HECTOR VINICIUS HIRATA

INVESTIGANDO O USO DE GPUS PARA APLICAÇÕES DE BIOINFORMÁTICA

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

São José do Rio Preto
2011

HECTOR VINICIUS HIRATA

INVESTIGANDO O USO DE GPUS PARA APLICAÇÕES DE BIOINFORMÁTICA

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

Orientador:
Prof. Dr. Aleardo Manacero Junior

São José do Rio Preto
2011

HECTOR VINICIUS HIRATA

INVESTIGANDO O USO DE GPUS PARA APLICAÇÕES DE BIOINFORMÁTICA

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

(assinatura)
Prof. Dr. Aleardo Manacero Junior (orientador)

(assinatura)
Hector Vinicius Hirata (aluno)

Banca Examinadora:

Prof. Dr. Renata Spolon Lobato

Prof. Dr. Masayoshi Tsuchida

São José do Rio Preto
2011

Hirata, Hector Vinicius.
Investigando o uso de GPUs em aplicações de Bioinformática / Hector Vinicius Hirata. - São José do Rio Preto : [s.n.], 2011.
40 f. : il. ; 30 cm.

Orientador: Aleardo Manacero Junior
Trabalho de conclusão (bacharelado - Ciência da Computação) - Universidade Estadual Paulista, Instituto de Biociências, Letras e Ciências Exatas

1. Computação. 2. Bioinformática. 3. Biologia molecular. 4 Processamento paralelo (Computadores) I. Aleardo Manacero Junior. II. Universidade Estadual Paulista, Instituto de Biociências, Letras e Ciências Exatas. III. Título.

CDU - 004:577.2

Ficha catalográfica elaborada pela Biblioteca do IBILCE
Campus de São José do Rio Preto - UNESP

Dedico esse trabalho a todas as pessoas que tenho estima e amor.

AGRADECIMENTOS

Agradeço aos professores do DCCE, professora Renata por ter sido uma amiga muito próxima durante essa fase difícil de minha vida. Ao professor Aleardo pela sua dedicação pelos seus alunos e especialmente a atenção que ele deu para realizar esse projeto final, e ao professor Masayoshi por ser um grande amigo durante toda a graduação. Também gostaria de agradecer ao professor Adriano e a professora Rogéria.

"Aint about how hard u hit its about how hard u can get hit and keep moving foward ...thats how winning is done"

Rocky Balboa – Rocky Balboa

RESUMO

O projeto a seguir tem como base, um estudo aprofundado sobre os algoritmos de alinhamento de sequências, como estes algoritmos são classificados e como eles são executados computacionalmente. Possui também um estudo mais detalhado sobre linguagens paralelas em GPUs assim como a implementação, os testes e a validação dos algoritmos de SMITH WATERMAN e BLAST, que são algoritmos de alinhamento locais, utilizando da linguagem CUDA e comparando o seu desempenho com o algoritmo de LEVENSHTEIN, algoritmo esse que faz alinhamento globalmente e pode ser implementado sequêncialmente.

ABSTRACT

The base of the following Project is a care studying about alignments algorithms, how they are classified and how they are executed in a computer. This work has a deep studying setting about parallel languages running in GPUs like they implementations, the final tests and the validation of Smith-Waterman and BLAST alignment algorithms using the CUDA language and comparing their time of execution with Levensthein algorithm that does the alignment globally and can be implemented sequentially.

ÍNDICE

Lista de Figuras

Lista de tabelas

Lista de abreviaturas e siglas

API : Application programming interface (interface de programação do aplicativo).

BLAST : *Basic Local Alignment Search Tool.*.

BLASTA : Ferramenta de alinhamento de sequência.

C : Linguagem de programação.

CAD : *Computer Aided Desing*.

ClustalW : Algoritmo de alinhamento.

CUDA : Linguagem de programação em GPU.

DNA : DeoxyriboNucleic Acid.

DCC : Outro programa de auxílio de desenho para computador.

FASTA : Algoritmo de alinhamento baseado no BLAST.

GPU : *Graphics Processing Unit.*

HSP : *High-score segment pair*.

INSDC : International Nucleotide Sequence Database Collaboration.

LTS : *Long Term Support*.

NCBI : National Center for Biotechnology Information.

NVIDIA® : Empresa responsável pela fabrição de grife famosa de GPUs.

MUMmer : Ferramenta para alinhamento de sequência.

MUSCLE : Outra ferramenta para o alinhamento de sequência.

RNA : Ribo Nucleic Acid

.

Capítulo 1

1.1 Introdução

Um dos mais recentes avanços da área de computação de alto desempenho é o uso de processadores gráficos (GPUs, ou Graphics Processing Units) como elementos de processamento paralelo [GOOD,2005]. Aplicações variadas como processamento de imagens, algoritmos genéticos, processamento de sinais, etc, têm sido desenvolvidas utilizando o poder computacional de GPUs [OHKS,2004],[TARA,2009],[ADAM,2007]. Em geral, aplicações que demandam paralelismo para viabilizar sua execução têm obtido uma sensível melhora com o uso de GPUs como elementos de processamento. O impacto da introdução de GPUs em computação de alto desempenho é ainda mais evidenciado pelo baixo custo associado a tais processadores quando comparados com processadores de uso geral. Em alguns casos isso ainda é facilitado pela possibilidade de se obter o paralelismo sem grandes modificações nas aplicações que já executavam em paralelo, desde que não seja necessário modificar grandemente o modelo de particionamento usado.

Em função do surgimento de GPUs capazes de execução em paralelo várias áreas têm buscado investigar a viabilidade de seu uso em aplicações típicas. A área de bioinformática é uma das que vem buscando criar versões de suas ferramentas que sejam eficientes em ambientes desse tipo. Numa área como a bioinformática, que demanda fortemente

computação de alto desempenho, isso acaba se tornando bastante importante. Em específico um dos campos da bioinformática que demanda computação de alto desempenho é a área de alinhamento de sequências.

O alinhamento de sequências é um componente chave na análise de biossequências. Em virtude da grande quantidade de informações envolvidas surgiram métodos distintos para se atacar o problema. Em particular, esses métodos podem ser classificados em locais e globais, em que cada alinhamento é realizado buscando otimizar a similaridade local ou global das sequências. Como resultado disso diversas ferramentas foram desenvolvidas para realizar alinhamentos, como MUSCLE, FASTA, BLASTA, MUMmer, ClustalW, entre outras, cada qual utilizando abordagens apropriadas para atingir objetivos específicos [SCHA,2007]. Assim, temos ferramentas para alinhamento local, para alinhamento global e ainda algumas que apresentam os resultados dos melhores alinhamentos em lugar de apenas um alinhamento ótimo.

Nessa direção o presente projeto busca avaliar a adequação da aplicação de programação paralela em GPUs para o alinhamento múltiplo de sequências, na mesma linha de trabalhos como [WEIG,2007], [LIUY,2009] ou ainda [TRAP,2009]. Dada a complexidade envolvida na construção de ferramentas para alinhamento múltiplo, o projeto deve se ater à investigação do impacto de seu uso em fases específicas do processo de alinhamento. Em particular, espera-se atuar na fase do cálculo da matriz de similaridades, que em função do volume de elementos a serem determinados aparenta ser uma boa aplicação para o uso das GPUs.

1.2 Objetivos

O presente projeto tem como objetivo o estudo e teste dos algoritmos e métodos de alinhamento de sequências, se utilizando do algoritmo de Levensthein como algoritmo global para comparação [NAVA,2001] e os algoritmos de Smith Waterman e BLAST [CRUZ,2003]

como algoritmo local de alinhamento. Esses algoritmos são amplamente utilizados para identificar regiões de similaridades no DNA, RNA ou em uma proteína, que podem indicar um sinal de alguma funcionalidade, estrutural ou evolucional entre as sequências e trechos comparados. Assim, a avaliação de seus desempenhos quando executados em GPUs é bastante relevante.

1.3 Organização da monografia

Este trabalho está organizado de forma que no capítulo 2 se apresenta uma revisão da literatura relevante ao projeto, fazendo referencia a alguns princípios básicos utilizados na programação em placas de vídeos (GPU'S). O capítulo 3 descreve, com detalhes, o desenvolvimento do sistema proposto, desde seu estudo até sua implementação, enquanto no capítulo 4 são mostrados os resultados obtidos durante os testes realizados. Após a análise dos resultados, apresentação das conclusões e contribuições, dificuldades encontradas durante o projeto e direcionamento para possíveis trabalhos futuros são apresentadas no capítulo 5.

Capítulo 2

Neste capítulo são revisados conceitos importantes para que o trabalho possa ter uma melhor compreensão. Apresenta-se inicialmente a bioinformática, mostrando sua importância, algoritmos de comparação de cadeias e banco de genes. Nesse estudo os algoritmos que serão utilizados no projeto terão mais ênfase. Após essa revisão, apresenta-se a descrição de computação paralela explicando as vantagens de sua utilização, o do conceito de GPU, incluindo as informações básicas das GPUs da NVIDIA e sua linguagem CUDA.

2.1 Bioinformática

A informação contida em cada organismo ou a planta para o desenvolvimento potencial de qualquer atividade e para o individuo- é o material genético, DNA ou, em alguns vírus [KIMU,1983], RNA. Moléculas de DNA são longas, lineares e em cadeia contendo uma mensagem num alfabeto de quatro letras (A, G, C e T/U). Mesmo para microorganismos a mensagem é longa contendo normalmente 10^6 caracteres. O processo que a dupla-hélice realiza durante seu auto complemento interno e provê replicação acurada são bem conhecidos.

Uma replicação quase perto da perfeição é essencial para a herança; mas algumas replicações imperfeitas, ou o mecanismo para importar material genético desconhecido, também é essencial, tanto para a evolução quanto para que não existam organismos assexuados.

A implementação da informação genética acontece, inicialmente, através da síntese do RNA e proteínas. Proteínas são as moléculas responsáveis pela maior parte da estrutura e atividades de um organismo. Nosso cabelo, músculos, enzimas digestivas, receptores e anticorpos são proteínas. Tanto o acido nucléico quanto as proteínas são longas moléculas em cadeia linear. Tipicamente, proteínas são de 200 a 400 aminoácidos longos, precisando de 600 a 1200 letras de mensagem do DNA para especificá-los. Algumas regiões da sequência do DNA são devotadas como mecanismos de controle, e um montante substancial de genomas de organismos superiores aparentemente é "lixo", o que em parte pode significar que sua função pode não ter sido entendida.

Questões sobre como os tecidos se tornam especializados ou generalizados durante seu desenvolvimento, ou como o ambiente afeta e exerce o controle em eventos genéticos, podem ter sua resposta auxiliada pela bioinformática. Esses problemas fascinantes sobre o fluxo de informações e o controle que um organismos têm sobre eles, constituem o escopo principal da bioinformática [MEID,1997].

2.1.1 Algoritmos de comparação entre cadeias

Existem dois tipos básicos de medidas de similaridade de sequências, isto é uma medida para determinar o quanto uma sequência é diferente da outra, classificados como globais ou locais. Algoritmos de similaridade global fazem uma otimização de todo o alinhamento entre duas cadeias de sequências, onde podem existir grandes partes com baixa ou nenhuma similaridade. Diferentemente, os algoritmos locais comparam as similaridades localmente entre as cadeias distintas de genes [WANG,1994]. Uma simples comparação pode ter como resultado muitos alinhamento de sequências e regiões não conservativas que não contribuem em nada para medir a similaridade entre os espécimes analisados. [SMIT,1980] Medidas de similaridade local são amplamente utilizados em banco de dados genéticos.

O algoritmo de Levenshtein é exemplo de algoritmo global, portanto sua implementação computacional pode ser feita de maneira sequencial. O algoritmo de Smith-Waterman assim como a família dos algoritmos BLAST, se utiliza do método de similaridade local. Várias medidas de similaridade, inclusive os do BLAST e de Smith-Waterman se iniciam com uma matriz de pontuações onde são alocados todos os pares de resíduos (partes de uma sequência). Identidades e substituições conservativas têm pontuações positivas/negativas, dependendo de como o algoritmo é implementado, mas mantendo a premissa de igualdade maior ou menor. As substituições possuem pontuações negativas. A pontuação de similaridade de dois segmentos é a soma dos valores de similaridades para cada par de resíduos alinhados.

Um biólogo molecular pode estar interessado em todas as regiões idênticas compartilhadas entre as duas proteínas, e não apenas nos pares que possuem maior pontuação. Por esse motivo os algoritmos locais fazem busca em pares de segmentos de Maximo local com pontuações maiores que um certo valor.

2.1.2 Bancos de Genes

O GenBank®, mais conhecido banco de genes da internet, é um banco de dados público responsável por guardar as proteínas, DNA, RNA e genomas que já foram ou estão sendo mapeados. Nesse banco todos os mapeamentos das sequências de nucleotídeos e a tradução de suas proteínas estão guardadas e mantidas pelo NCBI(*National Center for Biotechnology Information)* como parte da INSDC(*International Nucleotide Sequence Database Collaboration)*. O NCBI é uma parte do Instituto Nacional da Saúde dos Estados Unidos da América.

O GenBank® e seus colaboradores recebem sequências produzidas em laboratórios de mais de 100.000 organismos distintos do mundo todo. Em mais de 20 anos desde que foi inaugurado, GenBank® se tornou o mais importante e influente banco de dados para a maioria das pesquisas em quase todos os campos da biologia. O GenBank® ainda continua a crescer em uma taxa exponencial, dobrando o número de informações que possui a cada dezoito meses. O projeto 155, por exemplo, produzido em agosto de 2006 possui mais de 65

bilhões de bases de nucleotídeos em mais de 61 milhões de sequências. O GenBank® é construído por submissões de laboratórios individuais, como também a submissões de genes de centros de sequenciamento de larga escala.

A Figura 1 ilustra como os genes estão distribuídos na procura do GenBank® :

//Oct-1B[Homo sapiens] Proteina.

```
LOCUS       AAB28879                  57 aa            linear
PRI 14-JAN-1994
DEFINITION  Oct-1B [Homo sapiens].
ACCESSION   AAB28879
VERSION     AAB28879.1  GI:440978
DBSOURCE    locus S66901S1 accession S66901.1
KEYWORDS    .
SEGMENT     1 of 2
SOURCE      Homo sapiens (human)
  ORGANISM  Homo sapiens
            Eukaryota; Metazoa; Chordata; Craniata; Vertebrata;
Euteleostomi;
            Mammalia; Eutheria; Euarchontoglires; Primates;
Haplorrhini;
            Catarrhini; Hominidae; Homo.
REFERENCE   1  (residues 1 to 57)
  AUTHORS   Das,G. and Herr,W.
  TITLE     Enhanced activation of the human histone H2B promoter
by an Oct-1
            variant generated by alternative splicing
  JOURNAL   J. Biol. Chem. 268 (33), 25026-25032 (1993)
   PUBMED   8227066
  REMARK    GenBank staff at the National Library of Medicine
created this
            entry [NCBI gibbsq 139675] from the original journal
article.
COMMENT     Method: conceptual translation supplied by author.
FEATURES             Location/Qualifiers
     source          1..57
                     /organism="Homo sapiens"
                     /db_xref="taxon:9606"
ORIGIN
        1 madggaasqd essaaaaaaa dwkskksfpa flitkllsvf nesvqrknav
flyltqe
```

//fim da Oct-1B[Homo sapiens] Proteina'

Figura 1 – Exemplo de entrada no GenBank®

Nesse documento (tirado do GenBank® no endereço eletrônico de http://www.ncbi.nlm.nih.gov/protein/AH004419.1) encontramos todas as informações biológicas da proteína Oct-1B, os autores do mapeamento, onde foi publicado e com qual nome foi publicado tal estudo e no final do arquivo encontra-se uma lista de caracteres que, nesse caso, vem a ser um vetor contendo o mapeamento dessa proteína.

2.1.3 Algoritmo de Levensthein

A distância Levensthein é uma medida de similaridade entre duas cadeias de caracteres. A distância é o número de informações que é apagado ou substituído para que uma sequência que é comparada fique idêntica à string usada como comparação. O algoritmo tem como base a programação dinâmica, usando uma matriz que possua dimensões iguais ao número de algarismos adicionados de um, das cadeias a serem comparadas. Nessa matriz a primeira linha e a primeira coluna são zeradas, e para o resto da matriz (a matriz é preenchida do noroeste para o sudeste) se os caracteres são idênticos então se assume o valor 0(zero) senão o valor 1 (um), e a tabela recebe o menor custo entre o valor da linha anterior, o valor da coluna anterior ou o valor da diagonal superior. Seu pseudocódigo é visto na Figura 2

```
//Inicio do programa
Inteiro TAX //Tamanho da sequência X
Inteiro TAY//Tamanho da sequência Y
Inteiro TAB[TAX][TAY]//Tabela de tamanho TAX e TAY
Para X de 0 até TAX
        TAB[X][0]=X
Para Y de 0 até TAY
        TAB[0][Y]=Y
Para X de 1 até TAX
        Para Y de 1 até TAY
                Se STRING[X]==STRING[Y]
                        Cost =0
                Senao
                        Cost=1
                TAB[X][Y]= menor entre(TAB[X-1][Y]+1,TAB[X][Y-1]+1,TAB[X-
1][Y-1]+cost)
Distância de Levenshtein = TAB[TAX][TAY]
//Fim do programa
```

Figura 2 – Algoritmo de Levensthein para alinhamento de sequências.

2.1.4 Algoritmo de Smith-Waterman

Algoritmo proposto pelos pesquisadores que dão nome ao algoritmo Temple F. Smith e Michael S. Waterman em 1981. Esse algoritmo de programação dinâmico, tem como propriedade procurar o alinhamento local otimizado em respeito a um sistema de pontos usados. O caminho contrário do rastreamento começa da célula matriz com o maior score e vai recuando até que a pontuação zero seja encontrada, determinando assim o melhor alinhamento.

Assim para cada célula o algoritmo calcula todos os caminhos possíveis de se chegar até essa determinada célula. Os caminhos podem ser de qualquer tamanho e podem possuir inserções e exclusões[WATE,1976].

Descrição do algoritmo

Duas sequências A=a1,a2...an e B=b1,b2,...bm. Uma similaridade entre s(a,b) é dada entre segmentos de a e b. exclusões de tamanho k são dadas pelo peso W[k]. Para encontrar pares de segmentos com alto grau de similaridade, construímos a matriz H, fazendo primeiramente:

$$H[k][0] = H[0][l] = 0 \ \ para\ 0 <= k <= n\ e\ 0 <= l <= m$$

Os valores da matriz são dados por

$$H[i][j] = Max\{H[i-l[j-l+s(a[i],b[j]), Max(k \geq l)\{H[i-k][j]-W[k], Max\{H[i][j-l]-W[l]\},0\}$$

A fórmula para H[i][j] segue as seguintes possibilidades para o final dos segmentos em quaisquer a[i] e b[j].

-Se a[i] e b[j] são associados, a similaridade é:

H[i-1][j-1]+s(a[i],b[j]).

-Se a[i] está no fim da exclusão de tamanho k, a similaridade é

H[i-k][j]-Wk

-Se b[j] está no fim da exclusão de tamanho l, a similaridade é

H[i-k[j]-Wl

-Finalmente, um zero é incluso para *prevenir que o calculo não tenha como resultado uma similaridade negativa.* [SMIT,1981]

2.1.5 Algoritmo BLAST

BLAST ou *Basic Local Alignment Search Tool* é um algoritmo para comparar sequências baseado no algoritmo de Smith-Waterman, como quando se faz a comparação entre sequências existentes na natureza existe a possibilidade de haver alinhamentos bons ou

não ou então não existir nenhum. O que se deseja são apenas os alinhamentos significativos, em que a pontuação seja maior que um número estipulado pelo pesquisador. É exatamente isso o que o BLAST faz, não explorando todo o espaço de pesquisa entre duas sequências.

O Algoritmo BLAST se utiliza de três regras para refinar sequencialmente os potenciais pares de pontuação alta. Estas regras permitem ao BLAST, a partir de um refinamento gradual, amostrar todo o espaço de pesquisa sem perder tempo em regiões sem similaridade. Essas regras são, *"compiling a list of high-scoring words, scanning the database for hits, and extending hits*" que em uma tradução livre significam, compilar uma lista de palavras com score alto, procurar no banco os acertos e estender os acertos.

O encurtamento do espaço de pesquisa é a chave para que o BLAST seja considerado mais rápido que o Smith-Waterman, mas à custa da perda de sensibilidade, pois na tentativa de melhorar o desempenho do algoritmo pode ser que algumas sequências que possuem semelhanças, passem despercebidos. O acerto entre velocidade e sensibilidade é a chave na elaboração de alinhamento com BLAST.

O algoritmo BLAST usa como comparação entre sequências o algoritmo de Smith-Waterman. Entretanto existe um pré-processamento das sequências analisadas para que as possíveis partes que tem baixa probabilidade de ter uma pontuação alta sejam eliminadas. O BLAST pode ser implementado de diversas maneiras, sendo nesse projeto adotado a seguinte forma:

Algoritmo BLAST:

1. Remove-se regiões de baixa complexidade ou sequências repetidas durante a sequência analisada.

   As regiões de baixa complexidade são aquelas que são compostas por poucos tipos de elementos. Essas regiões podem ter uma pontuação alta que confunde o programa para achar as sequências significantes na database.

2. Utilizar palavras de k-letras da sequência analisada. (usamos apenas minipalavras para a comparação entre sequências).
3. Listar as possíveis palavras correspondentes.
4. Organizar as palavras de score alto restante em uma árvore de busca.
5. Repetir o passo 3 até o 4 para cada palavra de k-letras na sequências de consulta.

6. Procurar no banco de dados sequências que sejam idênticas com as palavras de pontuação alta.
7. Estender as correspondências exatas de alta pontuação (HSP), aumentando-se o número k.
8. Listar todas as HSPs em um banco de dados na qual apenas são considerada palavras com um score acima do permitido.
9. Avaliar a significância do HSP score.
   Mostrar o alinhamento Smith-Waterman local da consulta e cada umas das sequências que existem no banco correspondente.
10. Reportar cada semelhança na qual o score é maior que certo parâmetro.

Dependendo de como ele é implementado, ele pode ser otimizado para alguns tipos de cadeias a serem analisadas, BLASTN, por exemplo, é utilizado para analisar DNA com DNA, enquanto o BLASTP é usado para analisar proteína com proteína e o PSI-BLAST usado para encontrar relação entre proteínas distantes e etc.

2.1.6 Outros Algoritmos

Alem dos algoritmos aqui apresentados, são encontrados na literatura diversos algoritmos para alinhamento de sequências. Entre eles se encontram:

Hamming Distance: Este algoritmo, na sua versão simplificada, permite apenas substituições com custo 1. Na literatura este problema de busca é chamado de “string matching with k mismatches” [NAVARRO 01]. Outra definição [CHAPMAN 05] é: número de bits que diferem entre duas strings binárias, ou número de bits que precisam ser modificados para transformar uma string idêntica à outra.:

Este algoritmo é bastante eficiente, mas não é aplicável ao problema de alinhamento, já que este permite apenas substituições não permitindo nem inserções, nem remoções. Isto inviabilizaria comparações, por exemplo, de “datanascimento” com “data-nascimento”.

Bit-Parallelism: Não é um algoritmo, e sim uma técnica que visa explorar o paralelismo dos computadores quando estes trabalham com bits. A ideia básica é programar outro algoritmo de forma paralela utilizando bits. Existem dois algoritmos que são mais comumente implementados desta forma: autômatos não determinísticos; e programação dinâmica de matrizes (Levenshtein). Seja w o tamanho de uma palavra de máquina que pode ser de 32 ou 64 bits. Pode-se utilizar uma máscara para representar diversas palavras utilizando uma única palavra de máquina, possibilitando que sejam feitos matchs em conjuntos de caracteres em vez de ser feito em um único caractere. Esta técnica melhora a eficiência dos algoritmos de Levenshtein, e os valores das pontuações obtidas nas comparações não são modificados. Ela poderia ser utilizada na nossa aplicação para que houvesse um ganho na eficiência.

Stochastic Model Este algoritmo também é semelhante ao de Levenshtein, pois igualmente se baseia em inserções, remoções e substituições para atribuir escores a uma matriz. Pontuações estas que são atribuídos através de programação dinâmica [RISTAD and YANILOS 96]. Este algoritmo é muito eficiente para aprender os custos de edição de um corpo de exemplos, ou seja ele adapta a maneira como ele atribui os escores dependendo do corpo de exemplos que este algoritmo recebe. E tudo isto é feito de maneira automática. O seu desempenho é semelhante ao do algoritmo de Levenshtein.

A grande vantagem deste algoritmo, segundo os seus autores, é que ele apresenta apenas um quarto da taxa de erro do algoritmo de Levenshtein. Este algoritmo pode ser aplicado a qualquer problema de classificação de strings que possa ser resolvido usando funções de similaridade, embora tenha sido testado apenas para o aprendizado de palavras em conversações.

Ele alinha cadeias globalmente, podendo atribuir baixos escores para duas strings com sufixos semelhantes como é o caso, por exemplo, de “telephone” e “phone”. Mas é possível adaptar este algoritmo para que ele passe a alinhar strings localmente. A grande vantagem de utilizar este algoritmo na nossa aplicação seria a taxa de erro bem menor do que a encontrada nos algoritmos derivados do Levenshtein.

2.2 Computação Paralela

O paralelismo é o futuro da computação. O futuro do desenvolvimento de microprocessadores estará concentrado em adicionar núcleos ao invés de aumentar a velocidade dos processadores. A computação paralela é um paradigma dominante nas arquiteturas de computadores sob a forma de processadores multi-núcleos. A maioria dos aparelhos eletrônicos lançados no mercado hoje em dia se utiliza de processamento paralelo e processadores multi-núcleos.

A técnica de paralelismo já vem sendo empregada há alguns anos, principalmente na computação de alto desempenho. Esse tipo de computação tem como característica o uso de recursos computacionais que são aproximadamente uma ordem de grandeza superiores aos recursos normalmente disponíveis, como desktops ou estações de trabalho. A demanda por essas técnicas de software e hardware vem da demanda crescente por aplicações que necessitam de grande poder computacional para fins de simulação, escalabilidade, computação científica ou bioinformática, como no caso desse trabalho.

2.2.1 A programação paralela

Tradicionalmente um programa é escrito sequêncialmente e executado sequêncialmente. Dado um certo problema, um algoritmo é construído e implementado como um fluxo de instruções em série. Após isso, essas instruções são executadas pela UCP de um computador, sendo que apenas uma instrução é executada por vez num fluxo em fila.

A computação paralela se utiliza de diversos elementos de processamento simultaneamente para resolver, um determinado problema, em que cada parte menor do problema é resolvida simultaneamente por uma unidade central de processamento diferente. Tal conceito é conhecido como “Divide and Conquer” (Dividir e Conquistar), sendo a idéia principal reduzir um grande problema em problemas menores e dividir esses problemas menores para que possam ser resolvidos todos ao mesmo tempo por entidades diferentes, para que o problema maior possa ser resolvido mais rapidamente. O paralelismo pode ser dado por vários meios, em bit, instrução, de dados ou de tarefa.

A grande vantagem da programação e do processamento paralelo é a grande quantidade de recursos que você pode obter por um custo financeiro reduzido. Tais recursos podem ser escaláveis dependendo de como uma determinada aplicação é implementada.

Entretanto, apesar dessa quantidade quase ilimitada de recursos, a programação paralela possui alguns problemas que devem ser resolvidos na sua implementação, fazendo que a implementação de um código paralelo seja mais demorada. Essa depuração complicada se deve ao fato que pontos sem importância na programação sequêncial começam a ter maior importância na programação paralela. Um dos principais problemas enfrentados é o uso de informações ou recursos críticos, na qual mais de um processo pode acessá-lo e alterá-lo ao mesmo tempo. Tais conflitos devem ser resolvidos, como citado anteriormente, durante a construção de seu código.

Outro problema, com menor importância nos dias atuais, é a necessidade de replicação do código, fazendo com que o programa necessite de mais memória. Entretanto como atualmente o maior problema da computação de alto desempenho não é a memória e sim o tempo, esse problema muitas vezes não interfere no resultado.

Atualmente muitos aparelhos que requerem algum tipo de processamento complexo estão se utilizando de processadores *multicores*. Exemplo disso são os computadores pessoais, *tablets e* smartphones. Vale ressaltar também o uso de *multicores* em processamentos gráficos em placas de vídeo, que são a base deste projeto.

2.3 GPU'S

No começo dos anos 90, a interatividade por meio de gráficos 3D ainda era coisa de ficção científica. No final da década, entretanto, a maioria dos computadores possuíam uma unidade de processamento gráfico (GPU-*graphic processing unit)* ou unidade de processamento visual (VPU-*visual processing unit)* dedicada exclusivamente para prover uma visualmente rica, experiência interativa 3D de alto desempenho.

A mudança dramática era a inevitável consequência da demanda por videogames, avanços na tecnologia de fabricação, e a exploração do paralelismo inerente no "*feed-foward"* pippeline gráfico. Hoje, o poder computacional puro das GPUs sobrepõem a maioria das CPUs mais potentes, e a diferença está constantemente aumentando.

Entretanto, GPUs se distanciaram da tradicional função de pipeline de gráficos 3D para uma máquina computacional de múltiplas facetas. Hoje, GPUs podem ser usadas para implementar diferentes algoritmos paralelos diretamente usando seu hardware gráfico. Algoritmos bem construídos que aproveitam de forma eficiente toda a potencia computacional possíveis desses mecanismos, tem se mostrado uma tremenda vantagem em relação ao tempo de execução. Na verdade, a GPU foi o primeiro aparelho computacional paralelo amplamente utilizado em *Desktops.*

O principal objetivo de qualquer sistema gráfico 3D é o de sintetizar uma imagem de uma determinada cena, 60 vezes por segundo para um gráfico em tempo real como nos videogames. Essa cena contem as primitivas geométricas a serem vistas assim como a descrição da iluminação na cena, e a posição do usuário que enxerga as imagens assim como sua orientação[DAVI,2007].

As GPUs modernas usam a maioria de seus transistores para realizar cálculo de gráficos computacionais 3D. Inicialmente eram usados para acelerar o trabalho intensivo da

memória na texturização e renderização de polígonos, mais tarde adicionando unidades para acelerar cálculos geométricos como a rotação e translação de vértices em diferentes coordenadas do sistema.

Desenvolvimentos recentes em GPU incluem o suporte para programação de *shaders* que podem manipular vértices e texturas com muitas das mesmas operações que a CPU suporta, técnicas de interpolação e oversampling que reduzem o aliasing. Como muito desses métodos computacionais envolvem operações com matrizes e vetores, engenheiros e cientistas tem aumentado o estudo no uso das GPUs para cálculos não gráficos.

2.3.1 NVIDIA

A NVIDIA® é uma empresa de tecnologia multinacional que tem sua sede em Santa Clara, Califórnia. A companhia em si foi inventora das GPUs em 1999. Tem como principais produtos processadores gráficos, processadores de comunicação wireless, núcleo de *motherboards* e players de média. Na sua comunidade de usuários de computadores é muito conhecida por seus produtos da linha GeForce®, que consiste em uma linha completa tanto de chips gráficos discretos, encontrados em placas de vídeo, e tecnologia de gráficos usando em placas mãe que utilizam a nForce, como o console da Microsoft o Xbox e o PlayStation 3 da Sony.

Entre as famílias de processadores gráficos da NVidia se destacam:

NVIDIA® GeForce é uma família de GPUs desenvolvida pela NVIDIA® Os primeiros produtos da família GeForce eram vendidos para os consumidores de placas aceleradoras 3D de alto desempenho, mas os produtos seguintes expandiram a linha para todos os públicos de placas gráficas.

NVIDIA® Tegra é uma família de GPUs baseada em "System-on-a-chip"(Um sistema inteiro em um chip apenas), desenvolvida pela NVIDIA® para aparelhos moveis como smartphones e outros aparelhos de internet móvel. A família Tegra traz CPU com

arquitetura ARM, uma GPU "northbridge e southbridge", e controle de memória em apenas um pacote. Essa família enfatiza o baixo consumo de energia e alto desempenho para reprodução de media de áudio e vídeo.

NVIDIA® Quadro é uma GPU que utiliza a PCI Express produzida também pela NVIDIA®. Os seus desenvolvedores aperfeiçoaram a aceleração de ferramentas CAD e DCC, essas placas são normalmente utilizadas em Workstations(diferente da GeForce que tem como principal nicho econômico os jogadores de games).

NVIDIA® Tesla família de GPUs da NVIDIA® onde as GPUs são utilizadas especialmente para o processamento de operações que exigem um alto desempenho de computador, essas são as GPUs que comumente são utilizadas para GPGPU (*General Purpose Graphics Processing Unit* ou unidade de processamento gráfico de uso geral). As GPUs dessa família, são feitas exclusivamente para processamentos que necessitem de alto desempenho e que possam ou não ser processamento gráfico.

2.4 CUDA

CUDA é a linguagem de arquitetura paralela da NVIDIA®. Com ela é possível verificar um grande salto no desempenho da computação se utilizando o poder das GPUs. Com milhões de GPUs que tem suporte para CUDA vendidas até o dia de hoje, desenvolvedores de software, cientistas e pesquisadores tem encontrado usos abrangentes para CUDA, incluindo processamento de imagens e vídeos, bioinformática e química, simulação da dinâmica dos fluidos, reconstrução de imagens CT, análise sísmicas, trajetória de raios, previsão do tempo e muito mais.

A linguagem CUDA tem sido aceita com muito entusiasmo pela área acadêmica e científica. Por exemplo, o AMBER agora tem como motor a linguagem CUDA, essa que é um programa de simulação dinâmico usado em mais de 60.000 centros de pesquisa, e indústrias farmacêuticas multinacionais que tem como objetivo acelerar a descoberta de novas drogas.

2.4.1 Por que usar GPUs ao invés de CPUs para o paralelismo?

Em reportagem para o The New York Times(11 de outubro de 2011) Steve Lohr relata os planos para a construção do Titan, que quando estiver pronto, será o supercomputador mais rápido do mundo, passando o atual campeão o supercomputador K que se encontra no Japão. O Titan tem o potencial de ser duas vezes mais rápido do que seu concorrente direto.

Até a presente data, essa poderia ser uma informação que passaria despercebida, pois apesar de um número surpreendente, a construção de supercomputadores passou a ser algo um tanto comum devido à periodicidade na qual noticias desse gênero tem vindo a público. Entretanto o que realmente prende a atenção é que o supercomputador Titan pode ser até três vezes mais eficientes em relação ao seu uso de energia, tendo como grande responsável por isso a sua arquitetura baseada em GPUs.

O gasto de eletricidade dos supercomputadores é um dos principais fatores limitantes para que não houvesse atualizações significativas em relação ao poder computacional dessas superestruturas. Tendo em vista o consumo reduzido de eletricidade das GPUs, essa parece ser por enquanto a melhor resposta para a programação em larga escala.

O Titan terá um design híbrido, usando microprocessadores com GPUs NVIDIA®. A primeira fase desse projeto será um upgrade no supercomputador já existente da instituição até o fim do ano 2011. No começo do ano 2012, ocorrerá a mudança completa usando 18,000 placas de vídeo de forma paralela. Apesar desse planejamento, o tamanho e o quanto grande será o Titan, dependerá do quanto a agência de energia irá custear o projeto que pode, dependendo dos valores de outros supercomputadores, alcançar a marca de 200 milhões de dólares.

Capítulo 3

O capítulo três tem como principal objetivo, descrever de maneira detalhada todo o projeto, incluindo o estudo e levantamento das informações necessárias, definição do projeto, arquitetura e implementação dos algoritmos propostos e a devida conclusão destes.

3.1 O projeto

Nesta seção é apresentado o desenvolvimento dos programas utilizando os algoritmos propostos para alinhamento de sequência. O primeiro programa que utiliza o algoritmo de Levensthein tem como objetivo formar uma base de comparação para identificar ganhos de tempo de execução após a execução das unidades que utilizaram os algoritmos de Smith-Waterman e BLAST.

Utilizando uma sequência longa qualquer como base, retirada do GenBank® para determinar um tempo de processamento, é possível determinar se existirá uma economia do custo tempo após algumas execuções com diferentes sequências de diferentes tamanhos e comparar os resultados dos três algoritmos entre si.

Após o desenvolvimento, o projeto entra em outra fase, uma fase de testes na qual, são determinados os resultados dos programas desenvolvidos no trabalho.

3.2 Estudos e levantamento

Durante essa etapa houve um levantamento aprofundado sobre os algoritmos de alinhamento de sequências, seu funcionamento e como deveriam ser implementados em paralelo. Foi durante esse estudo inicial, que a linguagem de programação em GPUs CUDA, começou a ser estudada e analisada. Nessa etapa, houve contato excessivo com diversos estudos, livros, linguagens e trabalhos que apesar de terem ligação com o projeto proposto, não foram diretamente aproveitados na sequência do projeto.

3.3 Definição do projeto

Após o levantamento de diversas informações e da definição de um caminho a seguir no projeto, houve durante essa fase um direcionamento mais específico em relação ao estudo a ser feito. Nessa etapa as informações não relevantes e de menor importância ao projeto foram descartadas, sobrando assim apenas material relacionado ao escopo especifico deste projeto.

Durante essa etapa os algoritmos Levensthein, Smith-Waterman e BLAST, foram escolhidos como melhor opção para esse projeto, tendo em vista seu desenvolvimento natural e tomando como base, a evolução na qual esses três algoritmos têm correlação.

3.3.1 A escolha dos algoritmos

Tanto o algoritmo de Levensthein, Smith-Waterman e o BLAST são algoritmos que seguem certa linha cronológica. O Levensthein, o único algoritmo global dos três, na verdade é utilizado como base no Smith-Waterman que é base do BLAST, nesse caso a evolução dos algoritmos se dá da seguinte maneira:

Levensthein - Algoritmo global de comparação entre sequências.

Smith-Waterman- Algoritmo local de alinhamento de sequências, trabalha utilizando o algoritmo de seu predecessor para realizar o alinhamento em partes pequenas de uma sequência maior.

BLAST- Algoritmo local de alinhamento de sequências que utiliza um pré-processamento de informações para excluir possíveis interações que dependendo do contexto podem se tornar desnecessárias. Depois realizar o Smith-Waterman apenas nas sequências que já passarem pelo pré-processamento e que assim são viáveis dependendo de seu escopo.

3.4 Implementações

Durante essa etapa, que foi dividida em três partes, teve início a implementação dos três algoritmos utilizando a linguagem de programação em GPUs CUDA.

3.5 Levensthein

A implementação desse algoritmo seguiu sem problemas. Devido a sua relevada facilidade de implementação, apesar do sentimento pré-implementação de que poderia ocorrer um estouro no uso de memória, devido a grande quantidade de informações que uma fita de DNA possui. O problema não ocorreu em nenhum momento tendo em vista que sua implementação foi dinâmica, deixando assim o trabalho focado apenas na implementação do algoritmo em si, sem desviar o planejamento feito.

Vale ressaltar que o algoritmo implementado segue à risca o algoritmo e explicações apresentados no capítulo dois sessão *2.1.3*, com devidas alterações para a linguagem C++.

3.6 Smith-Waterman

O passo seguinte, após a implementação do algoritmo Levensthein, foi a implementação do programa que carregaria o Smith-Waterman, programa a ser executado paralelamente utilizando GPUs. Tendo em vista sua implementação mais elaborada, o programa foi escrito em três arquivos principais:

1- O arquivo SmithWaterman.cu é o programa em si, contendo a chamada da principal função e leitura de arquivos para uma eventual facilidade na leitura das cadeias a serem analisadas.

2- O arquivo seguinte, SmithWaterman_gold.cpp possui as principais funções do programa utilizadas em linguagem C++ (cpp), como a função para ler um arquivo com sequências, determinar o tamanho das cadeias de proteínas e etc.
3- Por ultimo temos o arquivo SmithWaterman_kernel.cu que possui a matriz blosum utilizada durante as substituições no alinhamento de sequências, e o alinhamento que cada processador da GPU devera realizar.

O algoritmo utilizado para a implementação é o mesmo descrito durante o capítulo dois sessão 2.1.4, exceto as mudanças necessárias de linguagem de programação para transformar a linguagem utilizada na linguagem CUDA.

3.7 BLAST

Após a implementação do Smith-Waterman, teve inicio a implementação do programa que utiliza o algoritmo BLAST, seguindo o mesmo exemplo do Smith-Waterman algoritmo utilizado foi o exemplificado anteriormente no capítulo dois sessão 2.1.5.

A implementação deste algoritmo teve como base a implementação disponível na biblioteca de arquivos NCBI, biblioteca em linguagem C utilizada na implementação BLAST que o Genbank® utiliza. Essa biblioteca pode ser encontrada em ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/2.2.25 (FTP oficial da ncbi).

A implementação do BLAST devido sua complexidade, foi escrita em 3 arquivos:

1- O arquivo gpu_blast.cu possui as principais funções escritas na linguagem CUDA, a própria chamada do programa BLAST, a leitura de arquivos de entrada e a função para liberar a memória utilizada

2- Enquanto isso o arquivo de nome gpu_blast.c possui a matriz blossom para as substituições e pequenas funções em C para copiar o número Maximo da matriz.

3- E durante o arquivo gpu_blast_kernel.cu, o principal arquivo, é especificado o que cada core CUDA precisa realizar .

Capítulo 4

Nesse capítulo são mostrados os resultados obtidos nesse projeto, apenas os resultados por eles mesmos, de maneira sucinta e explicando como eles foram obtidos sempre em conjunto com outro código para que, mesmo sem haver conclusões nesse capítulo (que ficará a cargo do capítulo cinco) existam dados para comparação.

4.1 Análise

O capítulo quatro expressa de maneira mais sucinta os resultados obtidos no projeto e os experimentos feitos. Os programas foram rodados em mais de uma maquina e por mais de uma vez para que os resultados fossem mais coerentes e precisos. Os resultados foram obtidos utilizando sequências distintas.

Para fins de comparação são rodados os mesmos algoritmos paralelos mais duas vezes, nesses novos casos, o numero de threads por bloco foi reduzido pela metade para podermos

comparar os resultados entre si. O primeiro teste será feito com 100% dos threads depois são utilizadas apenas 50% e depois 25% threads por bloco.

Foi utilizado para comparação também a implementação sequencial dos algoritmos Smith-Waterman e BLAST, nesse caso os códigos não foram implementados no projeto, sendo utilizada a implementação pronta (by Argonne National Laboratory), para a comparação dos algoritmos sequenciais e paralelos.

Foram coletadas 24 amostras das quais os tempos de execução são levadas em consideração. Para que o tempo de execução seja o mais fiel possível, são previamente eliminados dois resultados com menor tempo e dois resultados com maior tempo, eliminando-se assim as anomalias que podem aparecer.

Entretanto se existir algum ambiente no qual o menor tempo seja algo fora de padrão e significativo, tendo em vista que esse projeto é uma análise de desempenhos, houve tentativa de analisar de novo o resultado ou uma tentativa de clonagem desse resultado. Vale ressaltar que apesar do ambiente controlado proposto para as execuções, podem ocorrer alguns imprevistos.

4.2 TESTE 1

As analises se dão da seguinte maneira, o programa é preenchido com duas cadeias conhecidos e as funções de tempo dentro do programa retornaram apenas o tempo de execução do código. No caso do BLAST é contado também o tempo de pré-execução, onde acontece uma pré-seleção e filtro de informações que podem se tornar desnecessárias, pois isto é o que torna o BLAST diferente do algoritmo de Smith-Waterman.

Para facilidade de uso, daremos o nome de TESTE 1 para os testes realizados com as duas cadeias descritas a seguir. O primeiro par de sequências, que configura a primeira bateria de testes, será o genoma completo do Salivirus NG-J1 retirado do GenBank®

(http://www.ncbi.nlm.nih.gov/nuccore/NC_012957) o genoma em si é uma cadeia que possui 7983 proteínas (A, C, G, T).

A segunda sequência que configura o TESTE 1(teste entre o Salivirus e essa cadeia) é o nucleotídeo JP 20111429140-A/16045 encontrado no Homem e responsável pela fibrose, e pode ser encontrada no endereço http://www.ncbi.nlm.nih.gov/nuccore/HV472932.1 também parte integrante do GenBank®, essa sequência é relativamente menor, contendo 202 caracteres também distribuídos em A, C , G e T.

A comparação entre essas duas cadeias é usado como principal método de avaliação dos algoritmos apresentados tendo em vista o tamanho da cadeia apresentada anterior, cadeia contendo mais de sete mil novecentos e oitenta e um caracteres, podendo assim formar base para se concluir que os resultados obtidos aqui é mais significativos que os outros testes nos quais possuem cadeias menores.

4.2.1 GeForce®9300M GS

Utilizando um notebook contendo uma placa de vídeo GeForce® 9300M GS, que possui 8 núcleos, um processador Intel® Core™2 Duo CPU p8700 @ 2,53GHz e 2,53GHz e 4GB de RAM, em dois sistemas operacionais diferentes, (Windows 7 64 bits e Linux distribuição 64bits nesse caso o*Ubuntu*)

A Tabela 1 contem os resultados obtidos nesse ambiente utilizando os algoritmos sequenciais, paralelos utilizando 100%, 50% e 25% dos Threads.

Tabela 1 - Tempos Médios e *Speedups* GeForce® 9300M GS TESTE 1

	Sequencial	Threads 100%	Threads 50%	Threads 25%
Levensthein *Windows* 7	0,6316	-	-	-
Levensthein *Linux*	0,6222	-	-	-
Smith-Waterman *Windows* 7	0,6330	0,3458	0,3865	0,4355
Speedup	-	1,8305	1,6377	1,4535
Smith-Waterman *Linux*	0,6285	0,3230	0,3725	0,4045
Speedup		1,9458	1,6872	1,5537
BLAST *Windows* 7	0,5970	0,3394	0,3490	0,3810
Speedup		1,7589	1,7106	1,5669
BLAST *Linux*	0,5480	0,2822	0,3140	0,3595
Speedup		1,9418	1,7452	1,5243

Podem ser observados ganhos máximos de 1,830 vezes no Windows 7 e 1,945 vezes no sistema operacional Linux para o algoritmo Smith-Waterman. Ocorreu um *Speedup* de 1,830 vezes no Linux e 1,758 vezes no Windows 7 para o BLAST, esse ganho considerável foi em relação aos testes sequenciais e aos testes na qual o número de threads foi de 100% do possível, entretanto quando o numero de threads diminuiu, o tempo de execução foi maior nos casos de 50% e 25%.

A provável causa dos testes rodados a 100% dos Threads ser mais rápido do que os testes a 50% e 25%, pode ter sido ocasionada pela velocidade na qual a máquina esgota as informações para transferi-las para a GPU. Tendo em vista que o ambiente na qual esses testes foram realizados é o ambiente com o menor poder computacional, tanto em termos de CPU quanto de GPU, esses resultado já eram esperados.

4.2.2 GeForce® GTS 450

A segunda bateria de testes foi realizada em outro computador, que possui processador Intel® Core™ i5 CPU 650 @ 3.20GHz 3.19GHz, com 8 GB de RAM e uma GeForce® GTS 450, placa essa que diferentemente da anterior utilizada, que possui 192 núcleos CUDA. Os resultados em tempo médio são encontrados na Tabela 2.

Tabela 2 - Tempos Médios e Speedups GeForce® GTS 450

	Sequencial	Threads 100%	Threads 50%	Threads 25%
Levensthein *Windows* 7	0,4125	-	-	-
Levensthein *Linux*	0,4050	-	-	-
Smith-Waterman *Windows* 7	0,4205	0,1061	0,0955	0,1120
Speedup		3,9632	4,4031	3,7544
Smith-Waterman *Linux*	0,3460	0,0938	0,0890	0,0970
Speedup		3,6886	3,88764	3,5670
BLAST *Windows* 7	0,4205	0,0952	0,0890	0,103
Speedup		4,4170	4,7247	4,0825
BLAST *Linux*	0,3460	0,0755	0,0665	0,0825
Speedup		4,5827	5,2030	4,1939

Nesse caso, o melhor *Speedup* do Windows foi de 4,4031 e para o Linux obtemos o melhor *Speedup* de 3,8876 no Smith-Waterman. *O que podemos observar é um Speedup* maior em relação ao Windows, entretanto olhando em números reais podemos ver que o programa executado no Linux possui melhor desempenho. O que pode ser aferido aqui é que o Windows possui um gerenciamento de recursos menos eficiente que o Linux.

No caso do BLAST a implementação paralela consegue um *Speedup* de 4,7247 em relação a implementação sequencial, entretanto apesar de haver um ganho no custo tempo quando o numero de threads é reduzido pela metade, quando é reduzido a 25% do seu numero original, o tempo de execução piora em relação ao uso de 100% das threads, o que tem causa provável devido a arquitetura dessa placa de vídeo ou a arquitetura desse computador.

4.2.3 GeForce® 465 GTX

Os últimos testes foram realizados utilizando uma terceira máquina diferente, nesse caso um computador contendo Processador Intel® i7 i860 @ 2.8 GHz(processador com 4 núcleos)8 GB RAM e uma GPU GeForce® GTX 465, contando com 352 núcleos CUDA. Apenas para comparações, as placas topo de linha da NVIDIA® da linha Tesla tem aproximadamente 512 núcleos CUDA.

Nesse ambiente, o tempo de execução dos algoritmos executados sequencialmente foi menor se comparado aos ambientes anteriores, isso se deve ao fato da capacidade computacional mostrado por esse computador. Alem das implementações sequências, todos os testes paralelos tiveram tempo de execução menor se comparado aos ambientes anteriores, tanto para as Threads a 100% como 50% e 25%.

Tabela 3 - Tempos Médios e Speedups GeForce® 465 GTX

	Sequencial	Threads 100%	Threads 50%	Threads 25%
Levensthein *Windows* 7	-	-	-	-
Levensthein *Linux*	0,3411	-	-	-
Smith-Waterman *Windows* 7	-	-	-	-
Speedup	-	-	-	-
Smith-Waterman *Linux*	0,3350	0,0805	0,0675	0,0560
Speedup		4,1614	4,9629	5,9821
BLAST *Windows* 7	-	-	-	-
Speedup	-	-	-	-
BLAST *Linux*	0,2470	0,0541	0,0450	0,0285
Speedup		4,5656	5,4888	8,6666

Como pode ser observado na Tabela 3, não existem testes feitos no sistema operacional Winodws 7 (64 bits). Foram realizados apenas testes no sistema Linux. Utilizando essa GPU citada anteriormente e o Smith-Waterman ocorreu um *Speedup* máximo de 5,9821 com Threads a 25%.

Nesse caso os resultados obtidos foram normais, a menor quantidade de Threads enviadas por blocos reduziu o tempo de execução, nesse caso ocorreu um *Speedup* de 8,666 com as Threads a 25% utilizando o algoritmo BLAST.

4.3 TESTE 2

A segunda a bateria de testes utiliza cadeias de tamanhos mais semelhantes, nesse caso uma parte da proteína Brucella Canis GSS (http://www.ncbi.nlm.nih.gov/nucgss/FN555005.1) que constitui cadeia formada por 1545 caracteres dividido em um alfabeto de quatro caracteres e a proteína COY59 Canis (http://www.ncbi.nlm.nih.gov/nucgss/E1522359.1) contendo 353 caracteres divididos no mesmo alfabeto, ambas as cadeias foram retiradas do GenBank®.

4.3.1 GeForce®9300M GS

A segunda bateria de testes é implementada apenas para esse ambiente específico da placa GeForce® 9300M GS, pois nos outros ambientes estudados nesse projeto, o tempo de execução do algoritmo ficou abaixo do limite sensível do tempo medido, fazendo assim com que os testes rodados nesse ambiente tivessem que ser descartados, entretanto quando utilizado no ambiente descrito mais a frente, o custo tempo pode ser medido.

Esse ambiente se configura com GeForce® 9300M GS, que possui 8 núcleos, um processador Intel® Core™2 Duo CPU p8700 @ 2,53GHz e 2,53GHz e 4GB de RAM assim como o primeiro ambiente utilizado no TESTE 1.

Tabela 4 - Tempos Médios e Speedups GeForce® 9300M GS TESTE 2

	Sequencial	Threads100%	Threads 50%	Threads25%
Levensthein *Windows* 7	0,4125	-	-	-
Levensthein *Linux*	0,4050	-	-	-
Smith-Waterman *Windows* 7	0,4330	0,2183	0,2230	0,2283
Speedup		1,9835	1,9417	1,8966
Smith-Waterman *Linux*	0,4100	0,1944	0,2044	0,2200
Speedup		2,1090	2,0058	1,8636
BLAST *Windows* 7	0,4125	0,2269	0,2381	0,2425
Speedup		1,8179	1,7324	1,7010
BLAST *Linux*	0,3725	0,1677	0,1725	0,1799
Speedup		2,2212	2,1594	2,0705

Com *Speedup* máximo de 1,9835 para o Windows 7 e 2,109 para o Linux durante o Smith-Waterman, foram obtidos resultados muito semelhantes com o primeiro (Tabela 1) teste realizado nesse mesmo ambiente, tanto seus *Speedup*s e seus resultados quando o número de threads foi sendo dividido pela metade.

Foi alcançado um *Speedup* de 1,8179 para o Windows e 2,2212 para o Linux durante o BLAST, e da mesma maneira que os resultados anteriores demonstraram, houve uma semelhança para o TESTE 1 (Tabela 1) realizado nesse mesmo ambiente em relação aos *Speedups alcançados*.

4.4 Comparações com outros trabalhos

Utilizando os trabalhos já realizados e que são base desse projeto, podemos fazer algumas comparações entre os *Speedup*s conseguidos. Tomando como base o maior *Speedup* conseguido de 8,666 que foi conseguido no ambiente com a GeForce® 465 GTX, podemos fazer uma comparação com os resultados obtidos em [WEIG,2007], um número muito próximo aos 9,2. Esse resultado foi obtidos num ambiente com um processador Pentium 4® contendo 1GB de RAM e com a placa de video GeForce® 7900 GTX. Também podemos citar um *Speedup* muito parecido com outro ambiente, nesse caso o computador conta com o mesmo processador e mesma memoria RAM, mas difere enquanto a placa de video utilizada, uma GeForce® 6800 GTO, nesse caso foi realizando um teste aproximadamente 64 vezes maior que o teste anterior realizado nesse trabalho, atingindo um *Speedup* de 7,5.

Comparando o resultado obtido durante todos os testes, ainda não foi possível conseguir um *Speedup* parecido com o obtido em [LIUY,2009]. Nesse trabalho, são reportados números 26 vezes mais eficiente, entretanto durante esse trabalho, os dados obtidos tem ambientes diferentes. Os algoritmos implementados de forma sequencial não foram rodados no mesmo computador, para a implementação paralela. Temos um computador contendo um processador AMD® Opteron 248 @ 2,2 GHz com 2GB de RAM e uma placa de vídeo GeForce® GTX 280 contando 30 núcleos para as execuções paralelas, enquanto os testes sequenciais foram executados em um computador com um processador Pentium 4® @ 3,0 GHz com 1 GB de memória RAM sem placa de vídeo dedicada.

Capítulo 5

Este último capítulo apresenta as conclusões que podem ser tiradas a partir dos dados obtidos no capítulo anterior e também apresenta os possíveis trabalhos futuros que podem ser desenvolvidos a partir deste.

5.1 Conclusões

Utilizando como base todos os resultados obtidos durante o capítulo quatro, podemos concluir o seguinte. Existe redução de tempo instaurada no algoritmo BLAST em relação ao Smith-Waterman, e se as cadeias analisadas tiverem maior extensão, é certo que a diferença entre o tempo de execução do BLAST poderá ser muito melhor que o tempo do algoritmo comparado. Outro fator que pode ser percebido é como os sistemas LINUX trabalham melhor no gerenciamento de seus recursos, o ganho devido à mudança do sistema operacional é perceptível.

Tendo em vista o tamanho da maior cadeia analisada, de 7800 caracteres aproximados, e sabendo que este é um genoma completo, podemos definir se fizermos uma aproximação

grosseira utilizando uma PA para uma cadeia relativamente maior, no caso 1000000, nesse caso a cadeia é 129 vezes maior que a cadeia que utilizamos nos testes realizados, se multiplicarmos esse valor pelos resultados obtidos levando como media o valor de 0,08 segundos para o Smith-Waterman rodado no LINUX e 0,1 rodando no Windows, em uma cadeia 129 vezes maior esse número sobe para 10,32s contra 12,9s e tendo em vista que essas consultas são feitas inúmeras vezes em diversos genes dentro de um banco de genes gigante, é um tempo mais que considerável.

Outro ponto importante é a diferença nos tempos de execução entre os algoritmos onde o Smith-Waterman fez uma média de 0,09 segundos contra 0,06 segundos para o BLAST, temos o tempo de 11,61 segundos para o Smith-Waterman e 7,74 segundos para o BLAST, se multiplicarmos pelo fator 129, antes a diferença de tempo que parecia praticamente nula, agora já se torna algo muito maior e nem levamos em consideração que esses valores não crescem em PA e sim cresceram em PG, fazendo com que essa diferença seja maior tanto entre os algoritmos como entre os sistemas operacionais.

5.2 Trabalhos futuros

Como trabalho futuro pode ser citado o uso de cadeias maiores para realizar o alinhamento de sequências. Passando rapidamente pelo GenBank® fica evidente que os alinhamentos feitos no próprio GenBank® são feitos em quantidades absurdas e não utilizando cadeias muito maiores que as cadeias analisadas. Nesse caso o custo tempo em si tem como prioridade o alinhamento de várias cadeias de tamanhos diversos e não o alinhamento de apenas uma cadeia grande.

Outro trabalho futuro, seria uma melhor avaliação dos resultados obtidos nos ambientes contendo a GeForce® 9300M GS e a GeForce® GTS 450, para analisar os resultados considerados estranhos (Tabela 1, Tabela 2 e Tabela 4) onde os resultados obtidos com

Threads a 25% não tiveram um ganho de tempo em relação aos testes feitos de maneira diferente.

Também pode ser citada a futura avaliação dos resultados em placas de vídeo de arquitetura especifica para processamento CUDA, as GPUs Tesla alem de possuírem mais processadores CUDA do que as placas testadas, são manufaturadas de uma maneira otimizada para cálculos. Nesse caso seria interessante ver o seu desempenho em processos que demandem mais tempo, como aplicações para determinar o resultado de sistemas grandes ou até mesmo para previsão do tempo. Vale ressaltar que o ganho da programação em GPUs em si não é na quantidade de cálculos por medida de tempo que elas realizam e sim a quantidade de energia que cada *core* consome se comparado a uma CPU que possui a mesma potência.

Referencias :

[GOOD,2005] GOODNIGHT, N.; WANG, R.; HUMPHREYS, G. Computation on programmable graphics hardware. IEEE Computer Graphics and Applications, v. 25, n. 5, p. 12–15, 2005

.

[OHKS,2004] OH, K.-S.-S.; JUNG, K. Gpu implementation of neural networks. Pattern Recognition, v. 37, n. 6, p. 1311 – 1314, 2004.

[TARA,2009] TARABALKA, Y. et al. Real-time anomaly detection in hyperspectral images using multivariate normal mixture models and gpu processing. Journal of Real-Time Image Processing, v. 4,n. 3, p. 287–300, 2009.

[ADAM,2007] ADAMS, S.; PAYNE, J.; BOPPANA, R. Finite difference time domain (fdtd) simulations using graphics processors. In: DoD High Performance Computing Modernization Program Users Group Conference. [S.l.: s.n.], 2007. p. 334–338.

[SCHA,2007] SCHATZ, M. et al. High-throughput sequence alignment using graphics processing units. BMC Bioinformatics, v. 8, n. 1, p. 474, 2007.

[WEIG,2007] WEIGUO, L. et al. Streaming algorithms for biological sequence alignment on gpus. IEEE Transactions on Parallel and Distributed Systems, v. 18, n. 9, p. 1270 –1281, 2007.

[LIUY,2009] LIU, Y.; SCHMIDT, B.; MASKELL, D. Parallel reconstruction of neighbor-joining trees for large multiple sequence alignments using cuda. In: 2009 IEEE International Symposium on Parallel Distributed Processing. [S.l.: s.n.], 2009. v. 1, p. 1–8.

[TRAP,2009] TRAPNELL, C.; SCHATZ, M. C. Optimizing data intensive gpgpu computations for DNA sequence alignment. Parallel Comput., v. 35, p. 429–440, 2009.

[LIUW,2007] LIU, W.; MULLER-WITTIG, W. Performance analysis of general-purpose computation on commodity graphics hardware: A case study using bioinformatics. J. of VLSI Signal Processing, v. 48, p. 209–221, 2007.

[KHAJ,2010] KHAJEH-SAEED, A.; POOLE, S.; PEROT, J. B. Acceleration of the smith-waterman algorithm using single and multiple graphics processors. J. Comput. Phys., v. 229, p. 4247–4258,2010.

[MAR1,2009] MARUCCI, E. Paralelização da ferramenta de alinhamento de sequências MUSCLE para um ambiente distribuído. Dissertação (Mestrado) — UNESP, 2009.

[MAR2,2010] MARUCCI, E. et al. Finding fractional identities in multiple sequences using a fast parallel algorithm. In: Anais da ERAD-2010. [S.l.: s.n.], 2010.

[NAVA,2001] NAVARRO, Gonzalo. A Guided Tour to Approximate String Matching. University of Chile. ACM Computing Surveys, Vol. 33, No. 1, March 2001, pp. 31-88.

[CRUZ,2003] CRUZ, Por Leonardo M. BLAST Teoria e Prática. Curitiba, 2003.

[WATE,1976] Waterman,M.S., Smith,T.F. and Beyer,W.A. Some biological sequence metrics. Adv. Math., 20, 367±387,1976.

[WANG,1994] Wang,L. and Jiang,T. On the complexity of multiple sequence alignment. J. Comput. Biol., 1, 337±348,1994.

[KIMU,1983] Kimura,M. The Neutral Theory of Molecular Evolution. Cambridge University Press.1983.

[SMIT,1980] Smith, Temple F.; and Waterman, Michael S. . Identification of Common Molecular Subsequences. Journal of Molecular Biology 147: 195–197,1980.

[MEID,1997] MEIDANIS, João, SETUBAL, João Carlos. Introduction To Computational Molecular Biology. 1.ed. [s,l] : IE-Thomson, 1997.

[DAVI,2007] David, L. and Humphreys, G. How GPUs Work , IEEE Computer,February, 126-130, 2007.