INTERNATIONAL JOURNAL OF ENGINEERING AND MANAGEMENT SCIENCES

www.scienceandnature.org

PARALLELIZATION OF DATA INTENSIVE CODE USING COMPUTER UNIFIED DEVICE ARCHITECTURE (CUDA)

[1] Bhardwaj Aditi, [2]Bhardwaj Rohatash. K. [2]Shishir .K. Gangwar
[1]College of Technology, Govind Ballabh Pant University of Agriculture & Technology, Pantnagar
[2]College of Agriculture,Mekelle University , Mekelle Ethiopia

ABSTRACT
Parallel processing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently. Parallelism has been employed for many years, mainly in high-performance computing. As power consumption by Computer has become a concern in recent years, parallel computing has become the dominant paradigm in Computer structural design, mainly in the form of multi-core processors. Parallel Computer programs are more difficult to write than sequential ones. CUDA (an acronym for COMPUTER Unified Device Architecture) is a parallel computing architecture developed by NVIDIA. CUDA gives developers access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs. Using CUDA, the latest NVIDIA GPUs become accessible for computation like CPUs.GPU computing or GPGPU is the use of a GPU (graphics processing unit) to do general purpose scientific and engineering computing. RNA is made up of a long chain of components called nucleotides. Each nucleotide consists of a a nitrogenous base, a ribose sugar, and a phosphate group. The sequence of nucleotides allows RNA to encode genetic information. The chemical structure of RNA is very similar to that of DNA, with two differences **(a)** RNA contains the sugar ribose while DNA contains the slightly different sugar deoxyribose **(b)** RNA has the uracil while DNA contains thymine. In this an attempt is being made to COMPUTER RNA Secondary structure algorithm in parallel using CUDA architecture, COMPUTER the time of execution on CPU and GPU. And compare the different RNA sequences for further scientific advancements.

KEYWORDS: Computer, Parallel Programming and Processing, CUDA, RNA, Graphics processors, GPU.

INTRODUCTION

The recent switch to parallel microprocessors is a milestone in the history of computing. Industry has laid out a roadmap for multi-core designs that preserves the programming paradigm of the past via binary compatibility and cache coherence. Conventional wisdom is now to double the number of cores on a chip with each silicon generation. A multidisciplinary group of Berkeley researchers met nearly two years to discuss this change. Our view is that this evolutionary approach to parallel hardware and software may work from 2 or 8 processor systems, but is likely to face diminishing returns as 16 and 32 processor systems are realized, just as returns fell with greater instruction-level parallelism.

Parallel Processing

Parallel processing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel"). There are several different forms of parallel computing: bit-level, instruction level, data, and task parallelism. Parallelism has been employed for many years, mainly in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling. As power consumption (and consequently heat generation) by Computer has become a concern in recent years, parallel computing has become the dominant paradigm in Computer architecture, mainly in the form of multi-core processors.

Classification: Parallel Computer can be roughly classified according to the level at which the hardware supports parallelism with multi-core and multi-processor Computer having multiple processing elements within a single machine, while clusters, MPPs, and grids use multiple Computer to work on the same task. Specialized parallel Computer architectures are sometimes used alongside traditional processors, for accelerating specific tasks.

Challenges: Parallel Computer programs are more difficult to write than sequential ones, because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common. Communication and synchronization between the different subtasks are typically one of the greatest obstacles to getting good parallel program performance.

CUDA (Computer Unified Device Architecture)

CUDA (Computer Unified Device Architecture) is a parallel computing architecture developed by NVIDIA. CUDA is the computing engine in NVIDIA graphics processing units (GPUs) that is accessible to software developers through

variants of industry standard programming languages. Programmers use 'C for CUDA' (C with NVIDIA extensions and certain restrictions), compiled through a Path Scale C compiler, to code algorithms for execution on the GPU.

Architecture: CUDA provides both a low level API and a higher level API. The initial CUDA SDK was made public on 15 February 2007, for Microsoft Windows and Linux. Mac OS X support was later added in version 2.0, which supersedes the beta released February 14, 2008. CUDA works with all NVIDIA GPUs from the G8X series onwards, including GeForce, Quadro and the Tesla line. NVIDIA states that programs developed for the GeForce8 series will also work without modification on all future NVIDIA video cards, due to binary compatibility.

Accessibility: CUDA gives developers access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs. Using CUDA, the latest NVIDIA GPUs become accessible for computation like CPUs. Unlike CPUs however, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very quickly. This approach of solving general purpose problems on GPUs is known as GPGPU.

Applications: In the Computer game industry, in addition to graphics rendering, GPUs are used in game physics calculations (physical effects like debris, smoke, fire, fluids); examples include PhysX and Bullet. CUDA has also been used to accelerate non-graphical applications in computational biology, cryptography and other fields by an order of magnitude or more. An example of this is the BOINC distributed computing client.

GPU (Graphics Processing Unit): GPU computing or GPGPU is the use of a GPU (graphics processing unit) to do general purpose scientific and engineering computing.

Evolution: The GPU has evolved over the years to have teraflops of floating point performance. NVIDIA revolutionized the GPGPU and accelerated computing world in 2006-2007 by introducing its new massively parallel architecture called "CUDA". The CUDA Architecture consists of 100s of processor cores that operate together to crunch through the data set in the application**.**

Model: The model for GPU computing is to use a CPU and GPU together in a heterogeneous co-processing computing model. The sequential part of the application runs on the CPU and the computationally-intensive part is accelerated by the GPU. From the user's perspective, the application just runs faster because it is using the high-performance of the GPU to boost performance. GPGPU is a general purpose graphic processing unit and operates in parallel to CPU in order to simplify the execution of the code.

Application: The success of GPGPUs in the past few years has been the ease of programming of the associated CUDA parallel programming model. In this programming model, the application developers modify their application to take the COMPUTER-intensive kernels and map them to the GPU. The rest of the application remains on the CPU. Mapping a function to the GPU involves rewriting the function to expose the parallelism in the function and adding "C" keywords to move data to and from the GPU. The developer is tasked with launching 10s of 1000s of threads simultaneously. The GPU hardware manages the threads and does thread scheduling.

RNA STRUCTURE: Like DNA, RNA is made up of a long chain of components called nucleotides. Each nucleotide consists of a nucleobase (sometimes called a nitrogenous base), a ribose sugar, and a phosphate group. The sequence of nucleotides allows RNA to encode genetic information. For example, Some viruses use RNA instead of DNA as their genetic material, and all organisms use messenger RNA (mRNA) to carry the genetic information that directs the synthesis of proteins.

Like proteins, some RNA molecules play an active role in cells by catalyzing biological reactions, controlling gene expression, or sensing and communicating responses to cellular signals. One of these active processes is protein synthesis, a universal function whereby mRNA molecules direct the assembly of proteins on ribosomes. This process uses transfer RNA (tRNA) molecules to deliver amino acids to the ribosome, where ribosomal RNA (rRNA) links amino acids together to form proteins. The chemical structure of RNA is very similar to that of DNA, with two differences **(a)** RNA contains the sugar ribose while DNA contains the slightly different sugar deoxyribose (a type of ribose that lacks one oxygen atom), and **(b)** RNA has the nucleobase uracil while DNA contains thymine (uracil and thymine have similar base-pairing properties).

Background Study

Parallel Computing: The interest in parallel computing dates back to the late 1950's, with the advancements surfacing in the form of superComputer throughout the 60's and 70's. These were shared memory multiprocessors, with multiple processors working side-by-side on shared data. In the mid 1980's a new kind of parallel computing was launched when the Caltech concurrent computation project built a superComputer for scientific applications from 64 Intel 8086/8087 processors. This system showed that extreme performance could be achieved with mass market, off the shelf multiprocessors. These massively parallel processors (MPP) came to dominate the top end of computing, with the ASCI Red superComputer in 1997 breaking the barrier of one trillion floating point operations per second. Since then MPPs have continued to grow in size and power. Starting in the late 80's, clusters came to complete and eventually displace MPPs for many applications. A cluster is a type of parallel Computer built from large numbers of off the shelf Computer connected by an off the shelf network. Today, Clusters are the workhorse of scientific computing and are the dominant architecture in the data centers that power the modern information age. Today, parallel computing is becoming mainstream based on multi-core processors. Most desktop and laptop systems now ship with dual-core multiprocessors, with quad-core processors readily available. Chip manufacturers have begun to increase overall processing performance by adding additional CPU cores. The reason is that increasing performance through parallel processing can be far more energy-efficient than increasing multiprocessor clock

frequencies. In a world which is increasingly mobile and energy conscious, this has become essential. Fortunately, the continued transistor scaling predicted by Moore's will allow for a transition from a few cores to many. The software world has been very active part of the evolution of parallel computing. Parallel programs have been harder to write than sequential ones. A program that is divided into multiple concurrent tasks is more difficult to write, due to the necessary synchronization and communication that needs to take place between those tasks. Some standards have emerged. For MPPs and clusters, a number of application programming interfaces converged to a single standard called MPI by the mid 1990's. For shared memory multiprocessor computing, a similar process unfolded with convergence around two standards by the mid to late 1990's: Pthreads and OpenMP. In addition to these a multitude of competing parallel programming models and languages have emerged over the years. Some of these models and languages may provide a better solution to the parallel programming problem than the above "standards", all of which is modifications to conventional, non-parallel languages like C. As multi-core processors bring parallel computing to mainstream customers, the key challenge in computing today is to transition the software industry to parallel programming. The long history of parallel software has not revealed any "silver bullets", and indicates that there will not likely be any single technology that will make parallel software ubiquitous.

CUDA Computing: CUDA is NVIDIA's parallel computing architecture. It enables dramatic increases in computing performance by harnessing the power of the GPU. With millions of CUDA-enabled GPUs sold to date, software developers, scientists and researchers are finding broad-ranging uses for CUDA, including image and video processing, computational biology and chemistry, fluid dynamics simulation, CT image reconstruction, seismic analysis, ray tracing, and much more computing is evolving from "central processing" on the CPU to "co-processing" on the CPU and GPU. To enable this new computing paradigm, NVIDIA invented the CUDA parallel computing architecture that is now shipping in GeForce, Quadro, and Tesla GPUs, representing a significant installed base for application developers. In the consumer market, nearly every major consumer video application has been, or will soon be, accelerated by CUDA, including products from Elemental Technologies, MotionDSP and LoiLo, Inc.CUDA has been enthusiastically received in the area of scientific research. For example, CUDA now accelerates AMBER, a molecular dynamics simulation program used by more than 60,000 researchers in academia and pharmaceutical companies worldwide to accelerate new drug discovery. In the financial market, Numerix and CompatibL announced CUDA support for a new counterparty risk application and achieved an 18X speedup. Numerix is used by nearly 400 financial institutions.

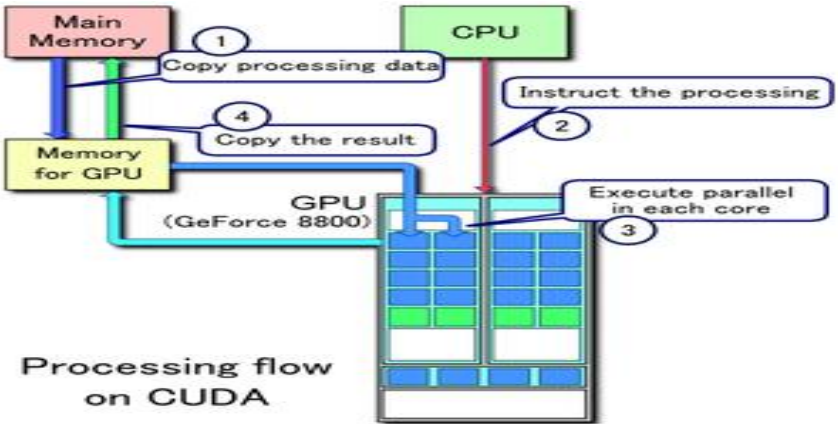

Figure 1: Processing Flow in CUDA Architecture

GPU Computing: Graphics chips started as fixed function graphics pipelines. Over the years, these graphics chips became increasingly programmable, which led NVIDIA to introduce the first GPU or Graphics Processing Unit. In the 1999-2000 timeframe, Computer scientists in particular, along with researchers in fields such as medical imaging and electromagnetic started using GPUs for running general purpose computational applications. They found the excellent floating point performance in GPUs led to a huge performance boost for a range of scientific applications. This was the advent of the movement called ***GPGPU*** or General Purpose computing on GPUs. The sequential part of the application runs on the CPU and the computationally-intensive part is accelerated by the GPU.

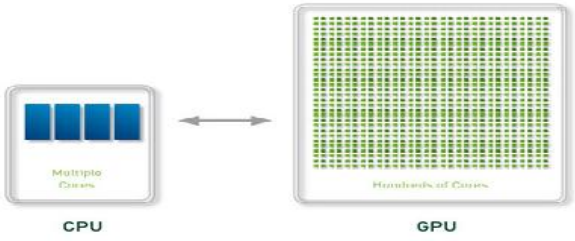

Figure 2: Multi-core and Many-core processors

The problem was that GPGPU required using graphics programming languages like OpenGL and Cg to program the GPU. Developers had to make their scientific applications look like graphics applications and map them into problems that drew triangles and polygons. This limited the accessibility of tremendous performance of GPUs of science. NVIDIA realized the potential to bring this performance to the larger scientific community and decided to invest in modifying the GPU to make it fully programmable for scientific applications and added support for high-level languages like C, C++, and FORTRAN. This led to the **CUDA architecture** for the GPU.

RNA Structure: RNA is made up of a long chain of components called nucleotides. Each nucleotide consists of a nucleobase (sometimes called a nitrogenous base), a ribose sugar, and a phosphate group. Each nucleotide in RNA contains a ribose sugar, with carbons numbered 1' through 5'. A base is attached to the 1' position, in general, adenine (A), cytosine (C), guanine (G), or uracil (U). Adenine and guanine are purines, cytosine, and uracil are pyrimidines. A phosphate group is attached to the 3' position of one ribose and the 5' position of the next. The phosphate groups have a negative charge each at physiological pH, making RNA a charged molecule (polyanion). The bases may form hydrogen bonds between cytosine and guanine, between adenine and uracil and between guanine and uracil. However, other interactions are possible, such as a group of adenine bases binding to each other in a bulge, or the GNRA tetraloop that has a guanine–adenine base-pair.

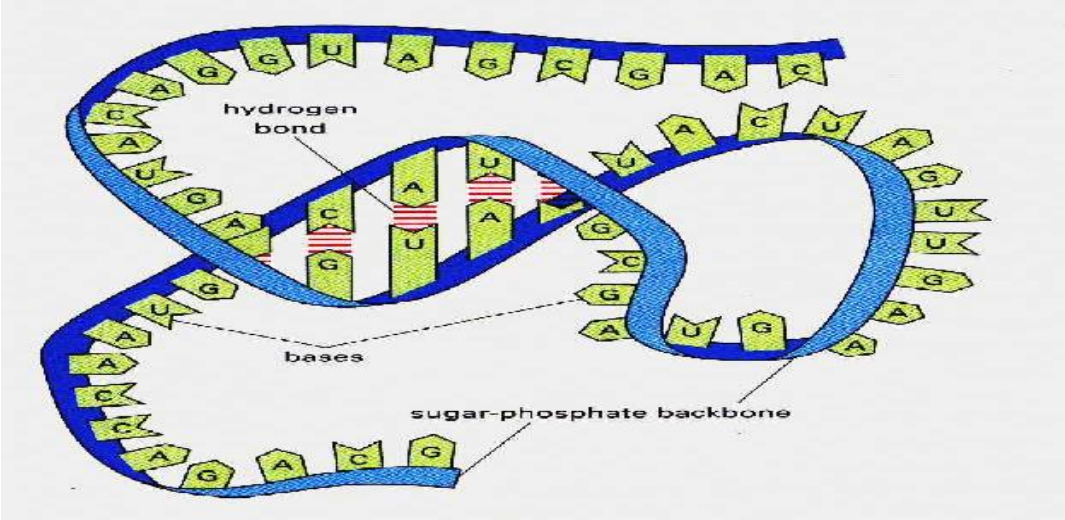

Figure 3 : RNA Helix Structure

The functional form of single stranded RNA molecules, just like proteins, frequently requires a specific tertiary structure. The scaffold for this structure is provided by secondary structural elements that are hydrogen bonds within the molecule. This leads to several recognizable "domains" of secondary structure like hairpin loops, bulges, and internal loops. Since RNA is charged, metal ions such as Mg^{2+} are needed to stabilize many secondary and tertiary structures.

PROBLEM SPECIFICATION

Problem Formulation

- *The first sub-problem is to configure the system in order to provide an environment for parallel processing.*
- *The second sub-problem is understanding and implementation of RNA secondary structure generation using Nussinov-Jacobson Algorithm in C.*
- *The third sub-problem is to do Profiling of the existing algorithm to find the most COMPUTER intensive block in the algorithm.*
- *The fourth sub-problem is the Parallelization of Algorithm using CUDA.*
- *The fifth and the final sub-problem is to check out the performance of the parallelized algorithm.*

Phase 1: In this phase firstly, CUDA Wizard needed to be installed along with the CUDA toolkit version 3.0 which supports EMU debug mode. EMU debug mode is required for running a parallel program on a system without any GPU present on it. Secondly, Visual studio 2008 is needed to be installed for integrating the functionalities of NVIDIA and C compiler. Lastly, GPU computing sdk has to be installed for providing the library and include files for NVIDIA and CUDA example codes.

Phase 2: Under this phase RNA structure is being needed to be understand.RNA molecules often fold back on themselves, forming stable double-helix structures akin to the famous DNA double helix, with G-C and A-U pairs forming, so called Watson-Crick base pairs.

GCCCACCUUCGAAAAGACUGGAUGACCAUGGGC CAUGAUU

(((((.(((....)))..(((....)))))))).(....)

Nossinov-Jacobsen Algorithm (for Watson-Crick base pairs)

The Nussinov-Jacobsen algorithm, based on the following recurrence, COMPUTERs, for each $1 < i < j < n$ the quantity B[i, j] which is the maximum number of pairs in any folding of the substring $x_i x_{i+1} \ldots x_j$ of the input:

B[i,j]=0 ,if i ^{3} j-4
max [B[i+1, j-1] + $p_{i,j}$, *max* { B[i,k] + B[k+1, j] : i £k < j }] , if i < j − 4,
Where $p_{i,j}$ = 1, if x_i could pair with x_j (i.e., G-C or A-U pairs), and $p_{i,j}$ = 0 otherwise.
B[i, j] is most easily represented as an n by n array, of which the lower-left triangle will be all zero. Since B[i, j] depends on entries to its left in the same row and entries below it in the same column, the convenient order in which to COMPUTER entries is to fill columns in left to right order, each column filled from the diagonal upwards. B[1, n] will hold the maximum number of pairs.

IMPLEMENTATION IN C

The algorithm is implemented by Dynamic Programming in C. The program takes a RNA sequence from a standard input prompt.

NOTE: The program doesn't have any input validation except the maximum length of a RNA sequence, which is 2000. So, user should enter a valid RNA sequence which consists of G, C, A,

Phase 3: Firstly we divide data intensive code into small modules. Then on the basis of the time taken by each module to execute, the priority is being given to most time taken module over others. This process is called Profiling. Profiling is a set of techniques for estimating the amount of time spent in various portions of the program. The size of the program unit being profiled is (sometimes) called the granularity. The profile will tell you how much of the execution time can be attributed to each grain that is being selected. The output of the profiler indicates what fraction of the execution time was spent in each grain. Presumably, to improve the performance the main focus is on the grains that are taking the most time.

Phase 4: Traditionally, CPU has been optimized to maximize the performance of a single-thread execution, and GPU to achieve a high throughput for a small number of fixed graphics operations. GPUs were generally good at processing a program with rich data parallelism, while CPUs were good at handling a program with irregular parallelism. However, recent trends show convergence of both Architectures more energy efficient throughput computing on CPUs and better programmability on GPUs. These days, such GPUs are called general-purpose GPU (GPU) because they can be used for non-graphics processing. Still, the programmability is one of the biggest challenges in GPGPU computing, CUDA is an extension to C language to ease parallel programming on NVIDIA'S GPGPUs.

Phase 5: First the code is being executed on the C compiler CPU. Next it is being executed on NVIDIA C compiler. Then the difference of the time taken by both compilers is taken. The output of this difference gives us the performance increase in the algorithm execution.

METHODOLOGY USED

Methodology for Various Phases

Phase 1: In this phase first the selected RNA secondary structure algorithm is executed in parallel on the CPU. And the time of execution is being calculated. Then the same algorithm is now been executed on GPU. And its time of execution is also been calculated. The difference between the time of execution of these two is calculated which give the measure of performance increase in execution of this algorithm.

Phase 2: In this phase to further increase the performance of the execution of the algorithm some modification in the code is done. These modifications done in while running the code in NVIDIA compiler increase the performance of the execution up to a great extent.

Phase 3: This phase is testing phase. In this testing is being applied on different modules to check for errors if any and every possible effort is being made to make the code more precise and efficient.

Approach to Solve Problem

The development strategy that encompasses the process, methods and tools and the generic phases is called Software Engineer Paradigm. The software paradigm for software is chosen based on the nature of the project and application, the method and tools to be used.

Software development has been characterized as the problem solving loops in which four distinct stages are encountered: Status quo, problem definition, technical development and solution integration.

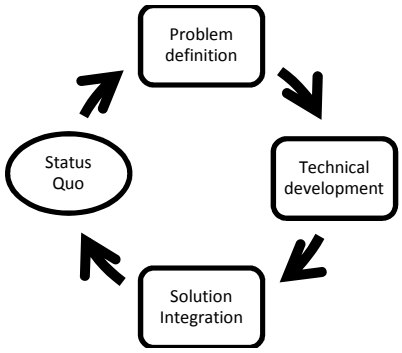

Figure 4: Problem Solving Loop

Status Quo: This represents the current state of affairs. In the present scenario, there are many available algorithms that need to be parallelized. CUDA architecture is one of it type which provide a platform for parallelization of the existing algorithm.

Problem Definition: This identifies the specific problem to be solved. As already defined, problem is divided into three phases: execution of code on CPU and GPU, modifications to further increase the performance and testing.

Technical Development: Under this, the problem is solved through the application of an appropriate technology. Here CUDA architecture is deployed to create the Environment used for execution of the algorithm.

Solution Integration: Through this phase the results are delivered to those who requested the solution in first place. This problem solving loop applies to software engineering work at many different levels of resolution. It can be used at the macro level when the entire application is considered, at a mid-level when program components are being engineered, and even at the line of code level.

Process Model Used: There are various software development models but Spiral Model has been used for development of this software. The spiral model, originally proposed by Boehm [BOE88], is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. It provides the potential for rapid development of incremental versions of the software. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced. A spiral model is divided into a number of framework activities, also called task regions. Typically, there are between three and six task regions as depicted in the figure. The methodology for our project consists of three of such regions.

Planning: In this region the planning was done for the next steps in the models by analyzing the results produced by the earlier iteration and the required result of the new iteration. In the first iteration the planning was done for implementing the algorithm into a sequential code. The second iteration needed the planning to be done for the parallelization of the code developed in the earlier iteration which was followed by the third iteration in which the parallelized code generated was applied to calculate the performance increase due to the parallelization and again in the fourth and the final iteration the parallelized code was implemented to calculate the match percentage of two different sequences of RNA being entered.

Risk analysis: One of the advantages of the spiral model is that it places the risk reduction mechanism in place. This task region is highly required for the comparison of two RNA sequences as it required by user for the testing of various vaccines to be used on human beings and if the project produces any vague results it can lead to the use of inappropriate vaccines for human beings and hence it is very risky and along with each iteration it is made to be sure that the result calculated at the end of each iteration is correct. In the first iteration the result calculated is the secondary structure of the RNA sequence being entered and it has to be correct and in order to make sure small sequences are entered as an input and the result is calculated manually and also by using the code implemented and these two results are then compared to check the righteousness of the code. The result generated at the end of the second iteration is compared with the result of first iteration to make sure that the sequence generated by the parallelized function is the same as the one generated by the sequential function. For the third iteration the risk is lower as the time computation does not affect the result for the matching of the two strings while for the fourth iteration the risk is very high as the sequences should be matched correctly and the result should be declared taking into account the risk factors involved.

Engineering: This task region includes the building of application for each iteration. For the first iteration the algorithm used is sequentially implemented using C. In the second iteration the changes are made to parallelize the COMPUTER intensive function of the sequential code from the earlier iteration. The third iteration requires the inclusion of time function using the clock class for the host code and cuda Events class for the device code for the calculation of both CPU and GPU timings. The fourth and final iteration requires adding the function for matching two sequences of RNA secondary structure generated and calculating the percentage of sequences being matched.

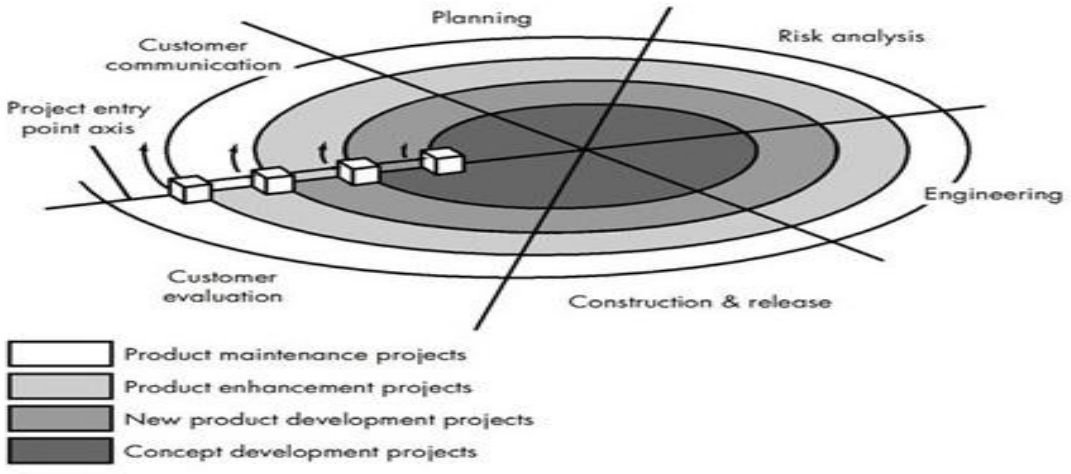

Figure 5. Spiral Model

REQUIRMENT ANALYSIS

This chapter describes the transformation of requirement specification to feasibility study. After collecting all the requirements of the user, feasibility of the software is determined. This phase ensures that all the requirements are mapped to the design phase.

Hardware requirements of the project

Following are the hardware requirements of the project to take input:

Hardware Needed:

1. HP Proliant Servers 2. HP SL390 G7 4U

Hardware Used:

1. Multiprocessor system.
2. Cache memory of up to 64 MB.
3. NVIDIA Graphic card.

Software Requirements of the project:

1. CUDA toolkit.
2. CUDA Wizard 32 bit.
3. GPU computing sdk.
4. Visual Studio 2008.

Feasibility Study: Feasibility study is the process of determination of whether or not a process is worth doing. Feasibility studies are under taken within tight time constraints and normally culminate in a written and oral feasibility report. It took 3 weeks to determine feasibility for this project. Feasibility study helped as a sound basis for deciding how to precede the project. It helped in taking decisions such as which software to use, hardware combinations, etc.

Technical Feasibility: Technical feasibility determines whether the work for the project can be done with the existing equipment, software technology and available personnel. This concerns specifying the equipment and software that will satisfy the user requirement.

ISSUES IN TECHNICAL FEASIBILITY

Practicality of the Proposed Technology or solution: The technologies used are mature enough so that they can be applied to our problems. The practicality of the solution we have developed is proved with the use of technologies we have chosen. The technologies such as parallel processing are still in the phase of development and are quite popular. With consistent efforts and study these techniques can be applied in practical terms also.

Availability of the necessary technology: It was first ensured that whether the required technologies are available for the project or not. There are many existing computational intensive algorithm that are needed to be parallelized in order to improve their performance and reduce their execution time. Only few of these algorithms have been parallelized till date.

Availability of the necessary Technical Expertise: This consideration of technical feasibility is often forgotten during feasibility analysis. The technology is available but at the same time skills have to be develop to properly apply that technology. CUDA Zone is a site available on internet by the NVIDIA in order to provide all the necessary expertise required by any CUDA user. There are also video lectures for CUDA created by the University of Illinois in collaboration with the NVIDIA Corporation which is helpful in developing the skills required for using CUDA architecture.

Economical Feasibility: Economic feasibility determines whether there are sufficient benefits in creating to make the cost acceptable, or is the cost of the system too high. This signifies cost-benefit analysis and savings. On the behalf of the cost-benefit analysis, the proposed system is feasible and is economical regarding its pre-assumed cost for making a system. During economical feasibility test, a balance between Operational economical feasibilities is maintained, as the two are conflicting.

For example the solution that provides the best operational impact for the end user may also be the most expensive and, therefore, the least economically feasible. For calculating the development cost, the certain cost categories were evaluated:

(a) Personnel cost
(b) Computer usage
(c) Software cost

Operational Feasibility: Operational feasibility criteria measure the urgency of the problem (survey and study phase) or the acceptability of a solution. There are various aspects of operational feasibility to be considered. These decide the effectiveness of the system. These measures can be collectively called ***PIECES.***

- **P (Performance):**
  The parallelization of the algorithm increases the performance.
- **I (Information):**
  This architecture provide end-users with timely, accurate and user formatted output.
- **E (Economy**):
  This architecture offer adequate service level and capacity to reduce the costs.
- **C (Control):**
  This software offer adequate controls to guarantee the accuracy while generating the RNA secondary structure.
- **S (Services):**
  This project provides desirable reliable service to those who need it. This software is flexible and expandable.

Design, Implementation and Testing

Design: Design is a meaningful engineering representation of software that is to be built. It can be traced to the requirements and at the same time assessed for quality against a set of predefined criteria for "good" design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations. Design patterns reside in the domain of modules and interconnections. At a higher level there are architectural patterns that are larger in scope, usually describing an overall pattern followed by an entire system. There are many types of design patterns like:

- ***Algorithm strategy patterns,*** *addressing concerns related to high-level strategies describing how to exploit application characteristic on a computing platform.*

- ***Computational design patterns,*** *addressing concerns related to key computation identification.*
- ***Execution patterns,*** *that address concerns related to supporting application execution, including strategies in executing streams of tasks and building blocks to support ta sk sy nchronization.*

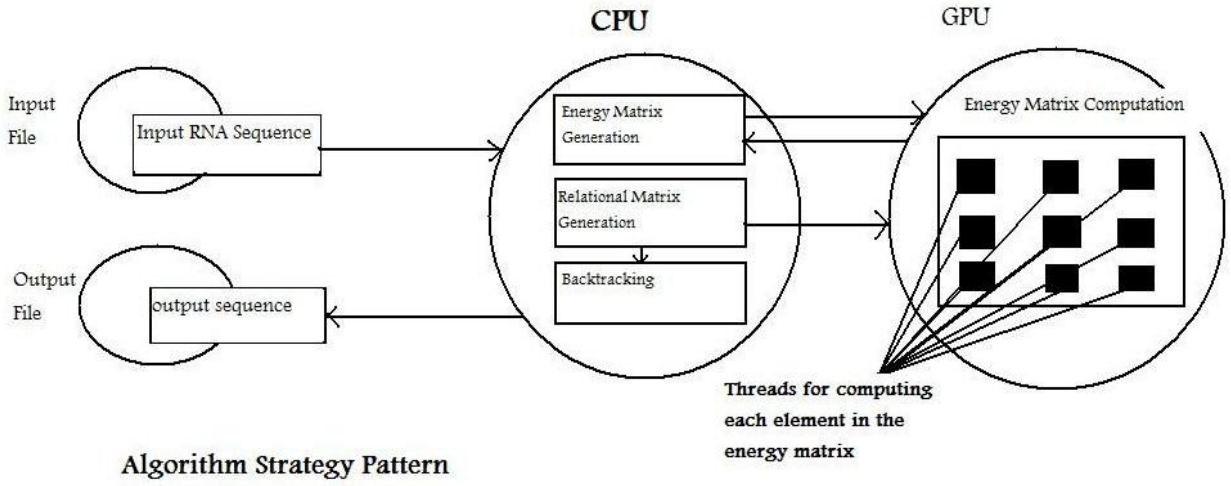

Figure 6: Algorithm Strategy Pattern

Implementation strategy patterns, addressing concerns related to implementing source code to support program organization, and the common data structures specific to parallel programming. In designing first of all an algorithm strategy is needed to be designed. For the Secondary Structure the algorithm strategy pattern is shown in the figure that follows. Generation of a RNA sequence using Nussinov-Jacobson Algorithm in a parallel computing environment. The input RNA sequence is taken from an input file and a corresponding initial energy matrix is generated, a relational matrix is also Computer which defines the pairing for the various sequence elements. Both the energy and relational matrices are copied onto the device memory for the energy matrix to be Computer parallelly on the GPU. The energy matrix thus Computer is used to form the secondary structure sequence using dot bracket notation which is written to the output file.

Computational Design Patterns: In the parallel programming panorama designing of the computational design patterns is of utmost importance before parallelization of the algorithm. This is done by profiling of the sequential implementation of the algorithm. In the RNA secondary structure generation using Nussinov-Jacobson Algorithm the most computation intensive function is found to be the Computer energy matrix function, the one used for the computation of the energy matrix which describes the pairing of the RNA sequence elements.

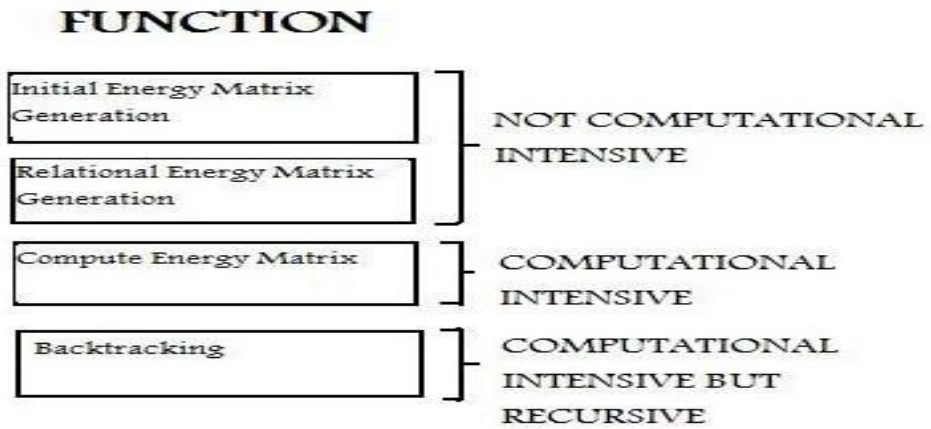

Figure 7: Computational Design Pattern

Execution Patterns

Dispatching GPU jobs by the host process is supported by the CUDA toolkit in the form of remote procedure calling. The GPU code is implemented as a collection of functions in a language that is essentially 'C', but with some annotations for distinguishing them from the host code. Source files for CUDA applications consist of a mixture of conventional C++ host code plus device functions. The CUDA compilation trajectory separates the device function from the host code, compiles the device function using proprietary NVIDIA compilers, compiles the host code using any general C/C++ compiler that is available on the host platform and afterwards embeds the compiled GPU functions as load images in the host object file. In the linking stage, specific CUDA runtime libraries are added for supporting remote SIMD procedure calling and for providing explicit GPU manipulation such as allocation of GPU memory buffers.

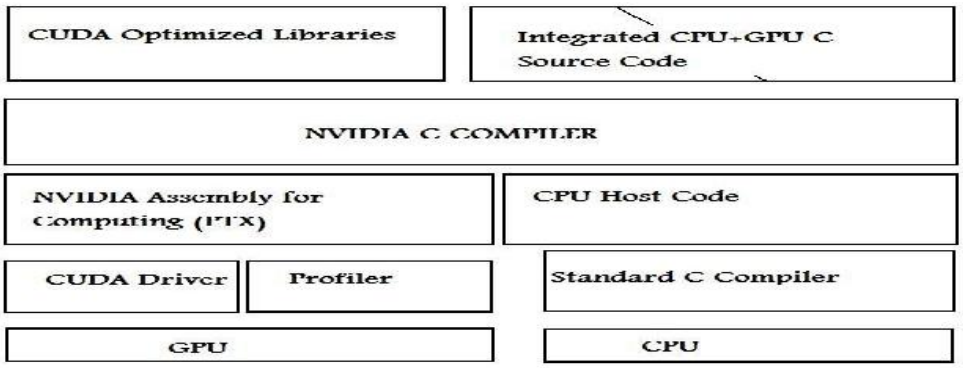

Figure 8: Execution Pattern

Implementation strategy patterns:

Implementation of an algorithm into a parallel program requires implementing the algorithm into sequential source code and then parallelizing the functions needed to be executed on the device. During host code implementation a RNA sequence is entered from an input file and the length n of the sequence is calculated. A matrix of n x n is Computer in which each i^{th}, j^{th} element describes the pairing of the i^{th} and j^{th} element of the RNA Sequence which is afterwards copied to the device memory for the computation of the energy matrix. During the implementation of device source code the data structures are required to be defined and initialized using cuda-malloc in the host source code and are copied to the device memory using cuda- memcpy function. The data structure used is a n x n matrix. When the device function is called the code is copied to the device memory and the code is executed on the GPU generating the resultant matrix which is copied back to the host using the cuda-memcpy function. The result is being displayed and stored in the result file and the memory allocations are freed using the cuda-free function.

Implementation: Implementation is transformation of design phase to coding phase. This chapter describes all the technologies used in development of the project. And it also shows the snapshots of database and frames built into the project.

Configuring the System: Before implementing the algorithm, a parallel programming environment has to be set-up using the following components.

Microsoft Visual Studio 2008: Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications. Visual Studio supports different programming languages by means of language services, which allow the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C/C++ (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), and F# (as of Visual Studio 2010). It also supports XML/XSLT, HTML/XHTML, JavaScript and CSS. Individual language-specific versions of Visual Studio also exist which provide more limited language services to the user: Microsoft Visual Basic, Visual J#, Visual C#, and Visual C++.

CUDA Toolkit: The CUDA Toolkit has all the development tools, libraries and documentations you need to create applications for the CUDA Architecture, including:

1) CUDA C/C++ Compiler
2) GPU debugging and profiling tools
3) GPU Accelerated math libraries
4) GPU Accelerated performance primitives

Hundreds of code samples and supporting documents are also available in GPU computing sdk.

Coding the Algorithm

After understanding the algorithm and designing the different modules we need to implement them in C using Visual Studio which provides an easy interface to implement C code and provides Graphical User Interface and integrate the functionalities of NVIDIA compiler and the standard C compiler. After implementing the code sequentially in C the most computational intensive parts of the code have to be parallelized using the CUDA functionalities. The following

figures show the code window of visual studio and the build and output windows of the program being implemented.

Implementation of Code:

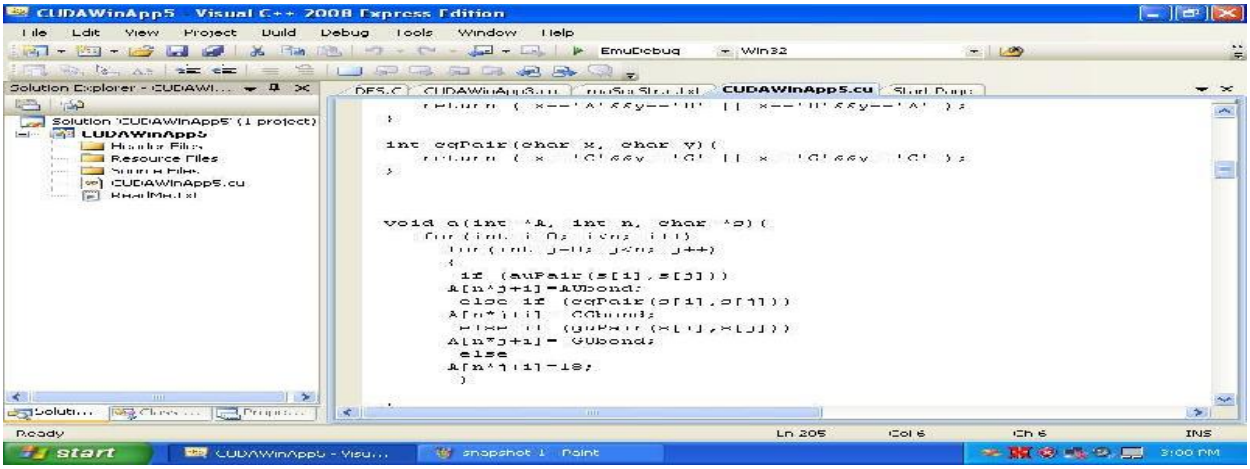

Figure 9: Visual Studio Source code window

Building of the Project:

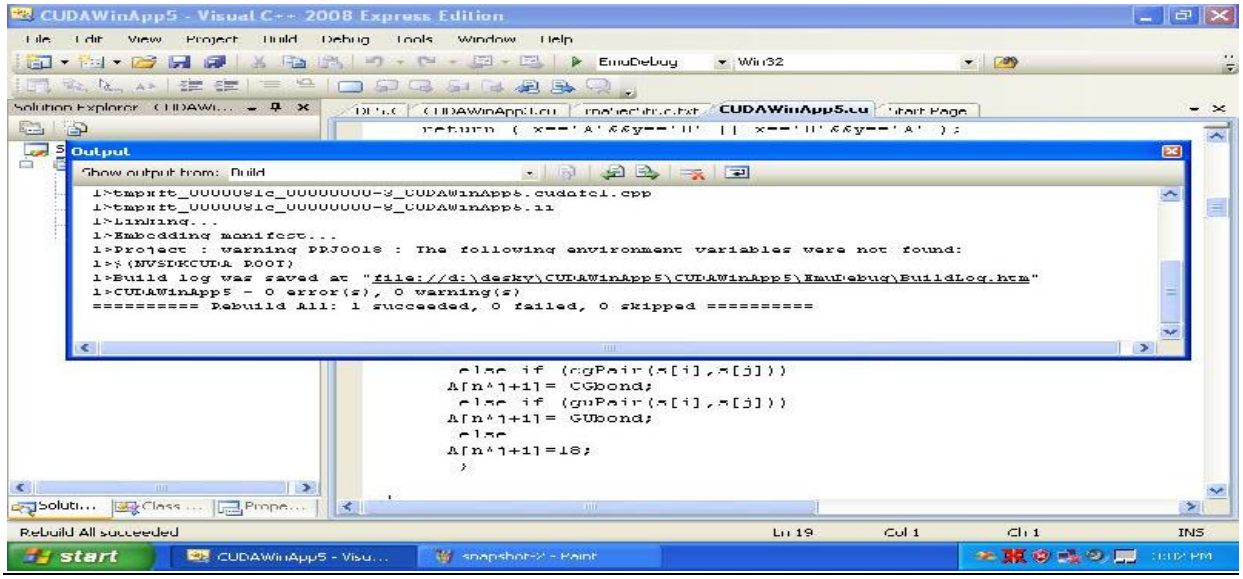

Figure 10: Building the window

Execution of Project:

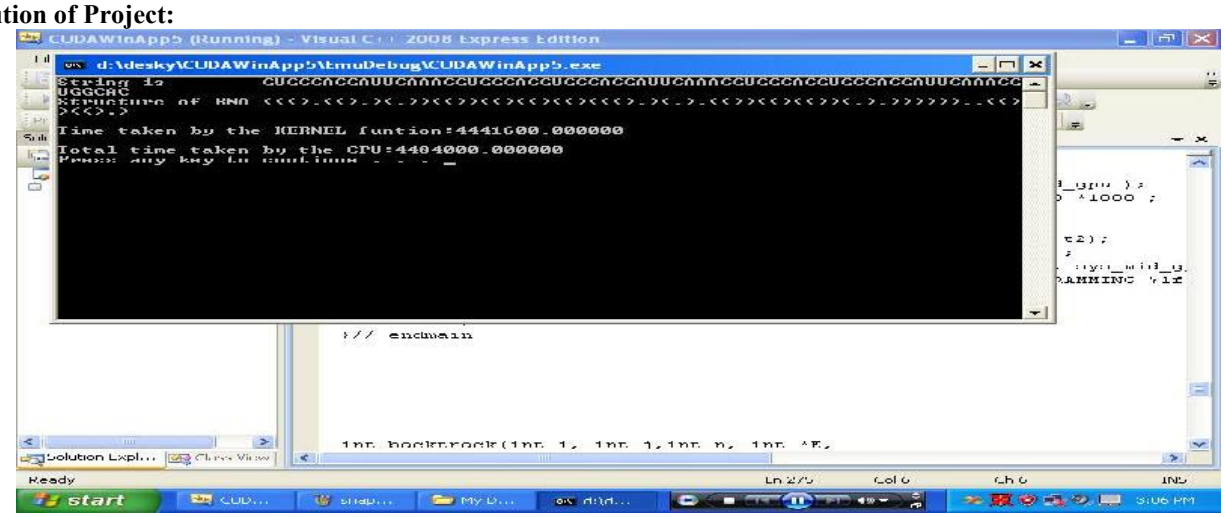

Figure 11: Output window

Testing: Testing is a set of activities that can be planned in advance and conducted systematically. In this software black-box and white-box testing is being used.

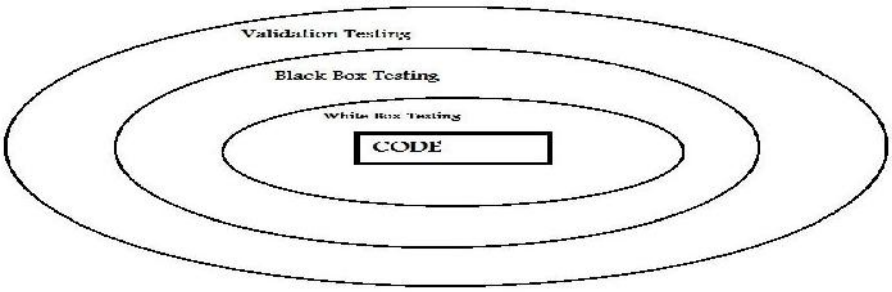

Figure 12: Testing

White-Box Testing: White-box testing is a method of testing software that tests internal structures or workings of an application. In white-box testing an internal perspective of the system, as well as programming skills, are required. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. In order to test the application the code implemented at each level of iteration is tested by displaying the computations performed within the structure of the code. The energy matrix being Computer for all the iterations of the loop are being displayed and the track of control and calculation is then checked using these matrices.

Black-Box Testing: Black-box testing is a method of software testing that tests the functionality of an application as opposed to its internal structures or workings. Specific knowledge of the application's code/internal structure and programming knowledge in general is not required.

Black-box testing attempts to find errors in the following categories:

(1) *Incorrect or missing functions: The correctness of the applied code is checked by initially entering some small RNA like sequences and by manually generating the result and comparing it with the result generated by the program.*

(2)*Interface errors: The output displayed is checked is it matches with the actually string generated and with the result string stored in the output file.*

(3)*Behavior or performance errors: Host execution time is measured using the clock function and the device execution time is measured using the CUDA class cuda Events. The timing is then used to calculate the percentage increase in the performance.*

RESULTS AND DISCUSSION

The main motive of paper is to reduce the time of execution of program thus increasing its performance.

Comparison on the basis of Time of execution

Human skin Cancer Papilloma virus

UGCAAGACACUUCUUUACAAGGAAUGGGGUAAUG
GGGGACACCAUACCAAAUGCAGUCAG
((()((.).(((((((()(..))(.)))((()(()(((..).)).).)))).(()))))(.)
Time at CPU= 3062000.000 microseconds
Time using GPU= 1528830.000 microseconds
Performance increased=50.07%

Rattus skin Cancer Papilloma virus type 5

AAAAUAAAGUGACUAAUUACUUGACCAGCUUGUC
CUCGCCUACUUCCUUUGCACCUGGGU

(((((()((((().)((()))(.(()).(()))))..(()...).)).))(()))(((())))
Time at CPU= 3766000.000 microseconds
Time using GPU= 1884569.625 microseconds
Performance increased=49.95%

From the above example we conclude that time of executing a code on CPU is considerably greater than that taken while executing with the help of GPU.

Thus, it can be clearly observed and concluded the performance of the execution of algorithm is increased almost by 50%.

MATCHING THE DIFFERENT RNA SEQUENCES

Human skin Cancer Papilloma virus

UGCAAGACACUUCUUUACAAGGAAUGGGGUAAUG
GGGGACACCAUACCAAAUGCAGUCAG

((())((.).(((((((()(..))(.)))((()(()(((..).)).).)))).(()))))(.)

Rattus skin Cancer Papilloma virus type 1

AUACCUUCGCCAUGUGGAGGAAUAUGAACUACAG
UUUGUGUUUCAACUUUGUAAAAUAAC

((()((((((()))))))(()(()(.(()..)))))(()((.((.))(()))))...
Percentage Match= 36.67%

Human skin Cancer Papilloma virus

UGCAAGACACUUCUUUACAAGGAAUGGGGUAAUG
GGGGACACCAUACCAAAUGCAGUCAG

((())((.).(((((((()(..))(.)))((()(()(((..).)).).)))).(()))))(.)

Rattus skin Cancer Papilloma virus type 2

UCCGAGCGAUUAUUAUUAUGCUCCGGACACGGCG
CAGGACCAAAAAACGGCAGGUCCUUC

((.))((()((()(()(()(()(((()))(..)(()))((.)))))(...(().(()))))))
Percentage Match= 51.67%

Human skin Cancer Papilloma virus

UGCAAGACACUUCUUUACAAGGAAUGGGGUAAUG
GGGGACACCAUACCAAAUGCAGUCAG

((())((.).(((((((()(..))(.)))(((()(()(((..).)).).)))))).((()))(.)

Rattus skin Cancer Papillomavirus type 3

UGUCUAUUUUGCUACCCCCAGUGGAUCCUUGGUG
UCCAGUGAUGGUCAGCUGUUCAACAG

((().(()(((()(().....(((()(()))(()))))...(()(()(()(()(()).)).)).))
Percentage Match= 38.34%

The above example shows the percentage match between the RNA sequence of Rattus norvegicus (Rat) and Homo sapiens (Humans).This matching can be very useful in the field of medical and scientific research. For example, if the percentage match is 80% or above the vaccines tested or experimented on rats could work for humans also.

CONCLUSION AND FUTURE SCOPE

The RNA secondary structure algorithm was selected and data intensive part of this algorithm was parallelized using CUDA architecture. Time taken of executing the code on CPU and GPU were successfully calculated and hence, the performance increased of execution of the code was measured.

On the other hand the two different RNA sequences were also compared. On the basis of the percentage matched it was concluded that if the percentage match is 80% or above, the vaccines used on the one species can also be used on the other.

Future scopes

Any attempt will need to be enough faster than CPU implementations to justify any sacrifices in quality. Further, the current CPU code is very well-written and highly optimized. At this point it seems that effort should be solely focused on the motion estimation segment of the encoding process. It represents the majority of the encoding time and is most suitable for parallelization. The other parts of the encoding process have complex control flows and many serial dependencies making them poor choice for CUDA.

We can begin transferring the content for the next frame to the graphics card while the current frame is encoding. There is also the question, which we did not answer, as to whether the down sampling should be performed on the GPU or on the CPU. While two sequential frames cannot be encoded in parallel due to the need to have the previously encoded frame available during parallel processing of the algorithm as a reference frame, frames reasonably far apart can be encoded in parallel. This would allow us to achieve higher occupancy on the GPU.

REFERENCES

Giorgos Vasiliadis, Spiros Antonatos, Michalis Polychronakis, Evangelos P. Markatos and Sotiris Ioannidis (September 2008, Boston, MA, USA). "Gnort: High Performance Network Intrusion Detection Using Graphics Processors" (PDF). Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID).

"Linear algebra operators for GPU implementation of numerical algorithms", Kruger and Westermann, International Conference on Computer Graphics and Interactive Techniques, 2005.

S Harding and W Banzhaf. "Fast genetic programming on GPUs". http://www.cs.bham.ac.uk/wbl/biblio/gp-html/eurogp07harding.html. Retrieved 2008-05-01.

W Langdon and W Banzhaf. "A SIMD interpreter for Genetic Programming on GPU GraphicsCards".http://www.cs.bham.ac.uk/wbl/biblio/gp-html/langdon_2008_eurogp.html. Retrieved 2008-05-01.

Almasi, G.S. and A. Gottlieb (1989). Highly Parallel Computing. Benjamin-Cummings publishers, Redwood City, CA.

Asanovic, Krste et al. (December 18, 2006). "The Landscape of Parallel Computing Research: A View from Berkeley" (PDF). University of California, Berkeley.

NVIDIA CUDA Software Development Kit (CUDA SDK) - Release Notes Version 3.0 for Window.

NVIDIA Clears Water Muddied by Larrabee Shane Mc Glaun (Blog) - August 5, 2008 – Daily Tech.

Higgs PG (2000). "RNA secondary structure: physical and computational aspects". Quarterly Reviews of Biophysics 33: 199–253.

RNA Folding with Nossinov-Jacobsen Algorithm, Hyung-Joon Kim, CSE 417, University of Washington.

Gnort: High Performance Network Intrusion Detection Using Graphics Processors" . Proceedings of the 11th International Symposium on RAID.

"CUDA Zone". www.nvidia.com/object/cuda_home.html.

www.wikipedia.com.

http://www.ics.forth.gr/dcs/Activities/papers/gnort.raid08.pdf.

http://setiathome.berkeley.edu/cuda.php.

http://setiathome.berkeley.edu/cudafaq.php.