - el_center_i + el_i;
                                x = j - el_center_j + el_j;
                                if(y >=0 && x >= 0 && y < img_in->m && x
< img_in->n && m_get_val(strel, el_i, el_j)!=0)
                                {
                                        temp = m_get_val(img_in, y, x);
                                        if(temp > max)  max = temp;
                                }
                        }
                }
                m_set_val(dilated, i, j, max);
        }
     }

     return dilated;
}
```