Pós-Graduação em Ciência da Computação

"UMA PLATAFORMA HÍBRIDA BASEADA EM FPGA PARA A ACELERAÇÃO DE UM ALGORITMO DE ALINHAMENTO DE SEQUÊNCIAS BIOLÓGICAS"

Por

LUIZ HENRIQUE ALVES FIGUEIRÔA

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE/2015

Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

LUIZ HENRIQUE ALVES FIGUEIRÔA

"UMA PLATAFORMA HÍBRIDA BASEADA EM FPGA PARA A ACELERAÇÃO DE UM ALGORITMO DE ALINHAMENTO DE SEQUÊNCIAS BIOLÓGICAS"

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Peranambuco como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração:

Engenharia da Computação

Orientador: Prof. Dr.

Manoel Eusébio de Lima

RECIFE

2015

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

F475p Figueirôa, Luiz Henrique Alves
Uma plataforma híbrida baseada em FPGA para a aceleração de um algoritmo de alinhamento de sequências biológicas / Luiz Henrique Alves Figueirôa. – Recife: O Autor, 2015.
128 f.: il. fig., tab.

Orientador: Manoel Eusébio de Lima.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da computação, 2015.
Inclui referências, glossário e apêndice.

1. Engenharia da Computação. 2. FPGA. 3. Computação de alto desempenho. 4. GPU. I. Lima, Manoel Eusébio de (orientador). II. Título.

621.39 CDD (23. ed.) UFPE- MEI 2015-151

Dissertação de Mestrado apresentada por ***LUIZ HENRIQUE ALVES FIGUEIRÔA*** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título ***"UMA PLATAFORMA HÍBRIDA BASEADA EM FPGA PARA A ACELERAÇÃO DE UM ALGORITMO DE ALINHAMENTO DE SEQUÊNCIAS BIOLÓGICAS"*** orientada pelo ***Prof. MANOEL EUSÉBIO DE LIMA*** e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. Dr. Abel Guilhermino da Silva Filho
Centro de Informática / UFPE

_____

Dr. Carlos Henrique Madeiros Castelletti
Instituto Agronômico de Pernambuco (IPA)

_____

Prof. Dr. Manoel Eusébio de Lima(Orientador)
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 17 de agosto de 2015.

_____

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

AGRADECIMENTOS

Primeiramente, agradeço a Deus por ter me dado saúde e determinação para superar todos os obstáculos e perseguir os meus objetivos. Agradeço aos meus pais por todo o carinho, sacrifício e dedicação a mim e aos meus irmãos nos proporcionando uma formação digna. Agradeço à minha esposa Rosana e a minha filha Mariana pelo apoio e paciência por ter compreendido as minhas eventuais ausências. Agradeço ao meu amigo prof. Manoel Eusébio por todo apoio, orientações e ter acreditado no meu potencial. Agradeço ao meu amigo prof. Remy Eskinazi por todo o incentivo e orientação ao longo dessa tragetória. E agradeço a todo o pessoal do HPCin/Sísmica pelo apoio: Vitor Medeiros, Abner Barros, Severino José, Bruno Pessoa, Antonius Pyetro, Rodrigo Camarotti, João Paulo, Lucas Torquato, Augusto Benvenutto, Gilliano e Joelma França.

Agradeço a todos.

RESUMO

A partir da revelação da estrutura em dupla-hélice do DNA, em 1953, foi aberto o caminho para a compreensão dos mecanismos que codificam as instruções de construção e desenvolvimento das células dos seres vivos. A nova geração de sequenciadores (NGS) têm produzido gigantescos volumes de dados nos Bancos de Dados biológicos cujas informações podem demandar uma intensa atividade computacional em sua compilação. Entretanto, o desempenho das ferramentas empregadas na Biologia Computacional não tem evoluído na mesma taxa de crescimento desses bancos, podendo impor restrições aos avanços neste campo de pesquisa. Uma das principais técnicas usadas é o alinhamento de sequências que, a partir da identificação de similaridades, possibilitam a análise de regiões conservadas em sequências homólogas, servem como ponto de partida no estudo de estruturas secundárias de proteínas e de construção de àrvores filogenéticas, entre outros. Como os algoritmos exatos de alinhamento possuem complexidade quadrática no tempo e no espaço, o custo computacional poderá ser elevado demandando estratégias de aceleração. Neste contexto, a Computação de Alto Desempenho (HPC), estruturada em Supercomputadores e Clusters, tem sido, empregada. No entanto, o investimento inicial e os requisitos de manutenção, espaço físico, refrigeração, além do consumo de energia, podem representar custos significativos. As arquiteturas paralelas híbridas baseadas na ação conjunta de PCs e dispositivos aceleradores como chips VLSI, GPGPUs e FPGAs, surgiram como alternativas mais acessíveis, apresentando resultados promissores. O projeto descrito nesta dissertação tem por objetivo a aceleração do algoritmo de alinhamento-ótimo global, conhecido como Needleman-Wunsch, a partir de uma plataforma híbrida baseada em um PC (host) e um FPGA. A aceleração ocorre a partir da exploração das possibilidades de paralelismo oferecidas pelo algoritmo e sua implementação em hardware. A arquitetura desenvolvida é baseada num Array Sistólico Linear apresentando elevado desempenho e boa escalabilidade.

Palavras-chave : DNA. HPC. FPGA. GPGPU. Array Sistólico.

ABSTRACT

From the revelation of the structure in double-helix of Deoxyribonucleic Acid (DNA) by James D. Watson and Francis H. C. Crick, in 1953, it opened the way for the understanding of the mechanismis that encoding the building instructions and development of cells of living beings. The DNA sequencing is one of the first steps in this process. The new generation of sequencers (NGS) have produced massive amounts of data on biological databases whose information may require intense computational activity in your compilation. However, the performance of the tools employed in Computational Biology has not evolved at the same rate of growth of these banks, may impose restrictions on advances in this research field. One of the primary techniques used is the sequence alignment that from the identification of similarities, enable the analysis of conserved regions of homologous sequences, serve as the starting point in the study of protein secondary structures and the construction of phylogenetic trees, among others. As the exact alignment algorithms have quadratic complexity in time and space, the computational cost can be high demanding acceleration strategies. In this context, the High Performance Computing (HPC), structured in supercomputers and clusters, has been employed. However, the initial investment and maintenance requirements, floor space, cooling, in addition to energy consumption, may represent significant costs. The hybrid parallel architectures based on joint action of PCs and devices accelerators as VLSI chips, GPGPUs and FPGAs, have emerged as more affordable alternatives, with promising results. The project described in this dissertation aims at accelerating the global optimal-alignment algorithm, known as Needleman-Wunsch, from a hybrid platform based on a PC, that acts as host, and an FPGA. The acceleration occurs through exploration of the parallelism opportunities offered by the algorithm and implemented in hardware. In this, an architecture based on a Linear Systolic Array offers high performance and high scalability.

Keywords : DNA. HPC. FPGA. GPGPU. Systolic Array.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ALM - **Adaptative Logic Module**

ASIC - **Application Specific Integrated Circuit**

BD **- Banco de Dados**

BRAM - **Blocks of Randomic Access Memory**

CI - **Circuito Integrado**

CLB - **Configurable Logic Block**

CPU - **Central Processing Unit**

CUDA - **Compute Unified Device Architecture**

DLL - **Dynamic Link Libraries**

DMA - **Direct Memory Access**

DSP - **Digital Signal Processor**

FIFO - **First In, First Out**

FPGA - **Field Programmable Gate Array**

FSM - **Finite State Machine**

GPGPU - **General Purpose Graphics Processing Unit**

GUI - **Graphical User Interface**

HDL - **Hardware Description Language**

HPC - **High Performance Computing**

JTAG - **Joint Test Action Group**

LAB - **Logic Array Block**

LUT - **Look-up Table**

NCBI - **National Center for Biotechnology Information**

PC - **Personal Computer**

PCB - **Printed Circuit Board**

PE - **Processing Element**

RNA - **Ribonucleic Acid**

SIMD **- Single Instruction, Multiple Data**

SoC - **System-on-Chip**

SSE - **Streaming SIMD Extensions**

USB - **Universal Serial Bus**

VLSI - **Very Large-Scale of Integration**

SUMÁRIO

1 INTRODUÇÃO

Desde a descoberta do DNA, em 1869, pelo bioquímico suiço Johann Friedrich Miescher, passando pelo modelo estrutural em forma de uma dupla-hélice ao redor de um eixo imaginário proposto, em 1953, por Watson e Crick (WATSON,1953) até o desenvolvimento do primeiro método de sequenciamento do DNA, estabelecido por Ray Wu, em 1970 na Universidade de Cornell, grandes avanços foram obtidos, incluindo o desenvolvimento de novas áreas de pesquisa científica, como a genética.Entre estes avanços deve ser registrado o sucesso do Projeto do Genoma Humano (VENTER,2001), iniciado em 1990 e concluído em 2003, fruto de um esforço internacional de diversos países com objetivo de elaborar o sequenciamento completo do DNA Humano e o respectivo mapeamento de todos os genes.

Podemos afirmar que a busca pelas informações contidas no DNA se popularizou com o advento dos primeiros exames de paternidade, denominados "testes de DNA". E, mais recentemente, quando celebridades autorizaram pesquisas em seus códigos genéticos para a identificação de fatores genéticos que potencializem o acometimento futuro de doenças graves, particularmente, o câncer. Nestes dois exemplos, o sequenciamento do DNA seguido por um Alinhamento de Sequências são essenciais. De fato, o sequenciamento gera a base-de-dados; no entanto, o resgate das informações só é possível a partir de algoritmos de pesquisa, dos quais, destacamos o Alinhamento de Sequências.

O sequenciamento do DNA já é indispensável em diversas áreas de pesquisas biológicas, tais como, na engenharia farmaceutica, na biotecnologia, na medicina genômica, na biologia forense, entre outras. Destas, vamos destacar duas àreas: A biologia forense e a medicina genômica.

A biologia forense, entre outros objetivos, dedica-se ao desenvolvimento de técnicas que sirvam de auxílio nas investigações criminais e na identificação de suspeitos utilizando, por exemplo, bancos de DNA com cadastro de criminosos. O

mais conhecido deles, é o NDIS (National DNA Index System), o qual tem como principal usuário, o FBI (Federal Bureau of Investigation).

Uma outra área a ser destacada, e em franca evolução, é a da medicina genômica que a partir do sequenciamento do genoma de um indivíduo permite a prática de uma medicina preventiva e personalizada com o objetivo de evitar ou retardar o desenvolvimento de doenças genéticas. Dessa forma, é possível vislumbrar um futuro no qual as pessoas quando forem às suas consultas médicas levarão consigo um *pen-drive*, ou algum outro dispositivo de armazenamento, contendo todas as suas informações genéticas. Assim, o especialista, não só poderá agir no seu problema imediato, como o faz na medicina tradicional, como poderá indicar tratamentos ou práticas preventivas para doenças para as quais o seu histórico genético aponta predisposição (SCALIONE,2015).

As metodologias de sequenciamento estão em contínua evolução, notadamente, a partir do início dos anos 90 com o advento da chamada Next-Generation Sequency (NGS). Como resultado, grandes volumes de dados têm sido gerados e depositados em Bancos de Dados Biológicos (BDBs, a partir deste ponto), como o GenBank do *NCBI (National Center for Biotechnology Information)* (NCBI,2015) e o banco de proteínas *UniProt* (*Universal Protein Resource)* (UNIPROT,2015).

1.1 Motivação

Do ponto de vista computacional, a macromolécula do DNA pode ser tratada como uma *string* composta por quatro tipos básicos de caracteres, onde cada um destes representa um tipo de nucleotídeo. Por sua vez, a Bioinformática dedica-se à análise e ao gerenciamento de informações biológicas aplicando conceitos da ciência da computação, utilizando a matemática e a estatística como ferramentas (LUSCOMBE,2001). Um dos principais atuais desafios da Bioinformática tem sido o processamento de toda esta massa de dados armazenada nos BDBs, para que sejam geradas informações relevantes às pesquisas biológicas. No entanto, como a análise desses dados, majoritariamente, se dá através de comparações entre enormes quantidades de sequências de DNA, RNA ou de proteínas, ela depende de algoritmos eficientes e que sejam executadas em plataformas computacionais, preferencialmente, com alto desempenho. Um modo de comparação entre sequências é através do seu Alinhamento que, intuitivamente, é um processo no

qual uma sequência é colocada sobre a outra de modo a ressaltar as similaridades/diferenças (DURBIN,1998).

Os Alinhamentos de sequências podem ser executados a partir de algoritmos heurísticos ou de algoritmos exatos. Nos algoritmos heurísticos, apesar de não oferecer garantias que o resultado ótimo foi obtido, tem como principal vantagem um menor tempo de resposta. Já nos algoritmos exatos, obtém-se o resultado ótimo o que, do ponto de vista biológico, é bastante relevante; no entanto, dependendo do tamanho das sequências, da base-de-dados pesquisada e dos recursos computacionais disponíveis, o tempo de resposta poderá se tornar impraticável. Uma solução que tem sido adotada, neste caso, tem sido através do uso de arquiteturas que, explorando as possibilidades de paralelismo, acelerem a execução desses algoritmos.

A Computação de Alto Desempenho *(High-Performance Computing-HPC)* tem sido usada na aceleração dos algoritmos exatos de alinhamento, seja através de *Clusters* de *Workstations* (SHRIMANKAR,2012), *Clusters* de *FPGAs* (GEORGE,2011; WIENBRANDT,2013), ou *Clusters* de *GPUs* (TRUONG,2014; LI,2013). As extensões *SIMD* dos processadores que compõem os *multicores*, como as extensões *SSE*, da Intel Corp. (INTEL,2015), também têm sido usadas nesta aceleração(ROGNES,2011).

Os *FPGAs* e as *GPUs* têm sido largamente empregados na aceleração de algoritmos de alinhamento de sequências biológicas (GEORGE,2011; LI,2013; SAVRAN,2014; SOTIRIADES,2006). Os *FPGAs* disponibilizam uma arquitetura reconfigurável que pode ser customizada para uma aplicação específica e, por consequencia, podem oferecer um desempenho superior com um menor consumo de energia; enquanto que, as *GPUs*, comparativamente, apresentam menores custos e, por permitir um nível mais alto de programação, possibilitam um maior produtividade nos projetos (BENKRID,2012).

Progressivos investimentos tecnológicos têm sido aplicados pelos fabricantes no sentido de dotar os *FPGAs* com cada vez mais recursos embarcados; inclusive, possibilitando a sua programação em um nível de abstração bem mais elevado que as linguagens *HDL (Hardware Description Language)*; como, por exemplo, a linguagem *OpenCL* (OPENCL,2015), à qual, é baseada em C e contém extensões que permitem a especificação de paralelismo.

Assim, a principal motivação para este trabalho está na proposição de uma plataforma que acelere o processo de alinhamento de sequências de DNA através do uso de um dispositivo reconfigurável, especificamente, um FPGA. A escolha do algoritmo de Alinhamento Global Needleman-Wunsch deveu-se a possibilidade de seu uso tanto em aplicações forenses como em outras áreas de pesquisas de doenças de natureza genética.

Assim, pegando o conceito, anteriormente, apresentado referente à medicina genômica, uma placa aceleradora baseada em um FPGA poderia ser instalada em qualquer PC convencional de um consultório médico, por exemplo.

1.2 Objetivos

O principal objetivo desta dissertação é o de apresentar o projeto de uma implementação do algoritmo Needleman-Wunsch focando o Alinhamento Global de Sequências Biológicas (DNA, RNA ou Proteínas) a partir de uma plataforma híbrida formada por um *PC (host)* e uma placa aceleradora baseada em um *FPGA*. A arquitetura parametrizável constituída por diversas Unidades de Alinhamento, que atuam em paralelo, será apresentada em detalhes. Cada uma destas Unidades têm como núcleo um conjunto de blocos de processamento, denominados *PEs (Processor Elements)*, estruturados em um *Array Sistólico Linear*.

Importante ressaltar que o foco deste projeto foi, exclusivamente, voltado para maximização de desempenho. Em assim sendo, nenhuma consideração foi feita quanto a dissipação de potência (ou consumo de energia), ocupação de área em *chip* ou aspectos referentes a custos financeiros.

Os resultados a serem apresentados foram obtidos a partir do alinhamento com sequências extraídas de bancos de dados biológicos. O desempenho da arquitetura proposta foi comparada com o de outras implementações deste algoritmo, em outras plataformas.

1.3 Estrutura da Dissertação

O Capítulo 1 (Introdução) introduziu o tema e os objetivos do projeto a ser apresentado. A partir deste ponto, a dissertação está estruturada da seguinte forma:

- O Capítulo 2 (Fundamentação Teórica) aborda os conceitos utilizados para o desenvolvimento do projeto propostos nesta dissertação.

- No Capítulo 3 (Trabalhos Relacionados) são apresentados trabalhos relacionados ao tema, onde foi buscado, na medida do possível, o estado-da-arte.
- O Capítulo 4 (Arquitetura Proposta) apresenta todo o detalhamento do projeto desde a sua concepção, passando pelos detalhes de hardware e de controle até aspectos quanto ao desempenho teórico esperado.
- O Capítulo 5 (Resultados Experimentais) apresenta e discute os resultados experimentais coletados comparando com os apresentados no capítulo de Trabalhos Relacionados.
- O Capítulo 6 (Estudo de Caso) apresenta um problema real e de que forma este projeto poderia contribuir em sua solução.
- O Capítulo 7 (Conclusão e Trabalhos Futuros) conclui a dissertação e comenta os trabalhos futuros.
- Referências Bibliográficas.
- Glossário.
- Apêndice A.

2 FUNDAMENTAÇÃO TEÓRICA

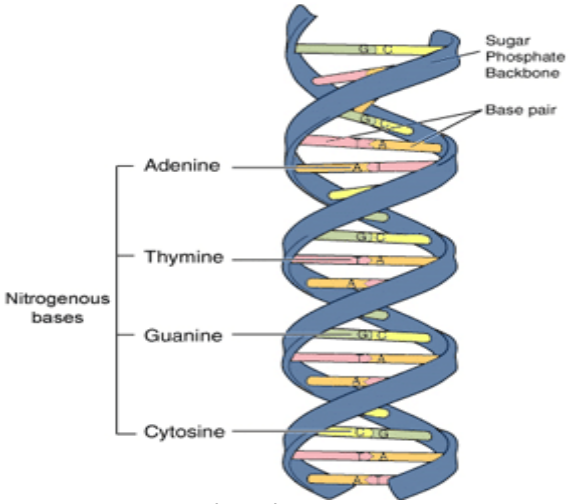
Este capítulo apresenta alguns conceitos básicos que serão relevantes para o perfeito entendimento desta dissertação.

2.1 DNA, RNA e Proteínas

Na Biologia, a célula é considerada o menor bloco de construção de um organismo, tais como, plantas, animais, bactérias, fungos, etc. Dessa forma, todos os seres vivos, desde os organismos unicelulares até o corpo humano são construídos a partir de células. No núcleo das células, encontram-se as moléculas do *DNA (Ácido Desoxirribonucleico)* nas quais estão armazenadas informações genéticas. Estas informações definem desde a cor dos olhos, o perfume de uma flor até a forma como uma bactéria infecta uma célula pulmonar (SCITABLE,2014).

A longa molécula do *DNA* é constituída por um encadeamento de moléculas menores denominadas nucleotídeos, os quais são compostos por uma base nitrogenada, um molécula de açúcar denominada desoxirribose (uma pentose com 5 átomos de carbono) e um resíduo de fosfato. Existem 4 bases nitrogenadas no *DNA*: Adenina (A), Citosina (C), Guanina (G) e a Timina (T). Ver figura 2.1.

figura 2.1 **- A estrutura em dupla-hélice do DNA**

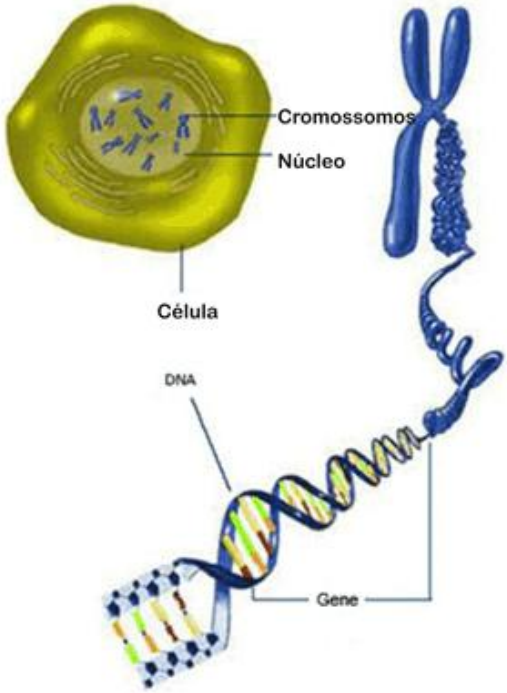


Fonte : National Human Genome Research Institute

A ordem específica na qual estas bases se organizam ao longo da cadeia é chamada de sequência do *DNA.* Na verdade, a molécula do *DNA* é formada por duas cadeias que descrevem a forma de uma hélice dupla que se entrelaçam. As duas hélices são mantidas unidas por ligações compostas por átomos de hidrogênio entre suas bases nitrogenadas, formando os pares-bases *(bp)* (GHR,2015).

Em cada célula do corpo humano existem 46 cromossomos, sendo 23 herdados do pai e 23 da mãe, compondo um total de 23 pares de cromossomos. Em cada um destes, existem trechos que guardam as informações que codificam o processo de síntese das proteínas. Estes trechos são conhecidos como *gene* e equivalem aos códigos de um programa. (SETUBAL,1997). Ver figura 2.2.

figura 2.2 **- Cromossomo e Gene**



Fonte: *http://www.sobiologia.com.br/conteudos/Citologia2/nucleo5.php*

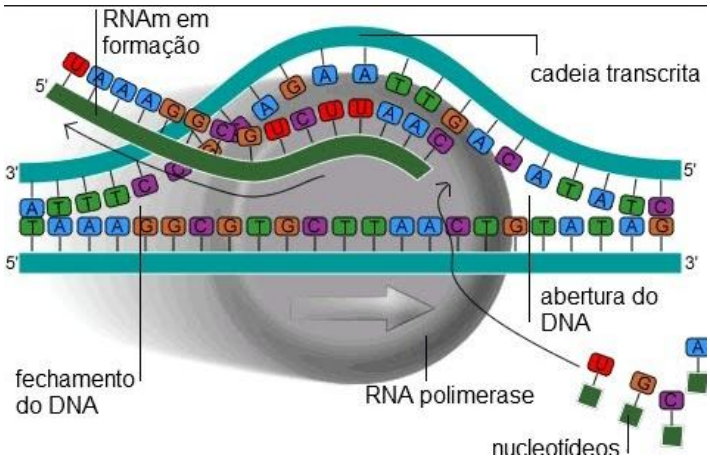
Genoma é a denominação dada ao conjunto de cromossomos contendo as regiões dos genes e as regiões não-codificantes. O tamanho do Genoma varia de organismo para organismo. No genoma humano, por exemplo, este tamanho é da ordem de 3 bilhões de *bps* (nucleotídeos) contendo entre 25 e 30 mil genes (VENTER,2001).

O processo de síntese das proteínas é iniciado a partir da geração do *RNA (Ácido Ribonucleico)*. Este Ácido Nucleico é semelhante ao *DNA* com, basicamente, 4 diferenças: Sua estrutura é composta por uma única cadeia, o açúcar é a *ribose,* a

base nitrogenada Timina (T) é substituída pela base Uracil (U) e existem 3 tipos de *RNA* com diferentes funções (SETUBAL,1997).

A geração do *RNA Mensageiro (mRNA)* se dá através de um processo denominado *transcrição*. Neste processo, uma enzima, que é um tipo de proteína, chamada *RNA polimerase*, provoca a abertura do *DNA* num trecho correspondente a um gene. À medida em que o *RNA polimerase* avança neste trecho, é produzida uma cópia de uma das hélices do *DNA*, gerando o *mRNA*, até que o trecho correspondente ao gene seja encerrado. Ver figura 2.3.

figura 2.3 - **Processo da Transcrição**



Fonte: *http://www.teliga.net/2010/08/aspectos-gerais-da-sintese-de-proteinas.html*

A transcrição ocorre no núcleo celular. A última fase do processo de geração das proteínas ocorre no *citoplasma*, que é a região interna da célula entre o núcleo e a membrana celular. O *mRNA* transcrito sai do núcleo e passa a interagir com uma estrutura complexa denominada *ribossomo*. Num processo denominado *Tradução*, o ribossomo faz a leitura da sequência do *mRNA* e a cada 3 bases lidas, conhecidas como *códons*, é codificado um *aminoácido*, que são os blocos básicos construtores das proteínas. Existem 20 tipos de aminoácidos.

As proteínas são responsáveis por diversas funções dentro das células, entre elas, as estruturais e as metabólicas. Como função estrutural, pode-se afirmar que, as proteínas são responsáveis pelo formato da célula, sua locomoção e pelo transporte intracelular de substancias (ZAHA,2003). Como exemplos, podem ser citadas: a hemoglobina, que é responsável pelo transporte de oxigênio através do sangue e a insulina que é um hormônio produzido no pâncreas e é responsável pela entrada da glicose nas células. Como função metabólica, pode-se citar as enzimas,

que possuem uma ação catalizadora nos processos bioquímicos que ocorrem dentro da célula como, por exemplo, a citada *RNA polimerase* (SETUBAL,1997).

2.2 Vírus e Vírus-Satélites

Vírus são parasitas no nível molecular. Em sua maioria são constituídos por uma capa proteíca *(capsídeo)* com algum material genético em seu inetrior, seja *DNA* ou *RNA*. Não apresentam nenhuma forma de *metabolismo*; ou seja, não ocorrem reações bioquímicas em seu interior. No entanto, eles podem se reproduzir ao infectarem células específicas denominadas hospedeiras *(hosts)*, fazendo uso de seu metabolismo. Quando uma célula é infectada por um vírus, o material genético deste é introduzido no *citoplasma* da mesma, passando a ser então, erroneamente, interpretado pelos mecanismos da célula como sendo o do seu próprio *DNA*. A partir daí, a célula passa a produzir as proteínas codificadas pelo vírus como se fossem às suas próprias. Neste processo de replicação, novos *capsídeos* são formados e novas partículas do vírus são construídas no interior da célula infectada até que a membrana celular seja rompida e tais partículas passem a atacar outras células (SETUBAL,1997).

No caso das plantas, mais de 90% dos vírus que as atacam possuem, apenas, *RNA* como o seu material genético. Nas hortaliças, por exemplo, considerando a importância econômica de suas culturas, entre os principais grupos de vírus que as infectam, podem ser destacados os *potyvírus* (família *Potyviridae*), os *tospovírus* (família *Bunyaviridae*), os *geminivírus* (família *Geminiviridae*, principalmente o gênero *Begomovirus*) e os *tobamovírus* (sem classificação de família). Destes, apenas, os *geminivírus* possuem DNA como material genético. Os *tobamovírus* são disseminados entre as plantas por meio dos tratos culturais; enquanto que, os demais gêneros são transmitidos por insetos, tais como, os pulgões, os tripes e a mosca branca (LIMA,2015).

Vírus-Satélites são agentes sub-virais que dependem da presença de um virus *(helper vírus)* para a sua propagação. Sua estrutura interna é composta por algum àcido-nucleico genômico que codifica uma estrutura proteíca que promove o seu encapsulamento. Um determinado vírus-satélite pode estar associado a diversos tipos de vírus. Alguns vírus-satelites podem agravar os sintomas das doenças causadas pelo *helper virus* ou, até mesmo, produzir novos sintomas não relacionados ao seu hospedeiro(BRIDDON,2003)

Os vírus-satélites podem infectar plantas, animais e bactérias. Apesar de, em sua maioria, estarem relacionados a vírus de plantas, existem exceções como o caso do vírus da hepatite D que, na verdade, é um vírus-satélite e que está associado ao vírus humano da hepatite B. Assim, as infecções causadas pelo vírus-satélite da hepatite D só ocorrem em indivíduos que já tenha o vírus da hepatite B. No mundo, isto representa, segundo estimativas, 5% das 350 milhões de pessoas portadoras do vírus da hepatite B. A ação conjunta destes dois vírus causam o agravamento dos quadros infecciosos da doença, podendo acelerar os processos que levam à cirrose hepática ou, até mesmo, ao óbito (RACANIELLO,2015).

Em nosso estudo de caso é abordada uma doença, de natureza viral, que afeta as plantações de algodão e de que forma a utilização de uma plataforma de alto desempenho poderia contribuir na busca de uma solução para o problema.

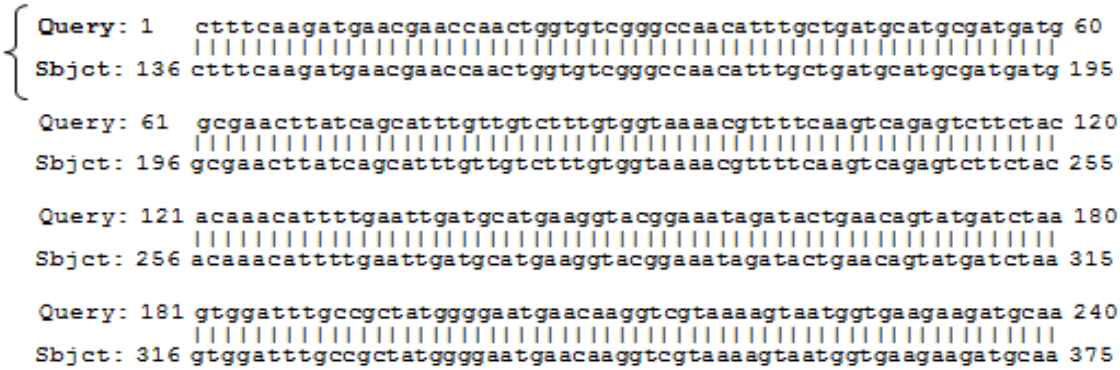
2.3 Alinhamento de Sequências Biológicas

Durante o processo evolutivo dos organismos vivos, alterações em seu material genético poderão ocorrer. Estas alterações podem ser causadas, por exemplo, por mutações provocadas por agentes externos (ação de um virús, exposição à radiações, etc.) ou por erros gerados durante a ocorrência de algum processo celular, tais como, a replicação do *DNA* ou sua transcrição. Tais mutações provocam a substituição, a inserção ou a remoção de bases*(bps*) ao longo da sequência do *DNA* (MEYERS,1995). Estas alterações são permanentes e poderão alterar as características e as funcionalidades das células e das proteínas geradas.

Na Bioinformática existem ferramentas especializadas como o Alinhamento de Sequências que são utilizados com o objetivo de determinar o grau de similaridade entre duas sequências biológicas distintas; sejam elas, sequências de *DNA,* de *RNA* ou de proteínas. A partir desta comparação torna-se possível, por exemplo, a inferência de homologias e a identificação de regiões conservadas ou que tenham sofrido algum tipo de alteração. Desta forma, informações de ordem funcional, estrutural, evolucionária, entre outras, podem ser geradas. (DURBIN,1998).

Os Alinhamentos de Sequências podem ser classificados como simples ou múltiplos. No Alinhamento Simples ou de pares de sequências, uma sequência é comparada a outra. Ver figura 2.4.

figura 2.4 **- Alinhamento Simples entre duas sequencias.**

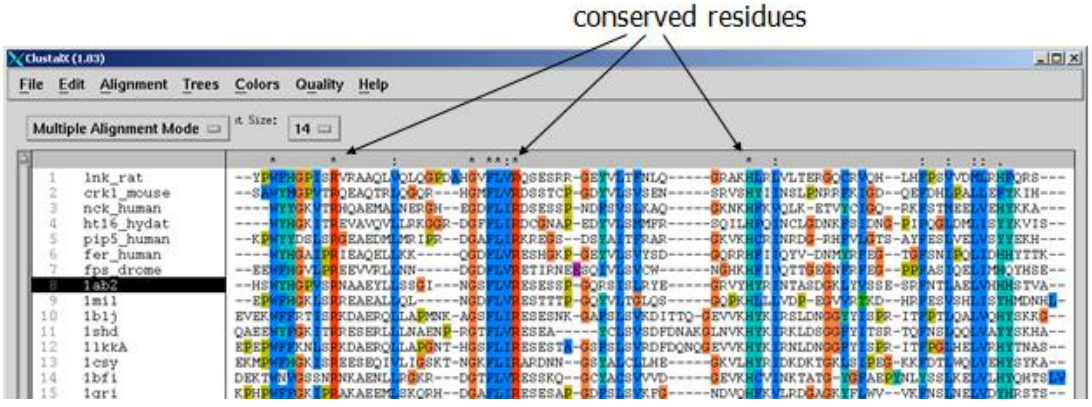


fonte: PROSDOCIMI,2015

Alinhamentos Simples ocorrem, por exemplo, quando uma sequência desconhecida, denominada como *"Query"*, é, individualmente, comparada a cada uma das sequências pertencentes a um determinado banco, buscando-se identicar similaridades com uma ou mais sequências pertencentes a este. Neste caso, a operação é chamada de múltiplo alinhamento de pares de sequências.

Por sua vez, no Alinhamento Múltiplo, três ou mais sequências sofrem um alinhamento simultâneo. Neste caso, poderá estar sendo buscada uma relação evolutiva entre as sequências, através da qual, estaria sendo compartilhada uma linhagem e a descendência com um ancestral comum. Um outro exemplo ocorre quando um determinado grupo de sequências de proteínas possuem funções similares para um determinado número de diferentes espécies. Neste caso, busca-se identificar que regiões destas sequências são similares e quais são distintas (SETUBAL,1997). Ver figura 2.5.

figura 2.5 **- Alinhamento Múltiplo de 15 sequencias**



fonte : ***http://www2.bioqmed.ufrj.br/prosdocimi/courses/bioinfo/***

O Algoritmo de Alinhamento busca descobrir qual é o menor número possível de operações de inserção, substituição ou de remoção de suas bases que poderiam transformar uma sequência em outra. Para facilitar a visualização das inserções e remoções, espaços vazios *(*denominados *gaps)* são inseridos entre as sequências. Em seguida, escores são atribuídos a cada um destes possíveis eventos : coincidencia de bases *(match)*, divergencia de bases *(mismatch),* inserção e remoção. Os valores a serem atribuídos a estes escores irão depender dos objetivos desejados. Ao final, estes escores deverão ser somados para que se obtenha a pontuação total do Alinhamento realizado. Em geral, quanto maior for esta pontuação, maior será o grau de similaridade entre as sequências. O Alinhamento é dito ótimo, se ele minimizar o número de operações necessárias para transformar uma sequência em outra (FISCHETTI,1992).

A figura 2.6 exemplifica um Alinhamento entre as seguintes sequências de DNA: *{CCATGT}* e *{CATAGCTC}*. Foram estabelecidos os seguinte escores: *matches* (+1), *mismatches* (-1) e *gaps* (-2). No final, a pontuação total é calculada.

figura 2.6 **- Exemplo de um Alinhamento**

C	C	A	T	_	_	G	T	_	
_	C	A	T	A	G	C	T	C	
-2	+1	+1	+1	-2	-2	-1	+1	-2	= -5

Fonte: Elaborada pelo autor

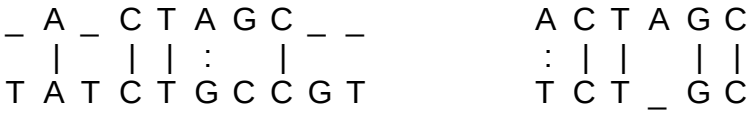
2.3.1 Classificação dos Alinhamentos

O Alinhamento de Sequências Biológicos são classificados em Alinhamentos Simples e Múltiplos, como já mencionado. Por sua vez, os Alinhamentos Simples podem ser sub-divididos em dois tipos: Alinhamento Global e Alinhamento Local.

O Alinhamento Simples apresentado na figura 2.6 é classificado como um Alinhamento Global, uma vez que as duas sequências são alinhadas como um todo e todos as bases participam do resultado final. Por esta razão, sequências similares com tamanhos e estruturas semelhantes são boas candidatas ao Alinhamento Global. Por sua vez, no Alinhamento Local busca-se, específicamente, por trechos ou segmentos com alta similaridade entre as sequências. Assim, este tipo de Alinhamento é mais adequado aos casos em que as sequências possuem regiões com alta similaridade e que não se encontrem na mesma posição em ambas.

A figura 2.7 apresenta um exemplo simples destes dois tipos de Alinhamento para as sequências *A = {ACTAGC}* e *B = {TATCTGCCGT}.*

figura 2.7 **- Diferenças entre os Alinhamentos: Global e Local**



a) Alinhamento Global **b) Alinhamento Local**

Fonte: Elaborada pelo autor

Em ambos os casos, encontrar o Alinhamento ótimo, que propicia a maior pontuação, não é uma tarefa trivial. Na verdade, o número de possibilidades de alinhamentos é exponencial no tamanho das sequências (n^n). Dessa forma, considerando sequências maiores, o número de possibilidades poderá ser gigantesco e, consequentemente, os tempos de busca poderão ser impraticáveis. Em função de sua forma de implementação, os algoritmos de Alinhamento, também, são classificados como Exatos ou Heurísticos. No primeiro caso, o algoritmo irá gerar um resultado ótimo; enquanto que, no segundo caso, heurísticas serão usadas com o objetivo de produzir um resultado o mais próximo possível do ótimo, porém, de uma forma muito mais rápida. Dentre os algoritmos heurísticos destacam-se, o Blast (ALTSCHUL,1997) e suas versões, o Fasta (PEARSON,1998) e o ClustalW (CLUSTALW,2015) são os mais utilizados (VIANA,2010).

Na busca pelo Alinhamento ótimo, os Algoritmos Exatos mais utilizados são: o Needleman-Wunsch (alinhamento global) e o Smith-Waterman (alinhamento local).

2.3.2 Alinhamento Global (Needleman-Wunsch)

Em (NEEDLEMAN,1970), Needleman e Wunsch propuseram um Algoritmo Exato, baseado em Programação Dinâmica, como solução para o Alinhamento ótimo Global entre duas sequências.

A técnica de Programação Dinâmica consiste em resolver uma determinada instância de um problema utilizando as soluções já computadas para instancias menores deste mesmo problema (SETUBAL,1997). Dessa forma, problemas complexos são particionados em um número razoável de sub-problemas mais

simples. A partir daí, estes sub-problemas serão resolvidos, recursivamente, o que ao final, acaba gerando a solução ótima para o problema principal.

Como o Algoritmo Needleman-Wunsch usa uma Matriz de Programação Dinâmica *(Matriz PD, a partir deste ponto)*, sua complexidade temporal e espacial são *O(mn)*, onde *m* e *n* são os respectivos comprimentos das sequências a serem alinhadas. O Algoritmo Needleman-Wunsch possui 2 fases distintas :

a) Construção da Matriz de Programação Dinâmica (ou *Matriz PD*).

b) *Traceback.*

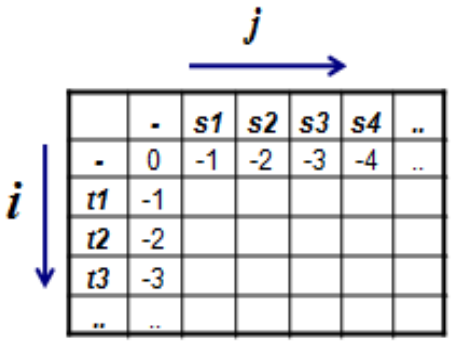
A primeira fase é iniciada a partir do preenchimento da primeira linha e da primeira coluna da *Matriz PD*. Esta inicialização segue as definições dadas pelas equações 2.1, onde *CI* e *CR* representam valores associados aos Custos de Inserção e de Remoção, respectivamente. Caso estes custos sejam iguais, eles podem ser substituídos pela letra *"g"* (custo do *gap*), conforme equação 2.2.

$$M(i,0) = CI.i \qquad M(0,j) = CR.j \qquad \text{eq. 2.1}$$

$$M(i,0) = g.i \qquad M(0,j) = g.j \qquad \text{eq. 2.2}$$

A figura 2.8, apresenta o resultado desta inicialização, considerando o Alinhamento de uma determinada Sequência ***"S"*** com uma outra ***"T"*.** Como exemplo, o custo do *gap* foi considerado igual a –1.

figura 2.8 - **Inicialização da Matriz de Programação Dinâmica**



Fonte: Elaborada pelo autor

Após a inicialização, a *Matriz PD* deverá ser preenchida com os escores parciais seguindo as relações de recorrência definidas pela equação 2.3. O Custo da Comparação *(CC)* irá depender das bases em atual comparação; caso sejam iguais, teremos o custo de um *match* e, em caso contrário, o de um *mismatch.*

$$M(i,j) = \text{máximo} \begin{cases} M(i-1, j) + CI \text{ (custo de inserção)} \\ M(i-1, j-1) + CC \text{ (custo de comparação)} \\ M(i, j-1) + CR \text{ (custo de remoção)} \end{cases} \quad \textbf{eq. 2.3}$$

A figura 2.9 apresenta o início do preenchimento da *Matriz PD* referente ao Alinhamento Global das sequências ***S*** = *{ TAGGAG }* e ***T*** = *{ TGCTAG }*. Além disto, foram estabelecidos os seguintes custos: *CI = CR = -1(custo do gap), CC* para o *match* = +1 e o *CC* para o *mismatch* = -1.

figura 2.9 - **Preenchimento da Matriz PD**

	-	***T***	***A***	***G***	***G***	***A***	***G***
-	0	-1	-2	-3	-4	-5	-6
T	-1	***1***	***0***	***-1***	***-2***		
G	-2	***0***	***0***	***1***			
C	-3	***-1***	***-1***				
T	-4	***-2***					
A	-5						
G	-6						

Fonte: Elaborada pelo autor

As setas indicadas na figura 2.9 apontam qual das 3 células anteriores, através da relação de recorrência, definiu o escore da célula atual. Estas indicações deverão ser armazenadas para posterior uso na segunda fase do Algoritmo.

Ao final do cálculo de todos os escores da matriz, a última célula abaixo e à direita, a *M(7,7)*, conterá o escore total do Alinhamento. Ver figura 2.10. Considerando que todas as relações de recorrência definidas pela equação 2.3 foram respeitadas, este escore é o maior possível para este Alinhamento.

figura 2.10- **Caminho do *Traceback***

	-	T	A	G	G	A	G
-	0	-1	-2	-3	-4	-5	-6
T	-1	1	0	-1	-2	-3	-4
G	-2	0	0	1	0	-1	-2
C	-3	-1	-1	0	0	-1	-2
T	-4	-2	-2	-1	-1	-1	-2
A	-5	-3	-1	-2	-2	0	-1
G	-6	-4	-2	0	-1	-1	1

Fonte: Elaborada pelo autor

Para construir o Alinhamento ótimo, basta descobrir a sequência de operações que levou a este resultado. Por este motivo, foi necessário o armazenamento das informações geradas pelas setas indicativas da figura 2.9. Com esta informação é possível percorrer o caminho reverso, ver figura 2.10, denominado *Traceback*, a partir da célula *M(7,7)* até a *M(1,1)*, e traçar o Alinhamento global ótimo. Ver figura 2.11.

figura 2.11 **- Resultado final do Alinhamento Global ótimo :**

```
S =  T   A   G   _   G   A   G
     |       |       :   |   |
T =  T   _   G   C   T   A   G

     +1  -1  +1  -1  -1  +1  +1   = +1
```

Fonte: Elaborada pelo autor

Um detalhe final sobre este Algoritmo é que, como podem ocorrer empate de valores nos cálculos das equações de recorrência apresentados na equação 2.3, uma escolha, nestes casos, deverá ser feita. Dessa forma, podem existir mais de um traçado para o *traceback* e, portanto, mais de um Alinhamento ótimo.

2.3.3 Alinhamento Local (Smith-Waterman)

Enquanto o Alinhamento Global executa o alinhamento das sequências inteiras, o Alinhamento Local busca, apenas, alinhar as regiões de alta similaridade entre as sequências (SETUBAL,1997).

Este Algoritmo, proposto por Smith e Waterman (WATERMAN,1981), é baseado no Algoritmo Needleman-Wunsch e foi adaptado para tratar o problema de forma local. A principal modificação em relação ao Alinhamento Global foi que, como este algoritmo busca as regiões de alta similaridade, os trechos que contenham muitas substituições, inserções ou remoções de bases, serão, automáticamente, descartados. Assim, a equação 2.3 foi alterada para não aceitar valores negativos, forçando o algoritmo a buscar aquelas regiões de interesse. Ver equação 2.4.

$$M(i,j) = \text{máximo} \begin{cases} 0 \\ M(i-1,j) + CI \text{ (custo de inserção)} \\ M(i-1,j-1) + CC \text{ (custo de comparação)} \\ M(i,j-1) + CR \text{ (custo de remoção)} \end{cases} \quad \textbf{eq. 2.4}$$

Na inicialização da matriz, como não podem existir valores negativos, a primeira linha e a primeira coluna são todas preenchidas com zeros. Além deste detalhe e da introdução de um zero na equação de recorrência, o seu *traceback*, também, difere do Needleman-Wunsch.

figura 2.12 **- Alinhamento Local**

	-	T	A	G	G	A	G
-	0	0	0	0	0	0	0
T	0	1	0	0	0	0	0
G	0	0	0	1	1	0	1
C	0	0	0	0	0	0	0
T	0	1	0	0	0	0	0
A	0	0	2	1	0	1	0
G	0	0	1	3	2	1	2

Fonte: Elaborada pelo autor

De fato, a construção do alinhamento ótimo poderá ser iniciada em qualquer local da matriz. Neste caso, é preciso localizar a célula (ou as células) com o maior escore e percorrer o caminho inverso até encontrar um escore igual a zero. Neste algoritmo, também, é possível identificar mais de um alinhamento ótimo. A figura 2.12 apresenta a *matriz PD* preenchida, considerando-se as mesmas sequências anteriores e os mesmos valores para os custos.

A figura 2.12, também, apresenta o *traceback* que identificou, apenas, um único alinhamento local ótimo, no caso:

S = T A G
| | |
T = T A G

Ou seja, foi identificado que o trecho TAG, comum às sequências, é o que apresenta maior grau de similaridade local entre elas.

2.3.4 Penalização dos *gaps*

Como pode ser observado, os Algoritmos Exatos de Alinhamento apresentados permitem o uso de *gaps*, ou de sequências consecutivas *gaps*, na busca das soluções ótimas. A inclusão de *gaps* consecutivos e de tamanhos diversos é, perfeitamente, viável na comparação de sequências biológicas, uma vez que, um único evento mutacional pode ser responsável pela remoção ou inserção de um longo trecho nestas sequências (GOTOH,1982). No entanto, as equações 2.3 e 2.4 anteriores impõem a mesma penalização, em termos de custos, tanto para os *gaps* isolados, tanto para aqueles que são inseridos de uma forma consecutiva. Este modelo de penalização é denominado *linear gap*; uma vez que, no cálculo do escore total, a penalização referente aos *gaps* é definida pela função linear :

$w_g(k) = k.g$ onde w_g é a pontuação geral dos *gaps*, k é o número total de *gaps* na expressão final e g é o custo atribuído a uma *gap.*

Uma vez que um evento mutacional tenha ocorrido, é bastante provável que ele afete todo um trecho de uma sequência biológica e, não, apenas, uma única base. Sendo assim, a penalização por um longo intervalo de *gaps* deveria ser, apenas, um pouco maior que a atribuída a um intervalo menor (ALTSCHUL,1986). Daí, uma função afim, denominada *affine gap*, foi introduzida por (GOTOH,1982) para tornar os Algoritmos de alinhamentos melhor adaptados ao uso em sequências

biológicas. Nesta nova abordagem, Gotoh propôs que o primeiro *gap*, pertencente a um intervalo de *gaps* introduzidos no alinhamento, deveria ter um custo diferenciado em relação aos *gaps* consecutivos pertencentes a este mesmo intervalo. Dessa forma, o custo dos *gaps* para um único intervalo, é definido pela seguinte expressão :

$$w_g(k) = g + k.e \qquad \text{com } w_g(0) = 0.$$

Nesta expressão, w_g é a pontuação referente ao custo dos *gaps* para cada intervalo, ***"k"*** é o número total de *gaps* neste intervalo, ***"g"*** é o custo atribuído ao primeiro *gap* do intervalo *(gap open)* e ***"e"*** é o custo atribuído aos *gaps* consecutivos *(gap extension)* do mesmo.

O uso desta função de custo para os *gaps* implica em uma nova metodologia de construção da matriz de programação dinâmica; uma vez que deverá ser feita uma distinção entre a penalização do primeiro *gap* em relação aos consecutivos, ainda, na fase de construção da matriz. Neste caso, a solução poderá exigir a construção de três matrizes (SETUBAL,1997), o que aumenta o tempo de execução do algoritmo. Como no projeto desta dissertação foi utilizado o *linear gap*, não será detalhada a metodologia de implementação em *hardware* utilizando o *affine gap.*

2.4 Computação de Alto Desempenho na Bioinformática

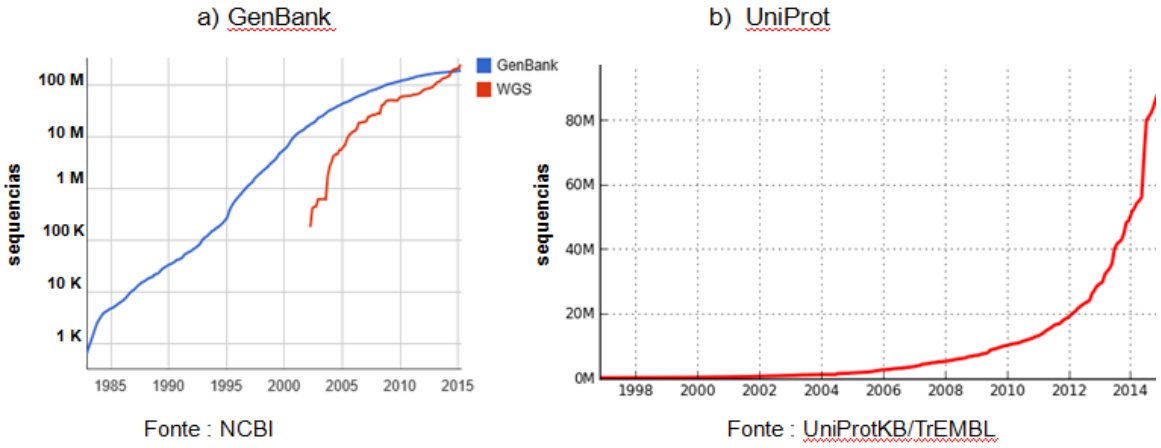
A Bioinformática pode ser definida como uma aplicação de conceitos da ciência da computação, da matemática e da estatística para a análise de dados biológicos (LUSCOMBE,2001).

Alguns problemas típicos enfrentados na Bioinformática são: a localização de genes em sequências de DNA, a análise de novas proteínas, a predição de estruturas de proteínas, a geração de árvores filogenéticas que exponham relações evolucionárias, entre outros.

Na última década, ocorreu um rápido crescimento no volume de dados armazenados nos *BDs* biológicos em função dos avanços nas técnicas de sequenciamento do *DNA* e, adicionalmente, pelo sucesso de projetos como o do Genoma Humano (VENTER,2001) e o do Genoma de Plantas (STERCK,2007). Dois exemplos, desse crescimento, são o GenBank e o UniProt, ver figura 2.13. O GenBank é um banco de sequências de nucleotídeos, incluindo sequências de

mRNA com regiões codificantes e segmentos de DNA genomicos; enquanto que, o UniProt é um banco, exclusivamente, constituído por sequências de proteínas.

figura 2.13 - **Crescimento dos Bancos-de-Dados Biológicos**

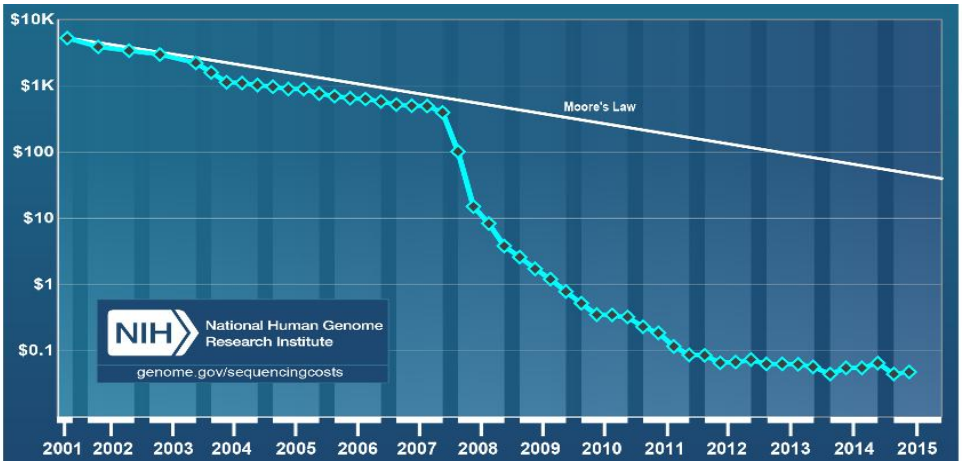


De fato, o crescimento nestes bancos deve ser atribuído, essencialmente, à nova geração de sequenciadores, denominada *NGS*, que através do uso de novas metodologias, aceleraram e baratearam o processo de sequenciamento.

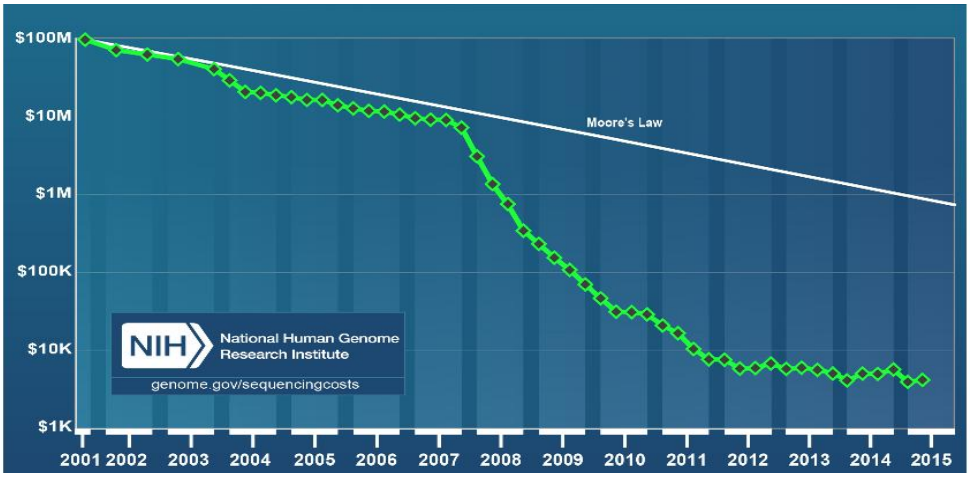
O primeiro sequenciador *NGS* comercial foi o *Massively Parallel Signature Sequencing (MPSS)*, lançado no mercado no ano 2000 pela *Lynx Therapeutics*. Estes sequenciadores se baseiam no processamento paralelo massivo de fragmentos do DNA. Enquanto um sequenciador da geração anterior era capaz de processar, no máximo, 384 fragmentos, por vez, o sequenciadores *NGS* podem ler bilhões de fragmentos, ao mesmo tempo (VARUZZA,2013).

Na medida em que os processos de sequenciamento tornaram-se mais rápidos e mais populares, o seu custo foi drásticamente reduzido conforme aponta os gráficos da figura 2.14. Pode ser observado um primeiro decréscimo, em 2004, quando ocorreu o lançamento do sequenciador 454 da Roche (ROCHE_454,2015). E, a partir de 2006 e 2007, ocorre um decréscimo ainda maior com o lançamento dos sequenciadores da nova geração da Illumina (ILLUMINA,2015) e da Life Technologies, atualmente, a Thermo Fisher Scientific Inc (THERMO,2015).

figura 2.14 - **Evolução no custo do Sequenciamento do DNA.**



a) Custo por Megabase *(Mb).*



b) Custo por Genoma.

Fonte: National Human Genome Research Institute.

O desafio dos pesquisadores passou a ser, então, o de transformar toda esta gigantesca massa de dados em conhecimento biológico, em um tempo razoável.

Paralelamente a isto, ao longo anos, as aplicações de computação de alto desempenho *HPC (High-Performance Computing)*, às quais são caracterizadas por cálculos extremamente complexos e/ou pela necessidade de extração de informações analíticas em grandes volumes de dados, têm demandado um poder de processamento superior à capacidade oferecida pelos processadores convencionais. Tais aplicações, estão presentes em áreas como meteorologia, cosmologia e astrofísica, mercado financeiro, criptografia, pesquisa petrolífera, bioinformática,

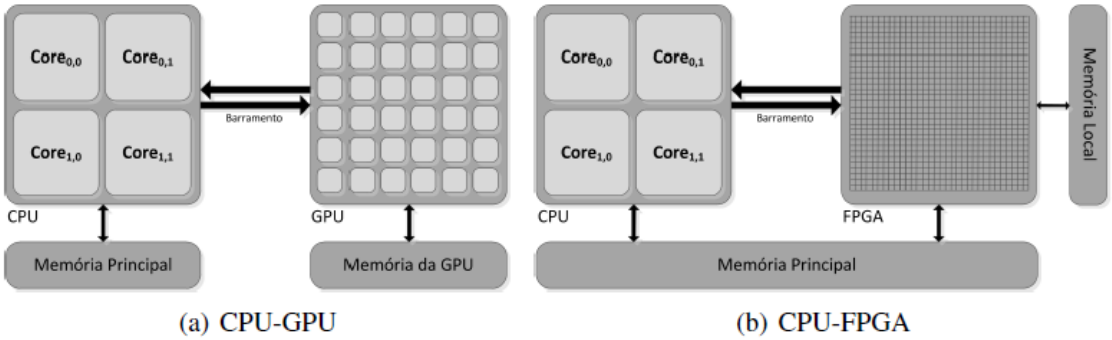
entre outras, exigindo taxas de operações da ordem de centenas de *Gigaflops*, ou superiores.

A abordagem amplamente adotada para as aplicações *HPC* tem sido através do uso da *Cluster Computing*. Nesta solução, um conjunto de computadores, distribuídos em *nós*, atuam em paralelo, executando a mesma tarefa, sob o controle e escalonamento de um único sistema operacional distribuído. Por sua vez, os computadores, em cada um desses *nós*, utilizam processadores *multi-cores*, o que aumenta, ainda mais, a capacidade total de processamento e as possibilidades de exploração do paralelismo (SCHAUER,2008).

Outra abordagem bastante adotada nas aplicações *HPC* tem sido através do uso das Arquiteturas Paralelas Híbridas. Nestas, um *hardware* dedicado, exclusivamente, à aceleração dos trechos de processamento massivo dos algoritmos atuam como co-processadores interfaceados com o processador principal *(host)* através de alguma tecnologia de comunicação. Estas estruturas aliam as unidades de processamento de propósito geral, como os processadores *multicores*, à recursos de *hardware* especializados em processamento massivo de todos em paralelo.

Os dispositivos dedicados, normalmente, utilizados nestas arquiteturas são os *ASICs*, *GPUs* ou um *hardware* reconfigurável, como por exemplo, os *FPGAs*. Como a utilização de *ASICs* só são justificadas em uma alta escala de produção devido aos elevados custos, principalmente, de projeto, e sua total ausência de flexibilidade, é predominante a utilização dos *FPGAs* e as *GPUs*, neste tipo de solução. A figura 2.15 apresenta dois exemplos destas arquiteturas.

figura 2.15 - **Arquiteturas Paralelas Híbridas**



(a) CPU-GPU (b) CPU-FPGA

Fonte: SCHAUER,2008.

Assim, a Computação de Alto Desempenho *(HPC)* passou a ser usada como solução para diversos problemas no campo da bioinformática. Implementações em processadores *multicores* (LIN,2008), em *General-Purpose Graphic Processors Units (GPGPU)* (MANAVSKI,2008), em *grids* (YANG,2009) e em plataformas reconfiguráveis, como o *FPGA* (SOTIRIADES,2006), apresentaram-se como promissoras soluções de aceleração, reduzindo, significativamente, o tempo de execução dos algoritmos, mesmo operando sobre grandes bases de dados.

2.5 Hardware Reconfigurável

As abordagens tradicionais, no tocante, às arquiteturas dos computadores fazem referencia aos Processadores de Propósito Geral *(*denominados *PPGs)* e aos dispositivos de propósito específico, os *ASICs (Application Specific Integrated Circuits)*. Os *PPGs* pertencem a uma classe de arquiteturas cujos processadores executam um conjunto genérico de instruções (PATTERSON,2000). Dessa forma, têm como principal vantagem a flexibilidade, uma vez que, a sua arquitetura interna foi projetada para executar uma infinidade de tarefas, definidas por software. No entanto, devido à esta estrutura genérica, normalmente, baseada no paradigma de *Von Neumann*, não conseguem oferecer o máximo desempenho desejado para as aplicações que demandem um processamento massivo de dados.

É neste contexto em que os *ASICs* são inseridos, uma vez que a arquitetura de seu *hardware* é construída, de forma otimizada, para atender a uma aplicação específica com alto desempenho e baixo consumo de energia. A família de dispositivos *ASICs* é classificada em sub-categorias. A categoria *ASIC (full custom)* operam em máxima velocidade e consumem uma menor área do *chip* (TOCCI,2007); no entanto, apresentam as seguintes desvantagens:

a) Alto custo de projeto, principalmente, no processo de elaborações do conjunto de máscaras, que só são justificados para uma larga escala de produção.
b) Maior tempo de projeto.
c) Total ausência de flexibilidade. Qualquer mudança, poderá implicar na re-elaboração de todo o projeto.

Por sua vez, os *ASICs PLDs (Programmable logic Devices)* é um dispositivo com características intermediárias entre os *PPGs* e os *ASICs (full custom).* Como esta categoria de dispositivos tem como principal característica um *hardware*

reconfigurável, eles combinam a flexibilidade do primeiro com o alto desempenho do segundo. Como o *hardware* reconfigurável pode ser programado para resolver um problema específico, ele poderá atingir um desempenho muito superior aos obtidos com os *PPGs* e com um maior eficiência energética. E, com a possibilidade de reprogramação, apresentam alta flexibilidade e baixíssimo custo de projeto, quando comparados aos *ASICs (full custom).*

O *FPGA* é um dos principais representantes dos *ASICs PLDs*.

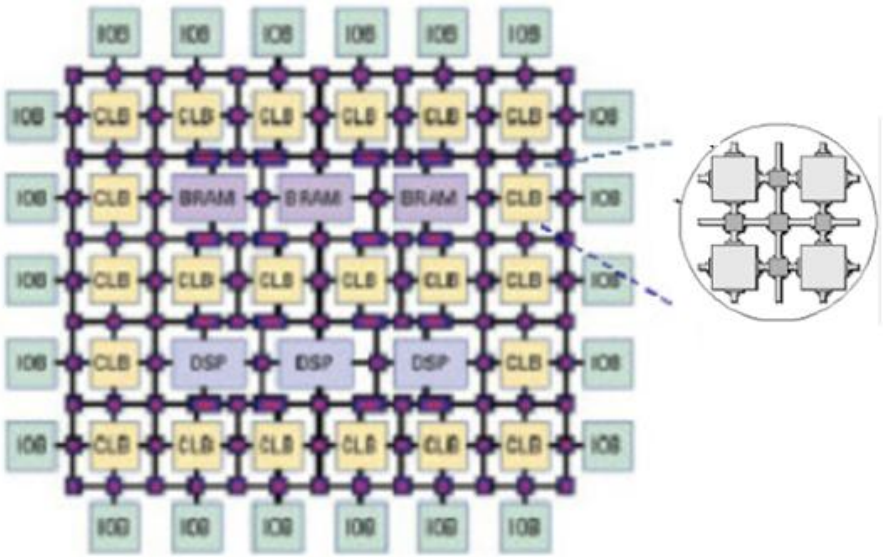
2.5.1 FPGA

Os *FPGAs* são construídos como uma matriz de blocos lógicos configuráveis e suas interconexões, internas e externas, também, são configuráveis (MAXFIELD,2004). Estes blocos podem ser configurados e interligados objetivando a construção de uma arquitetura customizada. A figura 2.16 apresenta a estrutura genérica interna de um *FPGA.* Os principais fabricantes atuais são a Altera Corporation (ALTERA,2015) e a Xilinx Inc (XILINX,2015).

Em sua arquitetura, o *FPGA* possui 3 componentes básicos:

- Os Blocos Lógicos Configuráveis *(CLBs)*.
- A matriz programável de interconexões (em detalhes, na fig. 2.15).
- Blocos de Entrada e Saída *(I/O Blocks).*

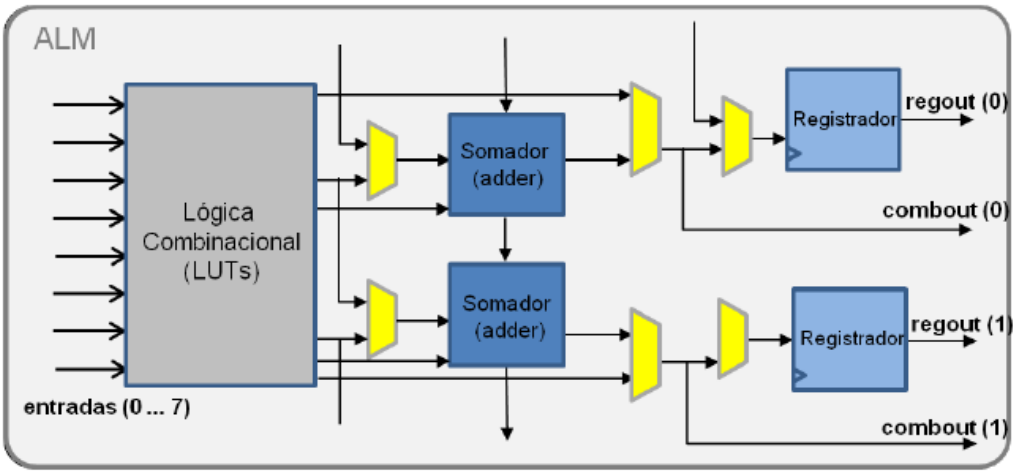
figura 2.16 **- Arquitetura interna de um FPGA**.



Fonte: Xilinx[2],2014.

Os *CLBs* são compostos, básicamente, por uma lógica combinacional, registradores (flip-flops), muliplexadores e somadores. A lógica combinacional é implementada em *look-up tables (LUTs)*, que são baseadas em memória. Na verdade, *CLB* é a designação dada pela Xilinx para estes blocos. A Altera, utiliza a sigla *LAB (Logic ArrayBlocks)*, cujos blocos são *Adaptive Logic Modules (*denominados *ALM*). A figura 2.17 apresenta a arquitetura de um *ALM*.

figura 2.17 **- Arquitetura de um ALM da Altera**



Fonte: Altera[4],2015.

A matriz programável de interconexões possibilita o *roteamento* entre os blocos. Por sua vez, os *I/O Blocks*, que encontram-se na borda da estrutura, proporcionam o acesso ao mundo externo, normalmente, com uma grande quantidade de pinos e uma elevada largura-de-banda.

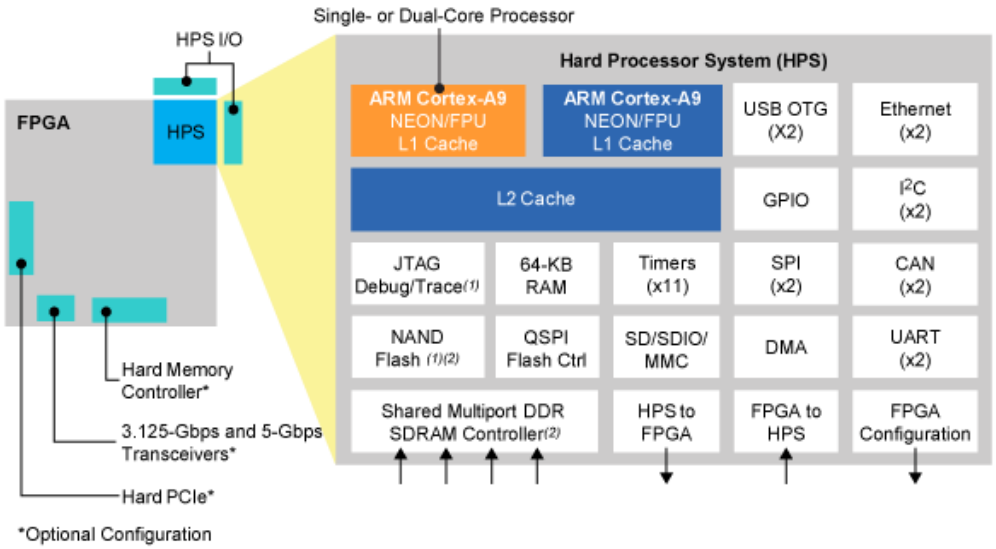
A figura 2.17, também, apresenta blocos de memória *BRAMs (Blocks of Randomic Access Memory)* que possibilitam a implementação da memória *on-chip*, de *FIFOs (First-In-First-Out)* e outros componentes de memória. A figura apresenta, ainda, blocos dedicados ao processamento digital de sinais *DSPs (Digital Signal Processors).* Na verdade, objetivando aumentar a capacidade computacional dos *FPGAs*, os fabricantes têm investido na inserção de novos blocos dedicados de *hardware* que, por esta razão, consomem uma menor área e alcançam maiores frequencias de *Clock* do que as implementações lógicas dos mesmos. Podem ser citados, como exemplo destes blocos dedicados :

- Controladores de memória.
- *Transceivers* de alta velocidade (Gigabit).
- Controladores de *PCI-express*.

- Blocos de *DSPs* e *DLLs (Dynamic Links Libraries)*.
- *Hard cores.*

Em relação a este último ítem, a Altera introduziu o *ARM* nas famílias Cyclone e Arria; enquanto que a Xilinx, já o havia feito, ao lançar no mercado o Zynq. Tomando, como exemplo, o Cyclone V da Altera, trata-se de um *SoC (System-on-chip)* que integra um *HPS(Hard Processor System)*, no caso, um *ARM Cortex-A9* dual core com seus periféricos e interfaces de memória. Ver figura 2.18.

figura 2.18 **- Arquitetura do Cyclone V destacando o ARM embarcado.**



Fonte: Altera[5],2015.

Esta união, possibilitou de forma simples o uso conjunto de todos os recursos existentes para o mundo *ARM* junto com um *FPGA*, incluindo, o Linux embarcado e, até mesmo, o Android (PRADO,2014).

Diversas linguagens são usadas no desenvolvimento de um projeto nos *FPGAs*, entre elas, as denominadas *Hardware Descripton Languages (HDLs)* como o *VHDL*, o *Verilog*, o *SystemVerilog*, entre outras. Através destas linguagens é possível implementar em *hardware,* a descrição comportamental dos sistemas. O arquivo gerado seja em código *HDL*, ou em diagrama de blocos, deve ser compilado, normalmente, através de uma ferramenta fornecida pelo fabricante do *FPGA*, gerando um *bitstream q*ue deve ser gravado em sua memória de configuração.

O problema, no entanto, está no relativo baixo nível de programação imposto por esta categoria de linguagens que, por vezes, compromete o tempo de projeto. Neste sentido, esforços têm sido demandados pelos fabricantes e pesquisadores, objetivando o desenvolvimento de linguagens com um maior nível de abstração. O

Khronos Group (KHRONOS,2015), por exemplo, é um consórcio de empresas, sem fins lucrativos, focado no desenvolvimento de padrões-abertos para a aceleração em ambientes de computação paralela. Este grupo desenvolveu o *OpenCL (Open Computing Language)*, que trata-se de uma linguagem, baseada em C, já bastante difundida em plataformas *multicores* e *GPUs.*

De um modo geral, sempre que for necessário o processamento massivo de dados de alto desempenho, o paralelismo ou as operações em tempo real, a utilização de um *FPGA* deve ser considerada (PRADO,2014).

O principal fator de comprometimento do desempenho das implementações em *FPGAs* como, também, nas *GPUs* está em suas interfaces de comunicação com a Memória *on-board* e com o *host*. A limitada Largura-de-Banda destas interfaces, em geral, estabelecem comunicações à taxas bem inferiores àquelas proporcionadas pelo processamento interno nas estruturas de *hardware* dedicado.

Além da aceleração de algoritmos na Bioinformática, os *FPGAs* vêm sendo utilizados na solução de problemas em diversas áreas, entre elas, destacam-se :

- Computação de Alto Desempenho*(HPC)*, em processamento científico.
- Setor elétrico, no processamento digital de sinais em tempo real.
- Telecomunicações, uso em *switches* e roteadores de alto desempenho.
- Multimídia, no processamento de imagens em tempo real.

2.6 GPU

As *GPUs (graphics processing unit)* são dispositivos cuja arquitetura interna é composta por uma grande quantidade de núcleos de processamento *(cores)* que operam de forma massivamente paralela, gerando desempenhos que, em determinadas aplicações, são, significativamente, superiores aos obtidos em *CPUs* convencionais. O princípio de paralelismo nas *GPUs é* baseado em *threads* que são escalonadas para os seus *cores*.

Inicialmente, as *GPUs* foram desenvolvidas com o objetivo de atender às demandas específicas da computação gráfica para o mercado de entretenimento e de jogos 3D. Tais demandas apresentavam algumas características típicas :

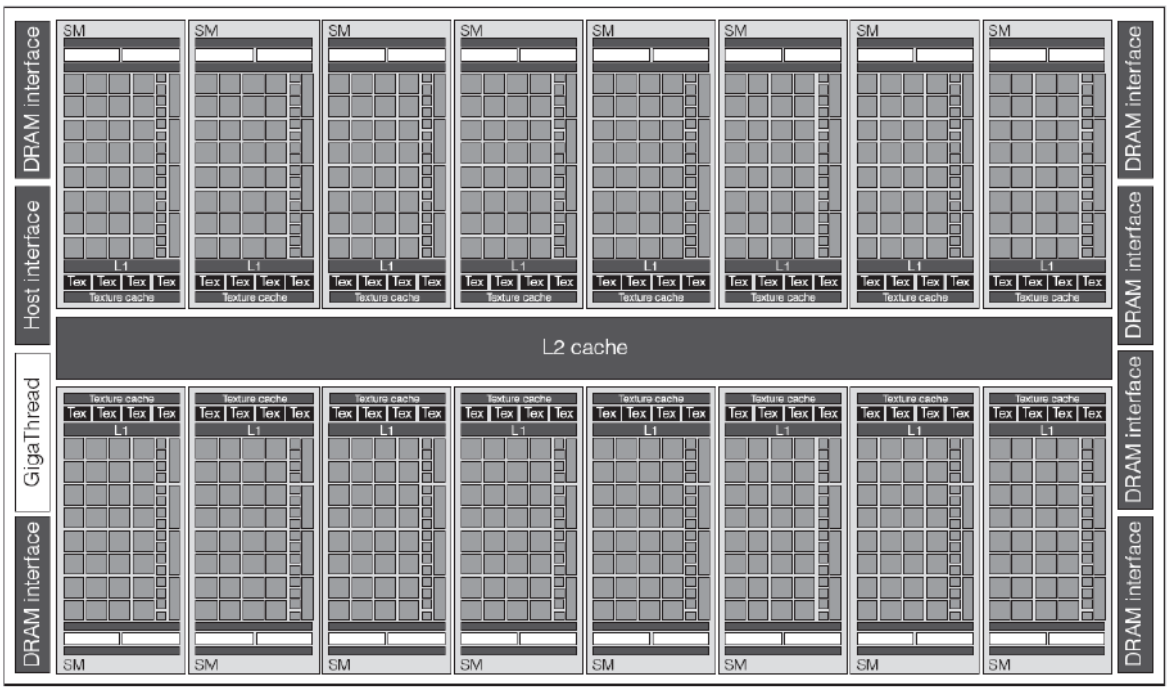
- Necessidade de grande poder computacional, em função, da renderização, em tempo real, de bilhões de *pixels* por segundo.
- Necessidade de uma arquitetura paralelizada.

- Oferecer elevado *throughput*, ainda que com alta latência, para atender a implementação do *pipeline* gráfico.

A percepção que estas características, também, estavam presentes em muitas outras aplicações, estimulou uma evolução nas *GPUs* e incentivou a sua utilização em demandas de propósito geral e não, apenas, em cálculos gráficos (OWENS,2008). Tal fato, deu origem às *GPGPUs (General Purpose Graphics Processing Unit*). Contudo, o uso de *APIs* baseadas em linguagens gráficas, como o *OpenGL*, para o mapeamento das aplicação de propósito geral, nem sempre garantiam resultados satisfatórios *(overheads)*, além de exigir uma longa curva de aprendizagem, entre outras dificuldades (PAGLIOSA,2013).

Em novembro/2006, a NVIDIA Corporation (NVIDIA,2015) lançou a plataforma de computação paralela denominada *CUDA (Compute Unified Device Architecture)* (CUDA,2015), que possibilitou a programação de suas *GPUs* através de linguagens como *C, C++, Java*, *Fortran*, entre outras. Extensões dessas linguagens são incorporadas, através de *keywords*, permitindo a exploração massiva do paralelismo nativo da arquitetura num alto nível de programação. As *GPUs* NVidia Fermi (FERMI,2009) são um bom exemplo dessa nova família de arquiteturas voltadas para aplicações de propósito geral. Ver figura 2.19.

figura 2.19 - **Arquitetura da GPU Nvidia Fermi**



Fonte: NVIDIA[2],2009.

Esta arquitetura dispõe 16 *streaming multiprocessors (SMs)* com 32 *cores* cada, denominados *streaming processors (SPs)* e uma *cache* L1 compartilhada pelas *threads* em execução no *SM.* Os 16 *SMs* compartilham uma *cache* L2 e o barramento de conexão com a *host interface* que, por sua vez, faz a conexão da *GPU* com a *CPU* através de uma interface PCI-e. O módulo *GigaThread* distribui os blocos de *threads* para as unidades de escalonamento dos *SMs.* Seis interfaces de acesso à memória externa com 64 bits dão suporte até 6 GB de *DRAM*.

Com sua arquitetura massivamente paralela, alto desempenho em operações em ponto flutuante e elevado *throughput* nas operações com a memória *onboard*, as *GPUs* tornaram-se, particularmente, adequadas ao uso em aplicações HPC (KINDATRENKO,2009).

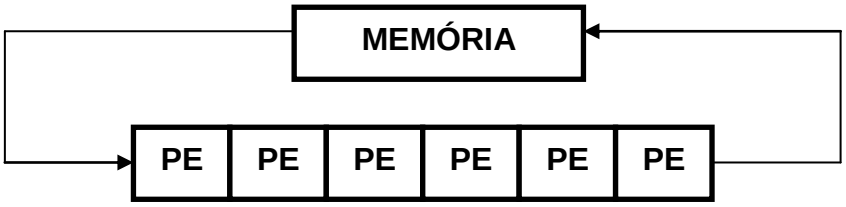
De forma semelhante ao que ocorre nas implementações com *FPGAs*, o principal fator de comprometimento do desempenho com *GPUs*, refere-se à largura-de-banda no acesso à memória externa, que pode se constituir em um “gargalo”, quando milhares de *threads* estão sendo, simultaneamente, executadas.

2.7 Arquitetura Sistólica

A Arquitetura Sistólica foi proposta por Kung e Leiserson, inicialmente, como uma solução eficiente na implementação em *Very Large-Scale of Integration* (*VLSI)* de operações de multiplicação de matrizes (KUNG,1978). Posteriormente, verificou-se que o algoritmo de programação dinâmica, dado o seu alto grau de paralelismo, também, poderia ser mapeado em um Array Sistólico (KUNG,1982).

O conceito de Arquitetura Sistólica baseia-se num fluxo contínuo de dados *(stream)* sincronizado por um sinal de *Clock*, originado a partir de uma Memória, seguindo em direção a um agrupamento de Elementos de Processamento *(PEs)*. Ao final, os resultados gerados retornam à esta Memória. Ver figura 2.20.

figura 2.20 – Princípio Básico da Arquitetura Sistólica



Fonte: Elaborada pelo autor.

A semelhança com o processo ventricular de *sístole/diástole* que ocorre com o sangue que flui para e a partir do coração, deu nome a este tipo de arquitetura.

Para um melhor entendimento das Arquiteturas Sistólicas, parte-se do princípio que as tarefas computacionais podem ser classificadas em dois tipos de famílias: *"I/O-Bound Computation" e "Compute-Bound Computation*". Se o número de operações a serem realizadas for superior ao número de operações de entrada/saída, temos uma computação do tipo *Compute-Bound*; caso contrário, ela será do tipo *I/O-Bound*. Exemplificando, a operação de multiplicação de matrizes é do tipo *Compute-Bound*, uma vez que, o número de operações de multiplicação e de soma, é bem superior ao número de operações de entrada/saída de dados. Já a operação de soma de matrizes, cujo número de operações internas de soma é inferior ao número de operações de leitura/escrita de dados, é um exemplo de uma *I/O-Bound Computation* (KUNG,1982).

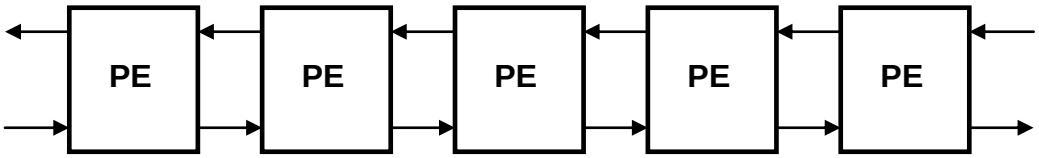
De uma forma geral, o aumento no desempenho em uma aplicação do tipo *I/O-Bound Computation*, passa, essencialmente, pela elevação da largura-de-banda de sua Memória principal. Por sua vez, as Arquiteturas Sistólicas buscam solucionar problemas de desempenho em tarefas do tipo *Compute-Bound*. A partir da substituição do único *PE* (ver figura 2.19) por um *Array Sistólico* de *PEs* (ver figura 2.21), busca-se um crescimento no *throughput* da arquitetura, mantendo-se a mesma largura-de-banda no acesso à memória. A solução está no máximo aproveitamento dos dados ao longo do *Array Sistólico*; de forma que, re-leituras não sejam necessárias ou que, pelo menos, sejam minimizadas. Assim, como os resultados são gerados a partir de um menor número de ciclos de leitura, o *throughput* do sistema é elevado.

Resumindo, a substituição de um único elemento de processamento por um *Array* de *PEs*, aumenta, significativamente, a capacidade de processamento da arquitetura e o seu desempenho, mantendo-se a mesma largura-de-banda da Memória.

A tarefa computacional a ser executada é subdividida em tarefas menores, e mais simples, e distribuídas entre os *PEs* que irão atuar de forma paralela. Estes, por sua vez, são organizados em estruturas unidimensionais ou bidimensionais, cuja topologia, seja ela retangular, triangular ou hexagonal, busca a máxima exploração do paralelismo (KUNG,1979).

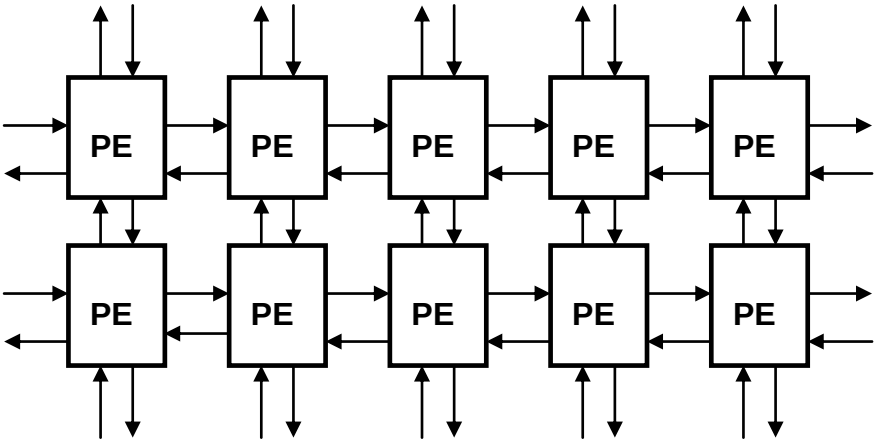
A figura 2.21 apresenta uma *Array Sistólico* unidimensional, onde cada *PE* comunica-se com, no máximo, outros dois e o fluxo de dados ocorre de uma forma semelhante ao que acontece em estruturas que fazem uso de *pipeline*. A diferença em relação a estas estruturas é que, no *Array Sistólico*, as unidades de processamento *(PEs)* executam, exatamente, a mesma tarefa computacional. As operações de I/O ocorrem, apenas, nos *PEs* localizadas nas bordas. A figura 2.22 apresenta uma topologia bidimensional.

figura 2.21 – Array Sistólico Unidimensional.



Fonte: Elaborada pelo autor.

figura 2.22 – Array Sistólico Bidimensional.



Fonte: Elaborada pelo autor.

A eficiência na implementação de um determinado algoritmo, em uma *Arquitetura Sistólica*, exige a verificação de seu potencial de paralelismo; ou seja, deve-se buscar o conjunto de tarefas nas quais não há dependência de dados e que, portanto, podem ser, simultaneamente, executadas. Assim, a identificação do grau de paralelismo de um algoritmo exige a exploração de suas características estruturais quanto aos aspectos de dependência de dados e de controle (HOSSAIN,2002).

Dessa forma, aqueles algoritmos que apresentarem forte dependência de dados e, com isto, impuserem uma execução sequencial em seus processos, serão de difícil mapeamento numa *Arquitetura Sistólica*.

Para uma implementação eficiente, a *Arquitetura Sistólica* deve ter um fluxo de dados simples e regular com poucas linhas de controle e utilizando, unicamente, o *Clock* como principal linha de sincronização (KUNG,1987). Com células de computação simples e interconectadas de uma forma regular, a *Arquitetura Sistólica* possui forte escalabilidade, permitindo a sua expansão em função das demandas do projeto.

Em geral, as aplicações que fazem uso de *Arquiteturas Sistólicas* apresentam, como principal característica, demandas computacionais elevadas. Abaixo, alguns exemplos :

- Programação Dinâmica (ALWAN,2012).
- Processamento de Imagens (SALVADOR,2013).
- Processamento de Sinais – FFT (NASH,2014)
- Redes Neurais (MAZURKIEWICZ,2002)
- Convolução (KUNG,1982).
- Álgebra Matricial, em geral.

2.8 Conclusão

Este capítulo apresentou alguns conceitos básicos que deverão facilitar o perfeito entendimento dos assuntos abordados nesta dissertação. Inicialmente, houve uma breve introdução na àrea biológica seguida de uma apresentação das técnicas de Alinhamento de Sequências Biológicas e de seus algoritmos exatos. Em seguida, foram abordados os conceitos de Computação de Alto Desempenho e de Hardware Reconfiguravel. Na sequencia, foram apresentados os dispositivos de aceleração, o FPGA e a GPGPU; bem como, a arquitetura sistólica.

3 TRABALHOS RELACIONADOS

Neste capítulo serão apresentadas algumas estratégias existentes na literatura recente para a execução do algoritmo Needleman-Wunsch em plataformas baseadas em *Clusters* ou em arquiteturas híbridas aceleradas por *GPUs* ou *FPGAs.*

Como a fase 1 do algoritmo Smith-Waterman é semelhante a do algoritmo Needleman-Wunsch, algumas das soluções apresentadas para aquele algoritmo com seus respectivos resultados foram, também, consideradas nesta apresentação.

Os resultados apresentados neste capítulo serão, mais adiante, comparados com os obtidos em nossa implementação.

3.1 Large-Scale Pairwise Alignments on GPU Clusters: Exploring the Implementation Space

Em (TRUONG,2014) foram propostas quatro implementações distintas para *GPU* do algoritmo Needleman-Wunsch para o múltiplo alinhamento de pares de sequências de *DNA*. As implementações estão disponíveis para *download* em: *http://nps.missouri.edu/nps_wiki/index.php/Code*. Em todas elas foi feita a opção pelo uso do *gap linear*.

As soluções adotadas nestas propostas são comparadas ao *Rodinia-NW*, uma implementação *open-source* "baixada" a partir do *Rodinia Benchmark Suite* (RODINIA,2015) que disponibiliza o *kernel* de diversos algoritmos para aplicações *HPC* em plataformas *GPUs e multi-core CPUs*.

Os *kernels* das implementações propostas pelo artigo foram integrados à uma estrutura *MPI* para o desenvolvimento de um *Cluster CPU-GPU*. Esta solução distribuída focou *Clusters* homogêneos e heterogêneos, este último, composto por *nós* com diferentes configurações de *hardware* e capacidades de processamento.

Na descrição do *cluster*, bem como, na apresentação dos resultados com ele obtidos, não foi definido qual das quatro implementações foi, de fato, a escohida para as *GPUs*, uma vez que, segundo o artigo, poderia ser qualquer uma delas.

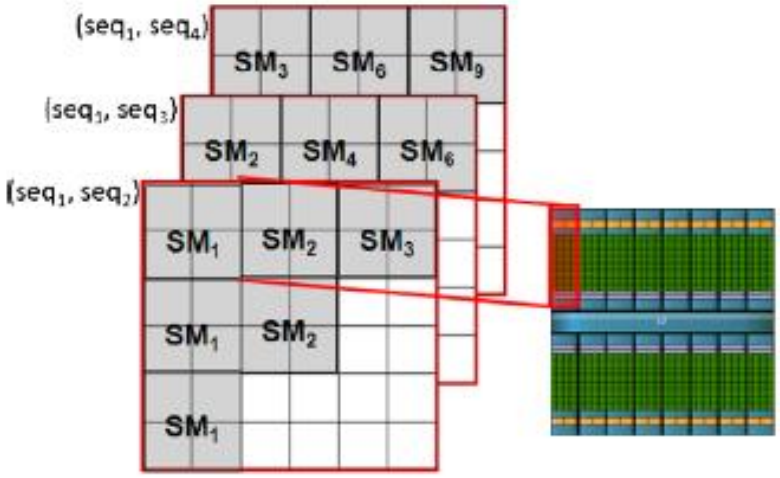
Assim, será apresentada a descrição detalhada de uma destas quatro possíveis abordagens de implementação do algoritmo *NW*, no caso, a *TiledDScan-mW.*

TiledDScan-mNW

Esta implementação segue a mesma abordagem usada no *Rodinia-NW*. Como o cálculo de cada elemento da *matriz PD* utiliza três outros adjacentes, para melhorar o desempenho, a idéia foi concentrar a utilização da memória compartilhada *(shared)*. Assim, a matriz *PD* é particionada em blocos (denominados *tiles*) e cada *tile* ocupa o espaço disponibilizado pela memória compartilhada.

Operando em modo diagonal, múltiplas invocações do *kernel* ocorrem durante o processo de alinhamento. Para cada uma delas, múltiplas *matrizes PD* são processadas, concorrentemente, utilizando diferentes *blocos* de *threads* em diferentes *streaming multiprocessors (SMs)*. A figura 3.1 apresenta esta estrutura num processo com três alinhamentos simultâneos.

figura 3.1 **- Configuração *TiledDScan-mNW***



Fonte: TRUONG,2014

Na primeira iteração, os *tiles* do canto superior-esquerdo das três matrizes são processados, em paralelo, por três blocos-*threads* que são mapeados em três *SMs*. Na segunda iteração, os *tiles* da segunda diagonal das matrizes são, simultaneamente, processados em seis blocos-*threads* mapeados do *SM1* até o *SM6* e, assim, sucessivamente. Dessa forma, ocorre a exploração do paralelismo em dois níveis: (i) através da *threads* dentro dos *tiles* e (ii) diferentes *tiles* na mesma diagonal são processados, concorrentemente, por *blocos* distintos. *Threads* pertencentes ao mesmo *bloco* utilizam, temporariamente, os dados armazenados na

memória compartilhada. Ao final do processamento de um *tile*, os dados são enviados, em modo coalescido, à memória global.

As duas principais otimizações da implementação *TiledDScan-mNW* em relação à *Rodinia-NW*, foram:

1) Considerando *m* alinhamentos, o número de invocações do *kernel* foi reduzido de um fator *m* e, para cada um deles, o número de blocos-*threads* é acrescido de um fator *m*. Desta forma, ocorrem duas vantagens: (i) a redução no número de invocações do *kernel*, reduzindo a participação do seu *overhead* no tempo total de execução e (ii) um maior número de *threads* simultâneas implica numa maior utilização dos recursos da *GPU*.
2) Na implementação *Rodinia-NW*, a *matriz PD* é inicializa na *CPU* e, em seguida, transferida para a *GPU*. O que, dependendo da quantidade de sequências e de seus tamanhos, pode consumir bastante tempo. A otimização, neste caso, é inicializar a matriz na *GPU*.

Uma outra otimização implementada pelas quatro abordagens propostas foi a utilização da *pinned memory* nos processos de transferência de dados entre a *CPU (host)* e a *GPU*.

Análise de desempenho

Para avaliar o desempenho obtido com uma única *GPU*, foram utilizados diversos dispositivos classificados em dois grupos *(High-end* e *Low-end GPUs)*, conforme apresenta a Tabela 3.1.

Tabela 3.1 - High-end e Low-end GPUs

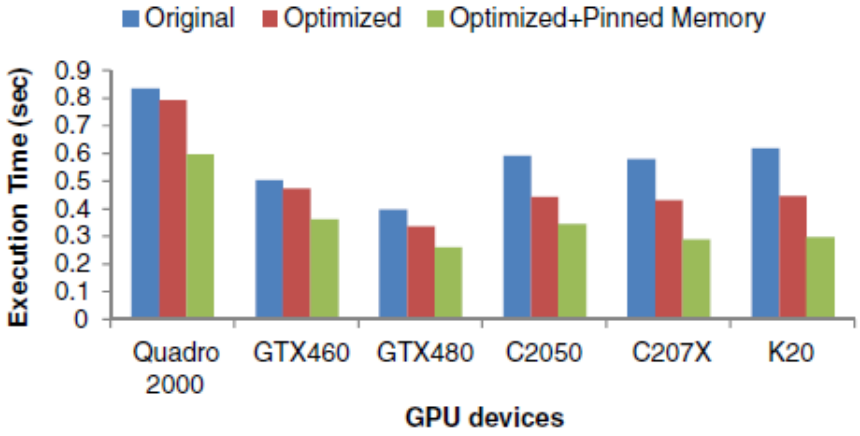
GPU	Type	Values
Low-end GPUs	Quadro 2000	4 SM × 48 cores ~1 GB Global memory
	GTX 460	7 SM×48 cores ~1 GB Global memory
	GTX 480	15 SM×32 cores ~1.5 GB Global memory
High-end GPUs	Tesla C2050	14 SM×32 cores ~2.6 GB Global memory
	Tesla C2070/C2075	14 SM×32 cores ~5 GB Global memory
	Tesla K20	13 SM×192 cores ~4.7 GB Global memory

Fonte: TRUONG,2014

Nos experimentos, inclusive, no *Cluster* foi utilizado um conjunto de dados com cerca de 25.000 sequências do gene bacteriano 16S rDNA, gerados a partir do *Ribosomal Database Project* (COLE,2009). Este gene, com cerca de 1.536 *bps,* é bastante usado nos estudos da filogenia e taxonomia de bactérias pelo fato de estar presente em quase todas as sequências genéticas das mesmas (JANDA,2007).

A figura 3.2 apresenta os resultados referentes ao tempo de execução para 64 alinhamentos com os diferentes dispositivos da Tabela 3.1 e comparando-os com a implementação original *(Rodinia-NW).*

figura 3.2 - **Gráfico comparativo de tempo de execução em relação ao *Rodinia-NW***

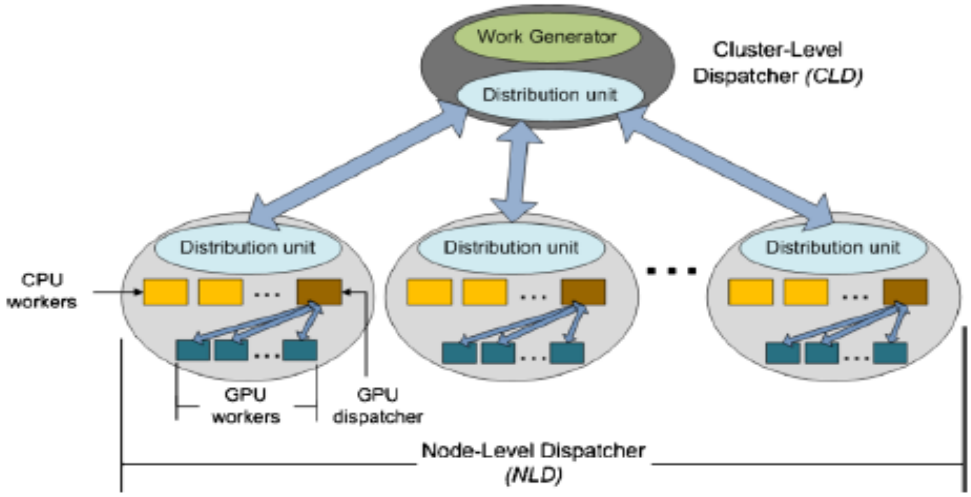


Fonte: TRUONG,2014

Estrutura do Cluster

A figura 3.3 apresenta a estrutura do *Cluster MPI CPU-GPU* composta por um *Cluster-Level Dispatcher(CLD)* e um conjunto de *Node-Level Dispatchers(NLDs)*.

Figura 3.3 - **Arquitetura do *Cluster***



Fonte: TRUONG,2014

O *CLD* atua no *cluster-level*, promovendo a distribuição dos dados *(work-list)* para os *NLDs* em cada um dos *nós* e, no final, agrega os resultados gerados. Por sua vez, no *node-level*, cada *NLD* distribui as tarefas entre os *nós* das *CPUs* e *GPUs*. Em cada *nó,* as *CPU-workers* e as *GPU-workers* executam o algoritmo *NW*.

Resultados obtidos no *Cluster*

Os testes foram realizados em duas diferentes estruturas de *Cluster* (*High-end* e *Low-end Cluster)* cujas configurações estão resumidos na tabela 3.2. No *High-end Cluster* são utilizadas 4 *workstations* e diversas *GPUs/nó*; enquanto que, no *Low-end Cluster* são 6 *desktops* e uma única *GPU/nó.* As *work-lists* enviadas pelo *CLD* aos *NLDs* foram ajustadas para 5.000 pares de alinhamento para a máxima utilização das *CPUs* e *GPUs*.

Tabela 3.2 - **Configurações dos *Clusters***

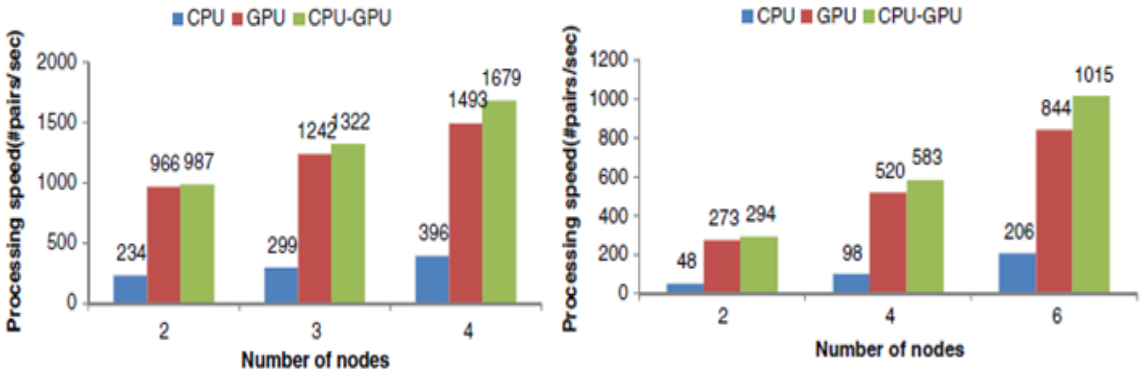
Cluster	Nodes	CPUs	GPUs
Low-end	Node-1	1 × Intel Core 2 Quad Q9400, 2.66 GHz, 4 GB RAM	1 × Quadro 2000
	Node-2		
	Node-3		
	Node-4		
	Node-5	1 × Intel Core 2 Duo E8400, 3.0 GHz, 4 GB RAM	1 × Quadro 2000
	Node-6	1 × Intel Xeon E5-1603, 2.8 GHz, 4 GB RAM	1 × GTX 460
High-end	Node-1	2 × Intel Xeon E5620, 2.4 GHz, 48 GB RAM	2 × Tesla C2050
	Node-2		
	Node-3	2 × Intel Xeon E5-2620, 2.0 GHz, 64 GB RAM	Tesla C2050 Tesla C2070 Tesla C2075
	Node-4	2 × Intel Xeon E5620, 2.4 GHz, 48 GB RAM	4 × GTX 480

Fonte: TRUONG,2014

Foi instalado o CUDA 5.0 em todas as máquinas. O S.O. instalado no *high-end Cluster* foi CentOS5.5/6 com g++4.1.2 e no *low-end Cluster* foi usado o S.O. Ubuntu 12.04 with g++ 4.6.3. Para o MPI foi utilizado o MPICH2 (versão 1.4.1p1).

A figura 3.4 apresenta os resultados referentes ao desempenho, em número de pares alinhados/s, para as duas configurações do *Cluster*. Este gráfico apresenta a escalabilidade da arquitetura e o desempenho em 3 situações: (i)uso, apenas, das *CPUs*, (ii) uso isolado das *GPUs* e (iii) uso do conjunto híbrido *CPU+GPU*.

figura 3.4 - **Desempenho apresentado pelos *Clusters***



a) High-end Cluster *b) Low-end Cluster*

Fonte: TRUONG,2014

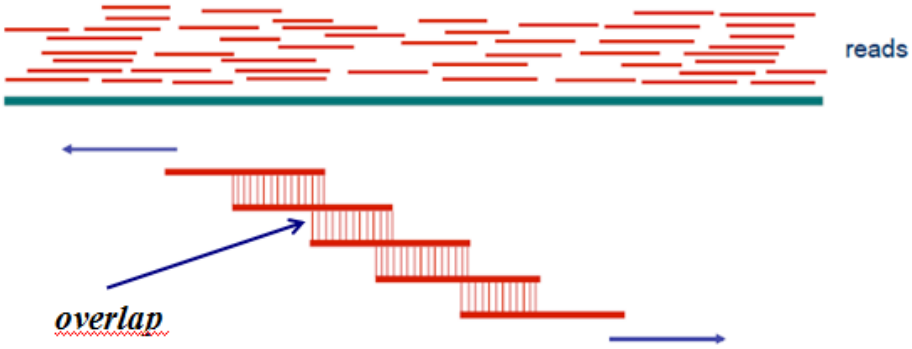
3.2 G-DNA - a highly efficient multi-GPU/MPI tool for aligning nucleotide reads

Em (FROHMBERG,2013) é proposta uma nova ferramenta de software denominada *G-DNA (GPU-based DNA aligner)* como uma solução altamente paralelizada para o processo de montagem das sequências do DNA a partir dos fragmentos gerados pelos modernos sequenciadores da geração *NGS*; particularmente, o 454 da *Roche (*ROCHE_454,2015), o *Solexa* da *Ilumina* (ILLUMINA,2015) e o *Solid* da *Applied Biosystems* (APPLIED,2015)**.** Esta ferramenta foi desenvolvida para o ambiente *CUDA multi-GPU*, incluindo *Clusters* no padrão *MPI*.

Os sequenciadores atuais ainda não são capazes de construir, de uma só vez, a sequência completa de uma longa cadeia *DNA* como, por exemplo, a do genoma humano com cerca de três bilhões de nucleotídeos. Estes equipamentos geram cópias do *DNA* original e, a partir daí, milhões de pequenos fragmentos denominados *nucleotide reads (DNA/RNA),* ou *short-reads*, de diversos tamanhos *(de 20 a 30.000 bps, dependendo do método escolhido)* são gerados. A *overlap-layout-consensus* é uma das metodologias, tradicionalmente, empregadas na montagem do DNA. Seu método de construção é baseado na comparação, às quais devem ser quantificadas através de escores, das regiões de *overlaps* entre o sufixo e o prefixo dos segmentos *short-reads*, conforme representado na figura 3.5. Esta comparação, de fato, trata-se de um alinhamento global nestas regiões de *overlaps*

e que, em função do seu elevado volume de operações, demanda uma aceleração em seu processo.

figura 3.5 - **Princípio básico do método *overlap-layout consensus***



Fonte: FROHMBERG,2013

A ferramenta proposta neste artigo faz uso do algoritmo *NW* para o alinhamento das regiões de *overlaps* entre os pares de *short-reads*, durante o processo de montagem do *DNA*. A solução foi desenvolvida para uso, tanto em plataformas multi-*GPU*, como em *MPI+GPU Clusters.*

Para que a implementação do *NW* obtivesse o desempenho desejado, algumas otimizações *coarse-grained* e *fine-grained* foram introduzidas. Uma delas foi a utilização de apenas 3 bits na codificação dos nucleotídeos na entrada de dados, já considerando a existência de um quinto símbolo N que representasse uma leitura desconhecida. Dessa forma, em uma palavra de 32 bits é possível o empacotamento de 10 nucleotídeos.

Além disto, algumas melhorias de baixo-nível também foram introduzidas no *software*. Por exemplo, para minimizar o número de instruções condicionais, foi dada uma ênfase especial no sentido de "desenrolar" cada possível *loop* no código do *kernel* da *GPU*. Estas intervenções, segundo o artigo, tornam o código específico para um determinado tamanho de sequência. Assim, para que fosse possível o uso da ferramenta com diversos tamanhos de *short-reads*, foi desenvolvido um determinado número de *templates* baseados no *kernel.* O resultado foi a geração de 28 funções para *GPU*, às quais cobririam todos os casos.

As demais otimizações e detalhes da implementação do *software* não foram apresentadas no artigo; contudo, o seu código-fonte foi disponibilizado para uso acadêmico e pesquisas, através do *site* do projeto: *http://gpualign.cs.put.poznan.pl.*

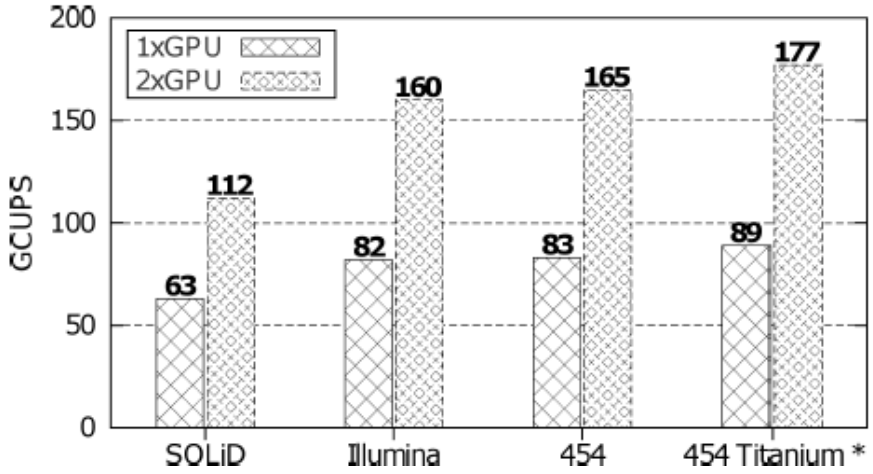
Avaliação e Resultados

Com o objetivo de avaliar o desempenho do software proposto foram usadas sequências geradas a partir de diversos sequenciadores. No caso do *SOLiD*, foram geradas 3,4 milhões de *reads*, com o mesmo comprimento de 46 *bps*, extraídos do genoma bacteriano *Streptococcus suis*. Do sequenciador *Illumina GA IIx* foram obtidas 34 milhões de sequências de 120 *bps* do *Clonorchis sinensis*. Por sua vez, o *Roche 454* gerou 436.000 *reads* da bactéria *Escherichia coli* com um comprimento médio de 235 *bps.* E, finalmente, os dados gerados pelo sequenciador *Roche 454 GS FLX Titanium* (ROCHE_TITANIUM,2015), com um comprimento médio de 1020 *bps*, foram usados para ilustrar o pico de desempenho da implementação.

A seguinte estrutura de *hardware* foi empregada nos testes: Uma CPU Intel Quad-Core Q8200 @ 2.33GHz e duas GPUs NVIDIA GeForce GTX 580.

A figura 3.6 apresenta o gráfico de desempenho em *Cells updates per second (CUPS)* para os diversos tamanhos de sequências. O *CUPS* é a métrica, normalmente, usada para comparar o desempenho entre arquiteturas que aceleraram algoritmos baseados em Programação Dinâmica. Ela indica quantas células da *Matriz PD* serão atualizadas a cada segundo; por outro lado, o seu valor inverso quantifica o tempo necessário para a computação de um só elemento da matriz (BENKRID,2012). Como em seu cálculo não são considerados os tempos de transmissão de dados, acesso à memória e nem de inicialização, o *CUPS* representa o desempenho obtido, exclusivamente, pelo *hardware* de aceleração.

figura 3.6 - **Desempenho para *reads* de comprimentos diversos**

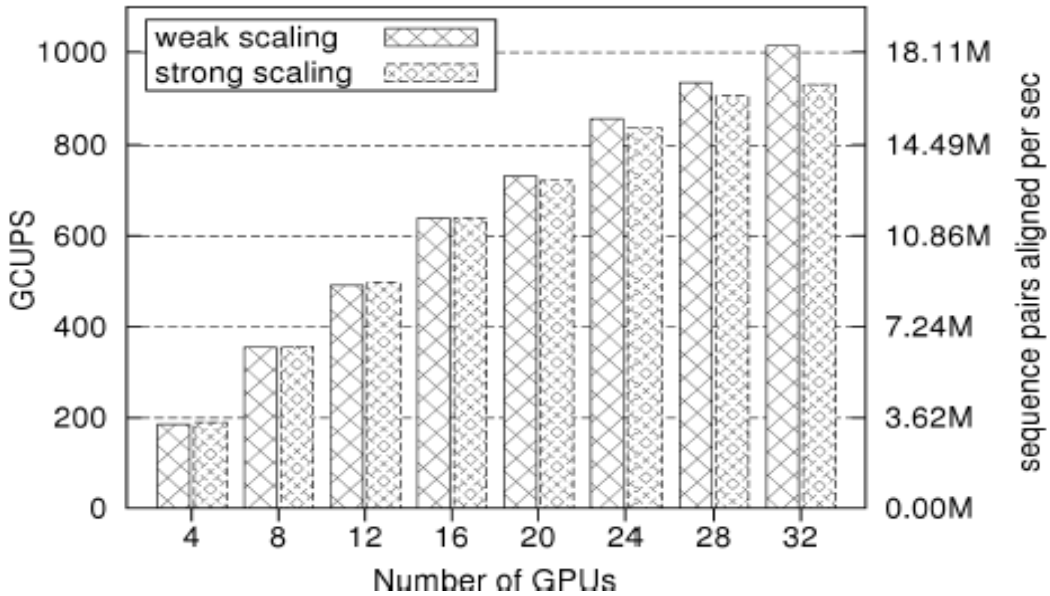


Fonte: FROHMBERG,2013

Pode-se observar que o desempenho do software aumenta com o crescimento do comprimento das sequências. O desempenho, praticamente, dobra com a utilização de 2 GPUs, graças ao paralelismo e à ação eficiente do balanceador de cargas.

A figura 3.7 apresenta a boa escalabilidade do software em um *Cluster MPI*. O desempenho máximo de 1014 GCUPS foi obtido em modo *weak scaling test*, no qual o tamanho do problema cresce proporcionalmente ao número de *GPUs*; neste caso, atingindo 110 milhões de alinhamentos para 32 *GPUs*. Por outro lado, no modo *strong scale test*, onde o tamanho do problema foi fixado em 55 milhões de alinhamentos, foi atingido o desempenho máximo de 929 GCUPS. Observando que, nestes dois modos de testes foram utilizadas as *short-reads* com um tamanho médio de 235 *bps* geradas a partir do sequenciador *Roche 454.*

figura 3.7 **- Desempenho no *Cluster MPI***



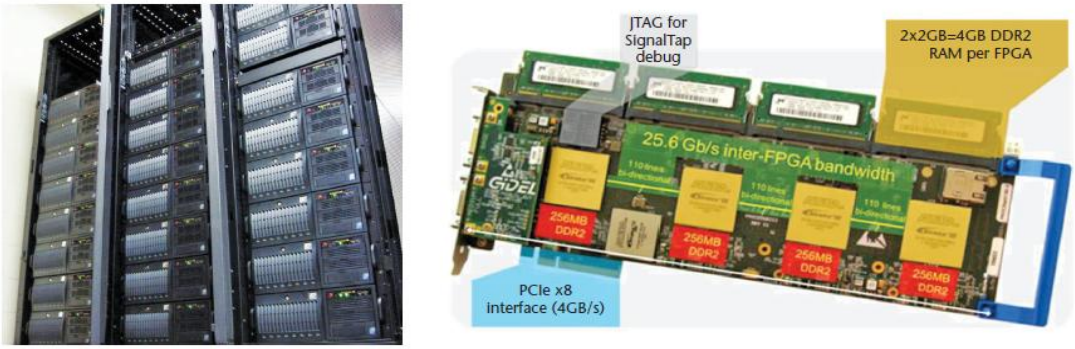
Fonte: FROHMBERG,2013

3.3 Novo-G : At the Forefront of Scalable Reconfigurable Supercomputing

Em (GEORGE,2011) é apresentado o *Novo-G*, um poderoso computador reconfigurável baseado num *Cluster* de *FPGAs*, construído com o objetivo de dar suporte às pesquisas acadêmicas. O *Novo-G*, instalado no *US National Science Foundation's Center for High-Performance Reconfigurable Computing* (*NSF CHREC Center*), da Universidade da Flórida, está operacional desde Julho/2009. Os estudos iniciais demonstraram que o *Novo-G*, com sua estrutura escalável em até 192

FPGAs, poderia, até então, competir em desempenho com os maiores supercomputadores do mundo com uma fração de seus custos, tamanho, consumo de energia e de refrigeração. A figura 3.8 apresenta o *Novo-G* e uma de sua placas com 4 *FPGAs*.

figura 3.8 - **O *Novo-G* e uma placa e uma placa *quad-FPGA***

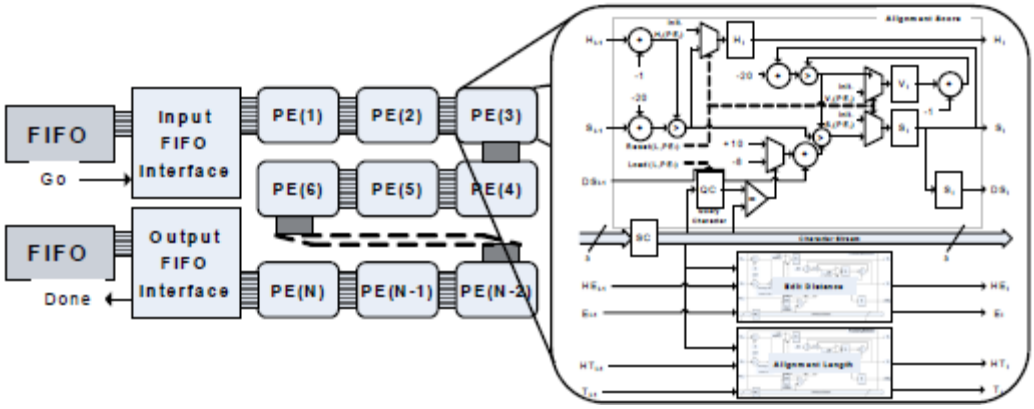


fonte : NSF CHREC

Na configuração apresentada no artigo, o *Novo-G* possui 24 *nós* que atuam como servidores Linux constituídos, individualmente, de um processador Intel quad-core Xeon e duas placas de aceleração *PROCStar-III* da GiDEL Ltd (GIDEL,2014). Cada placa *PROCStar-III* possui 4 *FPGAs Stratix-III* da Altera Corporation (ALTERA,2015), totalizando 192 *FPGAs*.

A abordagem usada na implementação do algoritmo *NW* foi através de um *Array Sistólico* único com 850 PEs em cada *FPGA* (PASCOE,2010). Ver figura 3.9.

Figura 3.9 - **Array Sistólico para *NW*.**



Fonte: GEORGE,2011

Além desta arquitetura, uma nova técnica, denominada *in-stream control*, foi empregada nesta implementação. Esta técnica consiste em: aproveitando-se do fato de serem usados 3 bits na codificação dos nucleotídeos (A,C,G,T) acrescido do

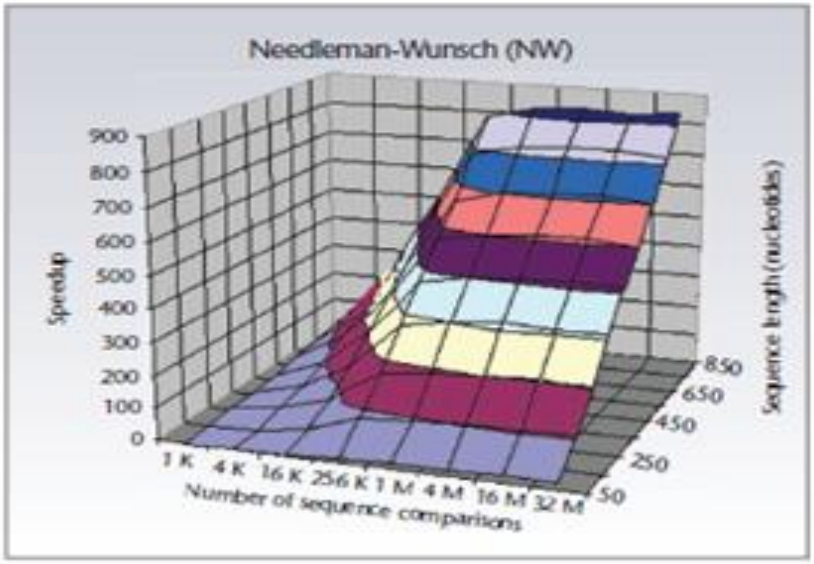
símbolo N que representa a ausencia(Null), inserir 3 novos símbolos de controle (L,R,P) na sequência *stream* de entrada. A leitura destes símbolos possibilita a imediata configuração dos *PEs* nos processos de *load (sequencia Query)*, *reset*, *start* e envio dos resultados para a *FIFO (Done)*. Desta forma, o objetivo é a busca por uma maior desempenho através de uma maior autonomia para o *hardware*. Além disto, há uma consequente redução na complexidade das unidades de controles associadas aos *Arrays Sistólicos* e de suas áreas ocupadas *on-chip*, liberando recursos para implementação de mais blocos de processamento *(PEs)*.

Avaliação e Resultados

Objetivo avaliar o desempenho da implementação do algoritmo *NW* no *Novo-G*, empregando a técnica *in-stream control*, foi utilizada uma grande base-de-dados e múltiplos *nós*. Foi definido que o número de PEs/FPGA seria igual ao tamanho das sequências. Para efeitos comparativos, o algoritmo foi executado numa implementação otimizada em C com um desempenho da ordem de 100 MCUPS em um processador *AMD Opteron*, a 2,4 GHz, num *S.O. Linux* de 64 bits. Por sua vez, os *FPGAs* do *Novo-G* operaram numa frequencia de *Clock* de 125 MHz.

A figura 3.10 apresenta o gráfico do *speedup* no tempo de execução com um *FPGA* em relação à implementação em C, considerando várias condições de entradas com diferentes números de alinhamentos e tamanhos de sequências.

figura 3.10 **- Gráfico de *speedup* : 01 FPGA versus implementação em C.**



Fonte: GEORGE,2011

Pode ser observado que o *speedup* aumenta na medida em que o tamanho da sequências é aumentada, uma vez que, por opção de projeto, o número de *PEs/FPGA* deveria ser igual ao tamanho das sequências até o limite de 850 PEs.

A Tabela 3.3 foi construída considerando um total de 32 milhões de alinhamentos, para os quais, a implementação em C dispendeu 11.026 horas. Ela demonstra a escalabilidade do *Novo-G* ao apresentar os resultados de *speedup* e de tempo de execução, à medida em que aumenta o número de *FPGAs*. O resultado para 192 *FPGAs*, apesar de ser uma estimativa, indica que um único *Novo-G* tem o desempenho equivalente a mais de 147.000 *PCs* convencionais Opteron.

Tabela 3.3 - Tabela comparativa para 32 milhões de alinhamentos

# FPGAs	Runtime(s)	Speedup
1	47.616	833
4	12.014	3.304
96	503	78.914
128	391	101.518
192(est.)	270	147.013

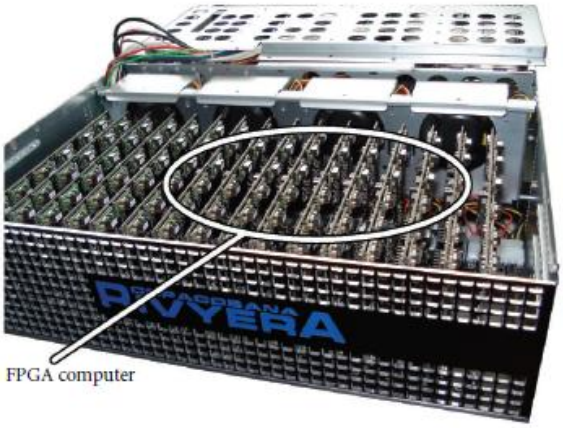
Fonte: GEORGE,2011

3.4 Bioinformatics Applications on the FPGA-Based High-Performance Computer RIVYERA

Em (WIENBRANDT,2013) o RIVYERA é apresentado. Trata-se de uma plataforma computacional de alto-desempenho baseada em *FPGAs*. Neste artigo, dois versões desta plataforma são apresentadas: o RIVYERA S3-5000, ver figura 3.11, o qual foi configurado para executar os algoritmos Smith-Waterman e BLAST e o, mais recente, o RIVYERA S6-LX150, adequado para o BWA *(Burrows–Wheeler Aligner)*, um algoritmo de alinhamento das *short-reads* no processo de montagem das sequências do DNA.

A arquitetura RIVYERA foi desenvolvida e distribuída pela SciEngines GmbH (SCIENGINES,2015), uma empresa especializada em aplicações de alto-desempenho através do uso da computação reconfigurável *(HPRC)*.

figura 3.11 - **O super-computador reconfigurável RIVYERA S3-5000.**



fonte: SCIENGINES,2015

O RIVYERA S3-5000 pode ser configurado com até 128 *FPGAs* Xilinx Spartan3-5000 distribuídos em 16 placas com 8 *FPGAs*, cada uma. Cada um destes *FPGAs* está conectado a 32 MB de *DRAMs*. Por sua vez, o RIVYERA S6-LX150, também, pode ser configurado com até 128 *FPGAs* Xilinx Spartan6-LX150; no entanto, cada um de seus *FPGAs* poderá estar conectado a 512 MB DDR3-RAM podendo ser extendido até 2GB. Com este elevado recurso de memória, o RIVYERA S6-LX150 consegue atender às demandas exigidas pelo algoritmo BWA que lida com extensos genomas de referência (até 4 *Gbps*) como o genoma humano.

O barramento de alto-desempenho ofertado pelo RIVYERA é organizado em uma cadeia sistólica; ou seja, todos os *FPGAs* pertencentes à mesma placa estão conectados ponto-a-ponto, ao seu vizinho, formando um anel.

A comunicação do *host* com as placas dos *FPGAs* ocorre através de uma interface *PCI-e*. Todas as rotinas de comunicação *software-hardware* foram implementadas numa *API* incluindo transmissão em *broadcast*, configuração dos *FPGAs* e acesso às *DRAMs* das placas.

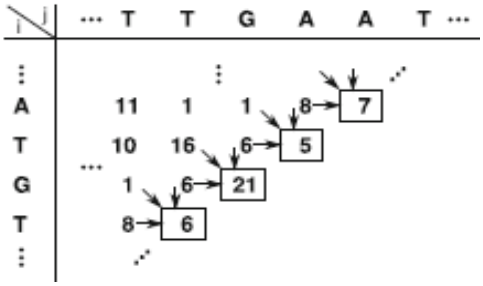
Implementação do Smith-Waterman (SW)

O artigo apresenta a implementação, e os resultados, para o algoritmo *SW* cuja fase 1, conforme já afirmado, possui grande semelhança com o *NW*.

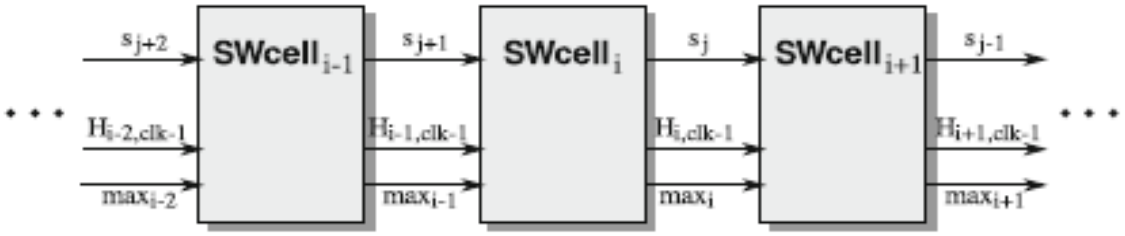
A figura 3.12a apresenta um exemplo de construção da *Matriz PD* evoluindo de modo paralelo ao longo das anti-diagonais, por não haver dependência de dados.

A figura 3.12b apresenta os blocos de processamento denominados *SWcell* e sua arquitetura baseada num *pipeline* de *Arrays Sistólicos* de *PEs*, em cada *FPGA*.

figura 3.12 - **a) Exemplo de cálculo da matriz *SW.***



b) Visão parcial da cadeia de *PEs.*



Fonte: WIENBRANDT,2013

Cada célula *SWcell* recebe um caractere da sequência *query*. Assim, o tamanho *query* define a quantidade de células *SWcell*. A cada novo ciclo de *clock,* cada célula *SWcell* gera o valor de uma célula da anti-diagonal da matriz.

Avaliação de Desempenho

Com o objetivo de avaliar o desempenho da plataforma, o algoritmo SW foi escolhido para a execução do alinhamento de *short-reads*, com um tamanho médio igual a 100 *bps.* As células *SWcell* foram configuradas para este algortimo, recebendo os quatro símbolos para os nucleotídeos e um quinto símbolo de ausência de dado (A,C,G,T e N). Além disto, haveriam quatro 4 cadeias de *SWcell*, em cada *FPGA*, cada uma com 100 células, atingindo um desempenho aproximado de 25 *GCUPS* (SCIENGINES_WP, 2012).

Então, o RIVYERA S3-5000 com 128 *FPGAs* totaliza 512 cadeias *SWcell* o que possibilita o alinhamento paralelo de 512 sequências. Com esta configuração a plataforma atingiu um desempenho de 3,05 *TCUPS*. Desta forma, o alinhamento de um milhão de sequências *reads* geradas pelo sequenciador Illumina com um

tamanho médio de 100 *bps* contra o genoma humano (*hg19* com 3,2 *Gbps*) requer, aproximadamente, 29 horas.

A tabela 3.4 cujos dados foram fornecidos pela *CLCbio* (CLCBIO,2012), uma empresa especializada no desenvolvimento de soluções para bioinformática, apresenta um quadro comparativo de desempenho entre o RIVYERA S3-5000 e diversas outras soluções em *PCs* e *Clusters*, na execução do algoritmo *SW* para *DNA* e Proteínas.

tabela 3.4 - **Quadro comparativo de desempenho no algoritmo SW.**

Architecture	DNA/prot.	Speed (GCUPS)
RIVYERA S3-5000	DNA	3,050
CLCbio Xeon X3210 @ 2.13 GHz (8 cores)	DNA	45
CLCbio Core2Duo @ 2.17 GHz (2 cores)	DNA	13
RIVYERA S3-5000	Protein	1,500
IBM QS20 blade (2x CellBE @ 3.2 GHz)	Protein	33
CUDASW++2.0 (GeForce GTX280)	Protein	17
Sony PS-3 (1x CellBE @ 3.2 GHz)	Protein	12

fonte : WIENBRANDT,2013

3.5 Conclusão

Neste capítulo, foram apresentados quatro trabalhos recentes com diferentes propostas de aceleração do algoritmo exato de alinhamento de sequências biológicas utilizando *GPUs* e *FPGAs*. Em todos eles, a solução final foi baseada no uso de *Clusters*. Também foram apresentados os resultados obtidos em cada uma destas implementações, essencialmente, referentes a desempenho. Estes resultados serão, individualmente, comparados aos obtidos com nossa proposta de arquitetura, mais adiante , no capítulo 5 - Resultados Experimentais.

O primeiro trabalho (TRUONG,2014) apresenta quatro implementações distintas, em GPU, para o algoritmo *NW*. Foi apresentado os detalhes de uma dessas implementações na qual as matrizes *PD* são particionadas em blocos e processadas, em paralelo, a partir de múltiplas chamadas do *kernel*. Na sequência foi apresentada a estrutura híbrida *MPI-CUDA* objetivando a construção de um *Cluster CPU-GPU*.

O segundo trabalho (FROHMBERG,2013) propôs um ferramenta de *software* à qual utiliza o algoritmo *NW* no processo de *assembly* do *DNA* baseado na

metodologia *overlap-layout-consensus*. A ferramenta implementa algumas otimizações que visam a aceleração do algoritmo. Os resultados apresentados demonstram excelente escalabilidade num sistema com múltiplas *GPUs*, incluindo um *Cluster* baseado no padrão de comunicação *MPI*.

O terceiro trabalho (GEORGE,2011) apresentou o *Novo-G*, um poderoso computador baseado num *Cluster* constituído por 192 FPGAs. A abordagem usada na implementação do algoritmo *NW* foi através de Arrays Sistólicos com 850 *PEs* por *FPGA*.

O quarto trabalho (WIENBRANDT,2013) apresentou o RIVYERA, uma plataforma computacional, também, baseada num *Cluster* com 128 *FPGAs*. O artigo apresentou a implementação do algoritmo *Smith-Waterman*, cuja fase 1 é semelhante a do algoritmo *NW*. A estratégia adotada para a construção da matriz *PD* utiliza as anti-diagonais e, também, faz uso de *Arrays Sistólicos*.

4 ARQUITETURA PROPOSTA

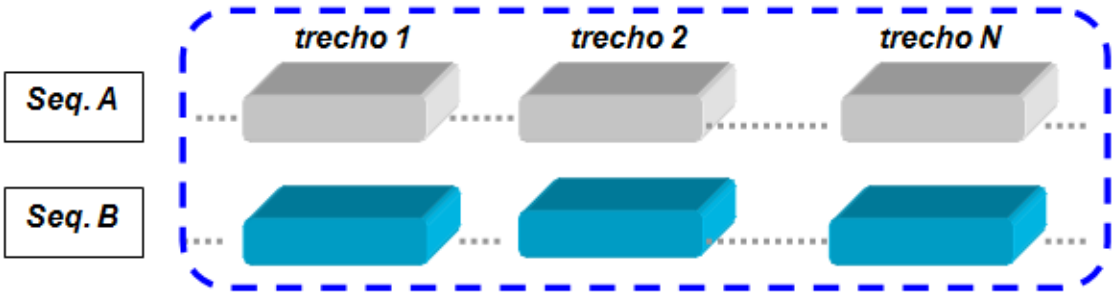
Neste capítulo é apresentada, em detalhes, a arquitetura híbrida baseada em um dispositivo reconfigurável proposta para este projeto. Inicialmente, serão apresentados os objetivos específicos do projeto. Em seguida, é feita uma breve apresentação da placa utilizada na prototipagem da arquitetura em seu *FPGA*. Na sequencia, serão apresentados os detalhes operacionais da plataforma, no tocante, às camadas de *hardware* e *software* e seu desempenho computacional.

4.1 Introdução

Conforme afirmado no capítulo 1, este projeto tem por objetivo o desenvolvimento de uma plataforma aceleradora, doravante denominada de sistema, para o algoritmo *NW* baseada em um dispositivo reconfigurável, no caso, um *FPGA*. Neste projeto o algoritmo *NW* será implementado, exclusivamente, para o alinhamento de sequências de *DNA*. Este Alinhamento poderá ser executado, pelo sistema, de duas formas distintas:

1) Alinhamento Global de 2 Cadeias de *DNA* aplicado em diferentes regiões de suas sequências. Dessa forma, trechos específicos de cada uma delas serão, simultaneamente, introduzidos no sistema para a execução paralela de seus Alinhamentos *(ver figura 4.1)*.

figura 4.1: **Alinhamento paralelo de trechos de 2 sequencias.**

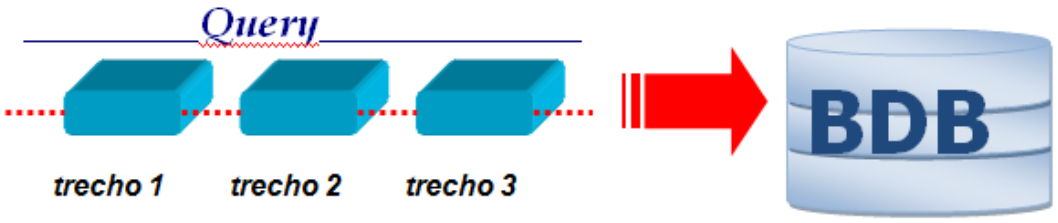


Fonte: Elaborada pelo autor

Neste caso, o Alinhamento busca quantificar o grau de similaridade entre Cadeias distintas, identificar homologias, pesquisar a ancestralidade entre espécies/indivíduos, entre outros.

2) Alinhamento Global de uma Cadeia (denominada *Query*) com as cadeias pertencentes a um determinado Banco de Dados Biológico *(doravante BDB)*. Neste caso, trechos específicos da *Query* são introduzidos no sistema para a execução, simultânea, do Alinhamento com os respectivos trechos das sequências deste *BDB (ver fig. 4.2).* Tais trechos, por exemplo, poderão pertencer a regiões genômicas.

figura 4.2: Alinhamento com as sequencias de um BDB.



Fonte: Elaborada pelo autor

Esta abordagem busca identificar similaridades entre uma determinada sequência em relação às demais pertencentes a um BDB e poderá ter como aplicação, por exemplo: a identificação de padrões que estejam associados a doenças genéticas, a busca por um indivíduo em um determinado *BD* (ex.: *BD* de criminosos), indentificar homologias entre um virus desconhecido em relação a outros já catalogados, entre outros.

Em todos estes processos ocorrem múltiplos alinhamentos de pares de sequências. Estes pares não precisam ser do mesmo tamanho. Os resultados dos alinhamentos poderão ser apresentados através de dois formatos distintos:

a) Apresentação de uma solução-ótima de alinhamento das sequências, duas a duas, com a explícita identificação de supostas ocorrências de mutações entre elas (Inserções, Remoções ou Substituições).

```
Q :          c g g g t a t c c a a
D:           c c c t a g g t c c c a
Q_alinh:     c g g g t a _ _ t _ c c a a
D_alinh:     c c c _ t a g g t c c c _ a
```

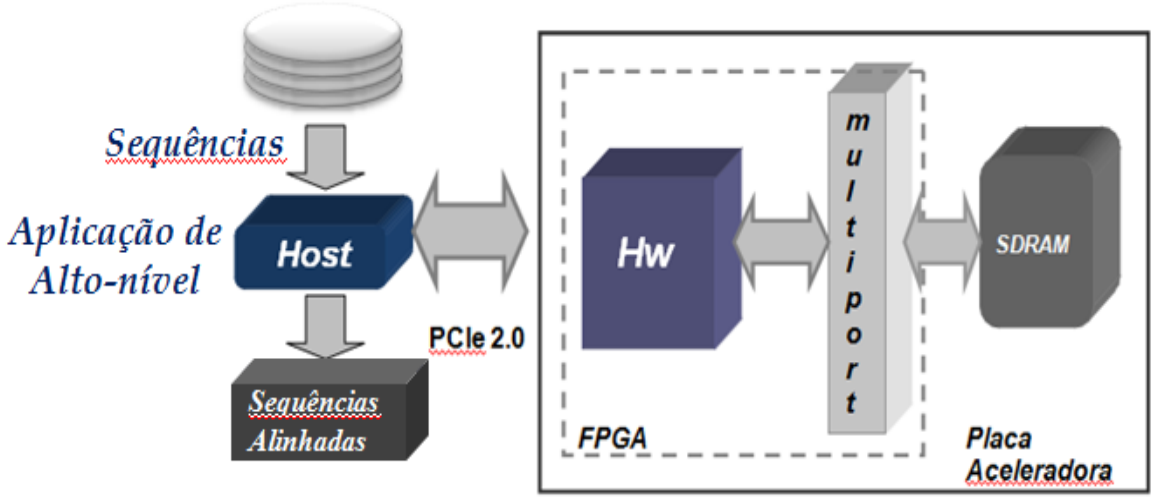
b) Geração de uma Taxa de Alinhamento que quantifica o grau de similaridade entre as sequências alinhadas.

4.2 Estrutura

A figura 4.3 apresenta a estrutura em blocos da arquitetura híbrida proposta. No sistema temos um *PC* que atua como *host,* a partir do qual, as sequências a sofrerem o alinhamento são transferidas à memória local *(SDRAM)* da placa aceleradora. Da mesma forma, ao final do processo, os arquivos com as sequências alinhadas são enviados para a memória principal do *host*. Toda esta comunicação ocorre via *DMA* utilizando uma interface *PCIe 2.0*.

Neste projeto, as sequências de entrada estão num formato de arquivo-texto. As mesmas podem ser acessadas a partir de *BDBs,* tais como: O *GenBank* e o *UniGene*, pertencentes ao *National Center for Biotechnology Information* (NCBI,2015), o *European Molecular Biology Laboratory* (EMBL,2015), entre outros.

figura 4.3: **Estrutura em blocos da arquitetura híbrida proposta**



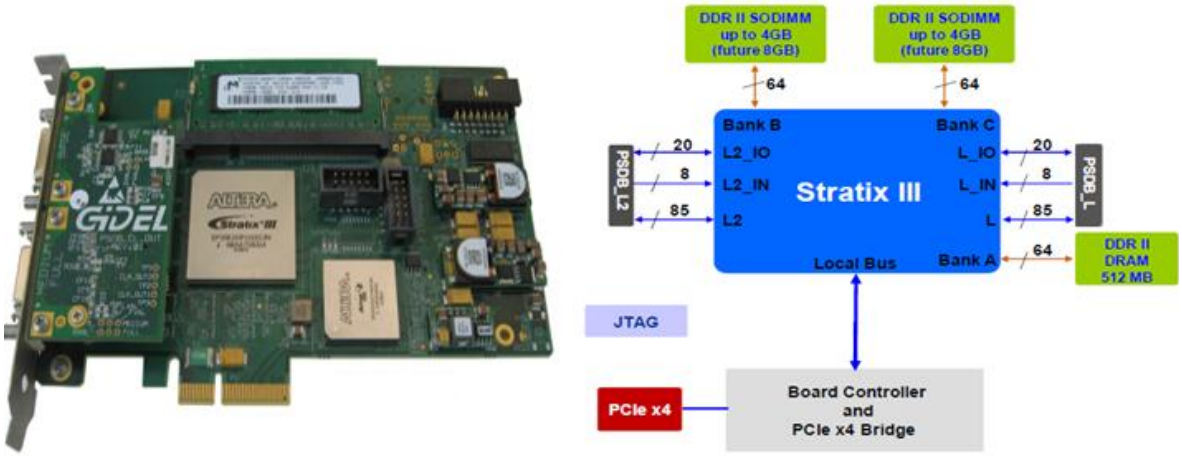
Fonte: Elaborada pelo autor

A placa aceleradora usada no projeto, foi a *PROCe III* fornecida pela *GiDEL Ltd* (GIDEL,2014), ver figura 4.4, a qual tem como componente processador o *FPGA EP3SE260F1152C2* com tecnologia de 65 nm da família *Stratix III* da *Altera Corporation*.

A placa possui 3 bancos de memória *SDRAM DDR2* sendo um deles *on-board* de 512 MB e, os demais, em soquetes *SODIMM* de 4 GB cada. A interface com estes bancos é sintetizado, no *FPGA*, através do *PROCMultiPort*, um *IP Core*

fornecido pela *GiDEL*, que pode possibilita um acesso *multi-port* com até 16 *ports* independentes com seus próprios domínios de *Clock* e largura de dados. O protocolo do *PROCMultiPort* pode ser configurado para acessos randômicos ou sequencial (incluindo o modo *burst).*

figura 4.4 - **Placa aceleradora GiDEL PROCe III**



a) imagem *b) diagrama-em-blocos*

fonte : GiDEL Ltd.

O módulo *PROCMultiport* possui uma largura máxima de 768 bits e sua estrutura interna de interface com os bancos de memória é baseada em *FIFOs*.

As setas bidirecionais na figura 4.3 representam os principais canais de comunicação da arquitetura. Dentre estes, o mais lento, é o da interface com o *host* através da PCIe 2.0 4x com o seu *throughput* máximo de 2.0 GB/s. Por sua vez, o canal de comunicação do *FPGA* com as memórias *SDRAM*, possui um *throughput* máximo de 4 GB/s, com a DDR2 de 512 MB, e de 6 GB/s com as *SODIMMs* de 4GB.

A interface de comunicação interna ao *FPGA* entre o *Multiport* e a arquitetura *(Hw)*, considerando o máximo uso da largura deste canal e da frequência de *Clock*, poderá atingir o seguinte valor de *throughput* :

throughput *(máximo)* = largura máxima do *port* x frequência máxima de operação do FPGA

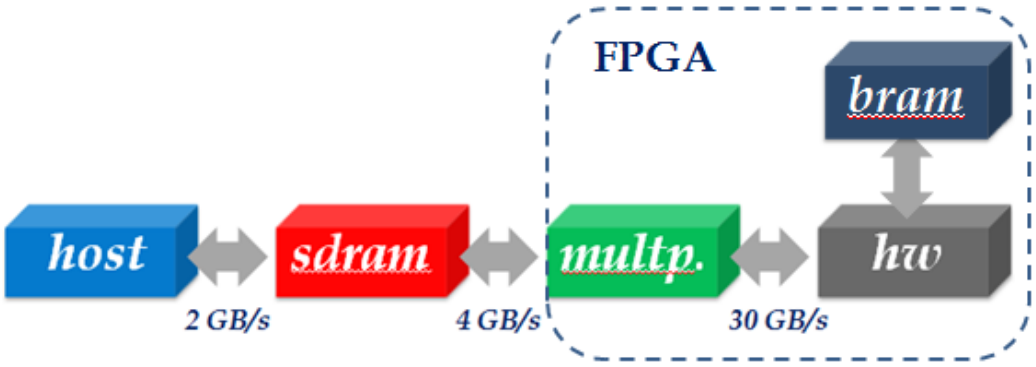
throughput *(máximo)* = 768 bits x 325 MHz = **31,2 GB/s**

Existe, ainda, uma outra interface a ser considerada a qual trata-se da memória interna do FPGA. Esta memória é gerada a partir dos blocos de memória

BRAMs (Blocks of Randomic Access Memory) disponibilizados pelo fabricante no dispositivo. Como será explicado mais adiante, esta memória será utilizada na fase2 do algoritmo *(traceback)*. Por ser intrínseca ao *hardware* do dispositivo, esta interface é a mais veloz de todas anteriores, uma vez que a mesma opera na mesma frequencia do *Clock* interno.

A figura 4.5 reapresenta a estrutura em blocos da arquitetura proposta evidenciando a largura de banda entre as interfaces.

figura 4.5 - **Largura de banda entre os principais blocos.**



Fonte: Elaborada pelo autor

Diante disto, fica evidenciado que as interfaces externas ao FPGA, tanto com as memórias da placa quanto com o *host*, constituem-se em “gargalos” nos processos de comunicação, de um modo geral. Ou seja, apesar do processamento interno ao FPGA ocorrer em alta velocidade, a necessidade frequente de comunicação com estes dispositivos externos degradam o desempenho total do sistema devido à limitada largura-de-banda de suas interfaces.

Como o principal objetivo deste projeto é a busca pelo maior desempenho possível na execução do algoritmo, buscou-se o desenvolvimento de uma arquitetura que tivesse como foco os seguintes requisitos :

1º) A máxima exploração das possibilidades de paralelismo oferecidas pelo Algoritmo. A exploração do paralelismo é uma das principais mecanismos quando o foco é desempenho (FERLIN,2008).

2º) A máxima exploração dos recursos nativos de paralelismo do dispositivo reconfigurável.

3º) A busca pelo máximo *throughput* na comunicação entre o *FPGA* e os bancos de Memória da placa.

4º) A minimização do número de transferências de dados entre placa aceleradora e o *host*.

Com relação ao primeiro requisito, o Algoritmo de Alinhamento *NW* apresenta como principal característica, a recursividade e, portanto, o uso maciço de *loops* com varreduras aninhadas em seu conjunto de dados. Dependendo do tamanho das sequências, esses blocos de processamento podem ser bastante custosos do ponto de vista computacional; no entanto, podem apresentar alto potencial de paralelismo, desde que sejam identificadas independências de dados que possibilitem iterações simultâneas.

No tocante aos requisitos 2 e 3, na medida em que os dados estejam sendo lidos da Memória, através dos canais do *multiport*, o algoritmo será executado, simultaneamente, pela arquitetura implementada no *FPGA* e, neste, a execução será paralelizada através de várias unidades de alinhamento, ainda, a serem detalhadas.

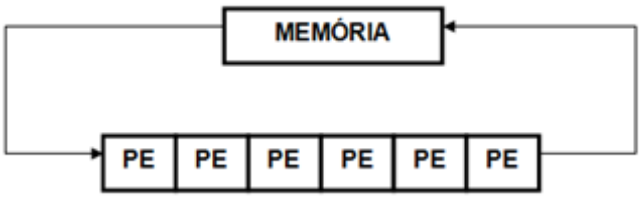
Quanto ao quarto requisito, a estratégia adotada é a de permitir que as transferências de dados entre a placa aceleradora e o *host*, ocorram em, apenas, dois momentos:

a) No início da execução do algoritmo quando, do *host* para a placa, ocorrem as transferências dos parâmetros da arquitetura e do arquivo de configuração do FPGA para a placa aceleradora. Em seguida, os arquivos com o conjunto de sequências a serem alinhadas passam por um processo de formatação e são transferidos, via *DMA*, da memória do *host* para os bancos de memória da placa aceleradora.
b) Ao final do processo de execução do algoritmo no *FPGA*, os arquivos resultantes dos alinhamentos gravados nos bancos de memória da placa são transferidos, via DMA, de volta para a memória principal do *host.*

4.3 Uso da Arquitetura Sistólica

Diante dos requisitos acima apresentados e conforme descrito no capítulo de Fundamentação Teórica, entendemos que a utilização de uma Arquitetura Sistólica seria a melhor abordagem a ser adotada neste projeto (ver figura 4.6).

figura 4.6 - Arquitetura Sistólica Básica.



Fonte: Elaborada pelo autor

Conforme descrito, numa Arquitetura Sistólica, a tarefa computacional a ser executada é subdividida em tarefas menores e distribuída entre os diversos *PEs* que compõem um *Array Sistólico*. Na medida em que estas células geram seus resultados, estes podem ser usados pelas células adjacentes e, assim, sucessivamente. Estes *PEs* deverão ser, na medida do possível, semelhantes, tornando o projeto da arquitetura mais simples, modular e escalável.

Dois mecanismos determinam o ganho de desempenho desta arquitetura:

a) Identifica-se no algoritmo os trechos passíveis de uma execução paralela. Estes, por sua vez, são transformados em sub-tarefas que serão distribuídas e executadas, em paralelo, nos *PEs*.
b) A topologia escolhida para a Arquitetura Sistólica e o seu quantitativo de *PEs* deve ser tal que possibilite o máximo aproveitamento dos dados de entrada, possibiltando que os resultados sejam gerados com o mínimo de re-leituras dos dados da memória. Com este uso otimizado da memória, elevados *throughputs* são obtidos com modestas larguras-de-banda nas comunicações de I/O (KUNG,1982).

A tarefa em seguida é a identificação das possibilidades de paralelismo oferecidas pelo Algoritmo *NW* e que irão nortear a construção da Arquitetura Sistólica, definindo a sua topologia. Ou seja, buscar o mapeamento de um Algoritmo de natureza sequencial em uma arquitetura de execução, essencialmente, paralela.

4.4 Mapeamento do Algoritmo *NW* numa Arquitetura Sistólica

O Algoritmo *NW*, conforme já apresentado no capítulo de Fundamentação Teórica, possui 2 fases distintas :

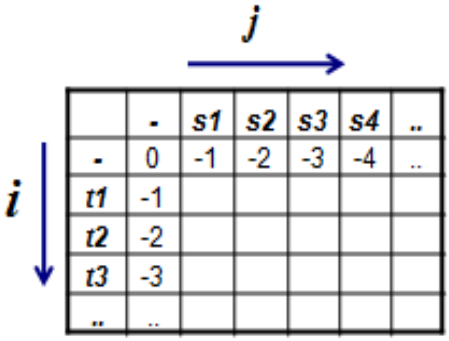
1) Construção da *Matriz PD*.
2) *Traceback*.

No início da primeira fase, a primeira linha e a primeira coluna da *Matriz PD* são inicializadas. Esta inicialização segue as definições dadas pelas equações 4.1a e 4.1b, onde *"g"* representa o custo do *gap* (inserção/remoção).

$$E(i, 0) = g.i \qquad \textbf{eq. 4.1a} \qquad\qquad E(0, j) = g.j \qquad \textbf{eq. 4.1b}$$

A figura 4.7 apresenta o resultado desta fase considerando o Alinhamento de uma sequência ***"S"*** com uma outra ***"T"***, utilizando o custo do *gap* igual a –1.

figura 4.7 - **Inicialização da Matriz PD**



	-	s1	s2	s3	s4	..
-	0	-1	-2	-3	-4	..
t1	-1					
t2	-2					
t3	-3					
..	..					

Fonte: Elaborada pelo autor

Concluída esta inicialização e considerando-se o uso do *gap linear*, a *Matriz PD* é preenchida, conforme as definições dadas pela equação 4.2, a seguir :

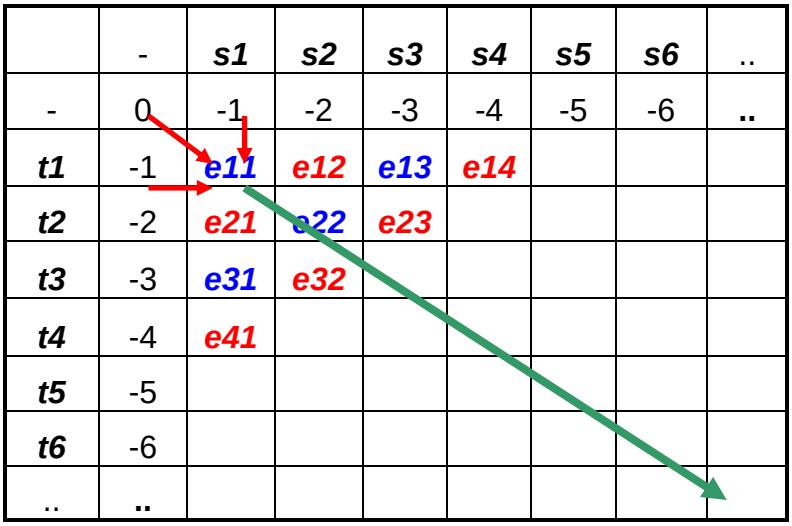
$$\textbf{E(i , j)} = \text{máximo} \begin{cases} \textbf{E(i-1, j) + } g \\ \textbf{E(i-1, j-1) + CC} \text{ (custo de comparação)} \\ \textbf{E(i, j-1) + } g \end{cases} \qquad \textbf{eq. 4.2}$$

A figura 4.8 apresenta a construção da *Matriz PD* e a seta maior indica a direção do seu preenchimento.

Esta fase do Algoritmo possui Complexidade Temporal e Espacial *O(nm)*, onde *n* e *m* são os respectivos comprimentos das sequências. Assim, dependendo do tamanho das sequências e do número de alinhamentos a serem realizados, no caso de uma busca comparativa com as sequências de um banco, o tempo de execução poderá ser muito longo. Dessa forma, a aceleração via *hardware* deve ocorrer nesta fase e implica, necessariamente, em sua paralelização.

A dificuldade, contudo, está na dependência-de-dados associada às questões de recorrência impostas pela equação 4.2, uma vez que, o cálculo de uma determinada célula *E(i,j)* depende dos valores das três outras adjacentes.

figura 4.8 – **Preenchimento da Matriz PD**



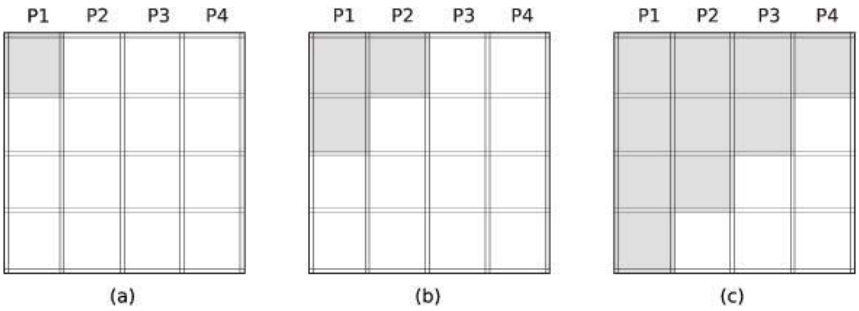
Fonte: Elaborada pelo autor

Observando-se o preenchimento da matriz percebe-se que não há dependencia-de-dados entre os elementos nas anti-diagonais e, portanto, podem ser, simultaneamente, gerados. Ou seja, o elemento *e11* é obtido no primeiro ciclo de geração. No segundo ciclo, os elementos *e12* e *e21* podem ser gerados, ao mesmo tempo. O mesmo ocorrendo com *e31*, *e22* e *e13,* no terceiro ciclo, e, assim, sucessivamente. Essa característica aponta que o algoritmo possui um bom potencial de paralelismo (ALTERA[2],2007).

Uma vez verificada essa possibilidade de paralelização do algoritmo, o próximo passo é o seu mapeamento em uma Arquitetura Sistólica. A abordagem, normalmente, usada neste caso é conhecida como método *wavefront*, ver figura 4.8. Os cálculos que podem ser realizados em paralelo, evoluem, a cada novo ciclo, como ondas em diagonais (BOUKERCHE,2007).

A solução oferecida pelo método *wavefront* é a linearização do processo de geração dos elementos da matriz utilizando um *Array Sistólico Unidimensional e Unidirecional* ou, simplesmente, um *Array Sistólico Linear.* Nesta topologia, cada um dos *PEs* pertencentes ao *Array* é responsável pela geração dos escores de uma única coluna da *Matriz PD*. Na figura 4.8, o método *wavefront* faz uso de 4 *PEs*.

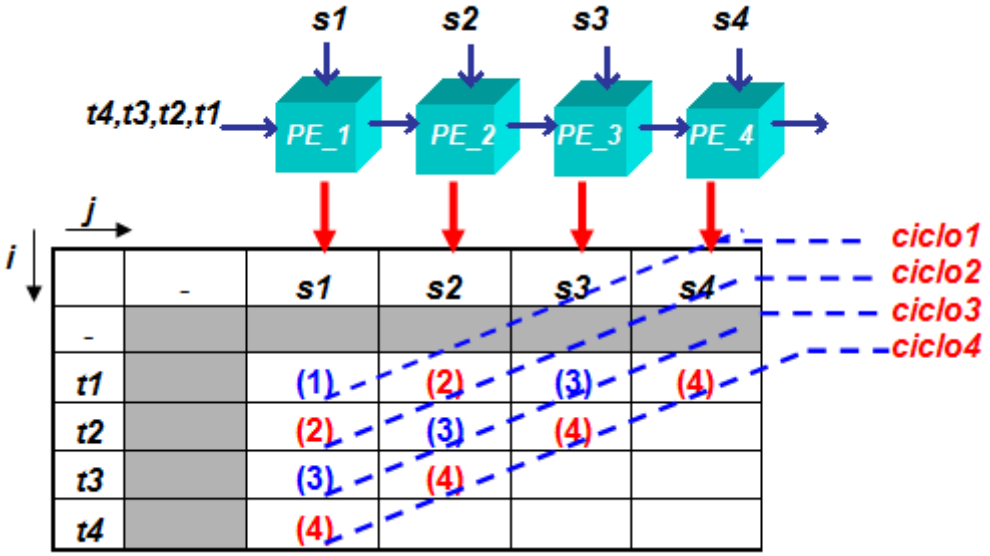
figura 4.9 - **Método *Wavefront***



Fonte: Elaborada pelo autor

A figura 4.10 apresenta o processo de construção da *Matriz PD* utilizando um *Array Sistólico Linear* constituído por um *pipeline* de 4 *PEs*.

figura 4.10 - **Array Sistólico Linear com 4 PEs**



Fonte: Elaborada pelo autor

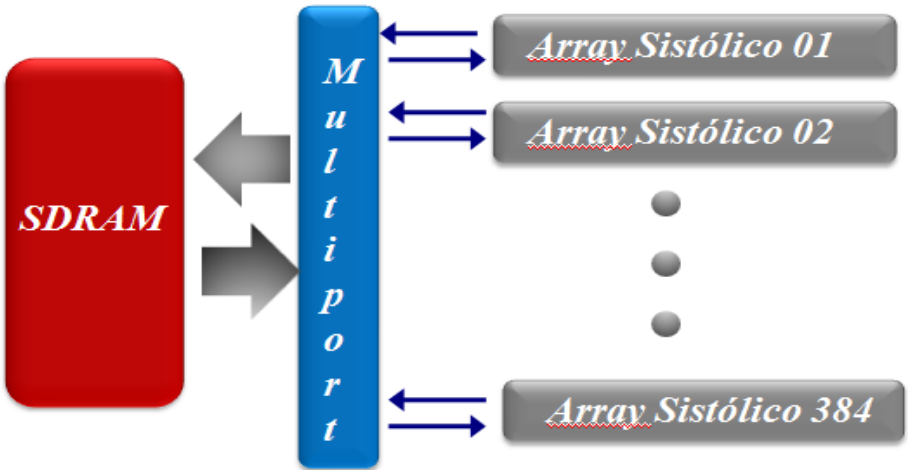
Os dados das sequências ***"S"*** e ***"T"*** são introduzidos de formas distintas. Inicialmente, os dados de ***"S"*** são "carregados", individualmente, um em cada *PE*. A partir daí, a cada novo ciclo de *Clock*, um dado da sequência ***"T"*** será introduzido e deslocado através do *Array*. As linhas tracejadas identificam os ciclos de *Clock* (c1,c2,c3 e c4) e a geração paralela das células da matriz.

Pode-se observar que, o uso do *Array Sistólico Linear* reduz a Complexidade Temporal na geração da Matriz de *O(mn)* para *O(m+n-1)*; o que, dependendo do tamanho das sequências, poderá representar uma redução bastante significativa nos tempos de execução dos alinhamentos.

Os *PEs*, não apenas, geram os escores da matriz como, também, a informação da direção de qual célula anterior, através da equação 4.2, definiu este valor. Como pode ser observado pelas setas que incidem sobre *e11*, na figura 4.7, existem 3 possibilidades de direção: acima, esquerda ou diagonal. Esta informação, que chamaremos de *dir_bits*, é codificada com 2 bits e deverá ser armazenada para ser utilizado, posteriormente, na última fase do algoritmo.

As sequências serão injetadas no *Array*, uma após a outra, em forma de *stream* de dados. Por se tratarem de nucleotídeos e existirem, apenas, 4 tipos, estes dados de entrada são codificados, também, com 2 bits. Dessa forma, a cada novo ciclo de *Clock*, o primeiro *PE* recebe um novo dado de 2 bits através do *multiport*, proveniente da memória. Como esta interface possui uma largura máxima de 768 bits, ela poderá transferir até 384 sequências, simultaneamente, para a arquitetura. Ou seja, com a ocupação máxima do canal do *Multiport*, até 384 Alinhamentos em paralelo poderão ser executados. A figura 4.11, apresenta esta estrutura.

figura 4.11 - **Estrutura com máximo Alinhamento em paralelo**



Fonte: Elaborada pelo autor

Com esta arquitetura foi buscado o atendimento aos três primeiros requisitos, anteriormente, apontados para este projeto, os quais foram:

1) A máxima exploração das possibilidades de paralelismo oferecidas pelo Algoritmo.
2) Exploração máxima do paralelismo no FPGA.
3) Buscar o *throughput* máximo nos canais de comunicação entre o FPGA e as memórias da placa.

O paralelismo nativo do *FPGA* foi explorado em dois níveis de abordagem: um nível de granularidade fina *(fine-grained)* entre os *PEs*, no interior de cada Array, e um de granularidade grossa *(coarse-grained)* entre os *Arrays*.

4.5 Nr. Alinhamentos x Disponibilidade de Recursos

Por se tratar de um projeto cujo principal foco é o desempenho, quanto maior for o número de operações paralelizadas, melhores deverão ser os resultados obtidos. No entanto, a quantidade de recursos disponibilizados pelo FPGA, muitas vezes, é o principal fator limitante.

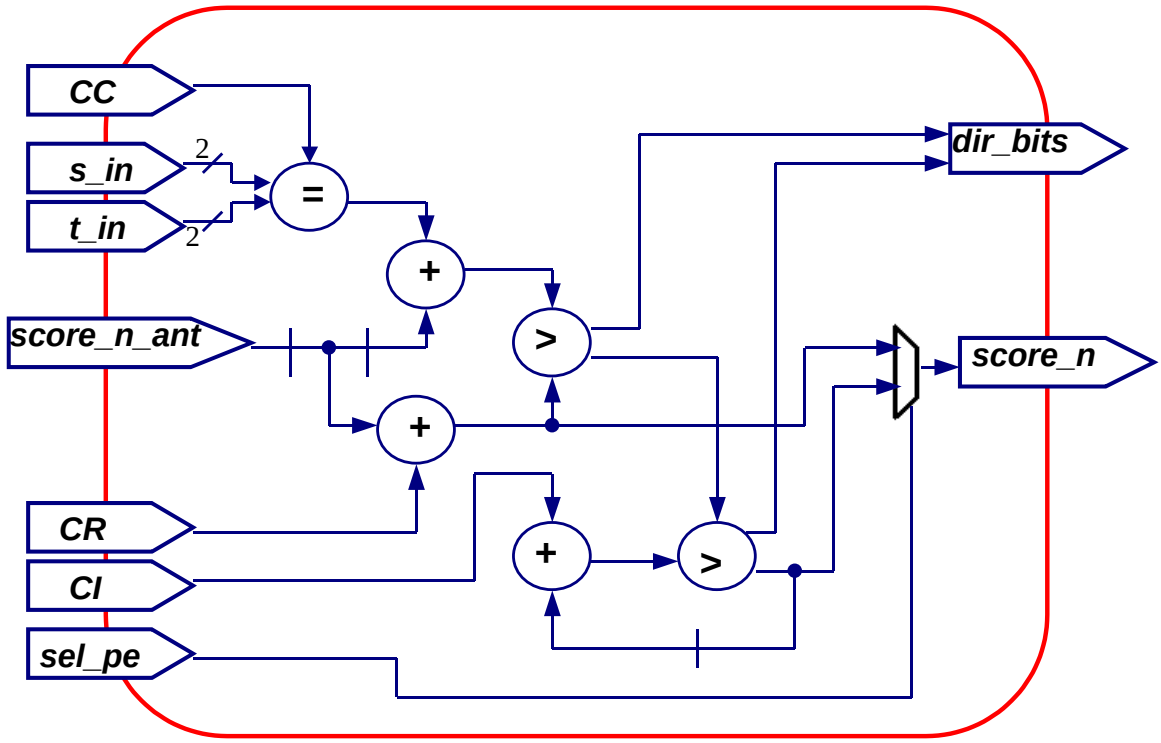
Neste caso específico, a estrutura representada na figura 4.10 depende da quantidade de recursos disponibilizados pelo FPGA da placa *(o Stratix III EP3SE260F da Altera)*. Inicialmente, precisamos determinar com estes recursos, que quantitativo máximo de *PEs* pode vir a ser implementado. A partir daí, definiremos a melhor relação entre a quantidade de Alinhamentos em paralelo e a quantidade de *PEs* a serem alocados em cada Array, objetivando o máximo desempenho.

Naturalmente, para que seja definido o número máximo de *PEs* a serem implementados, precisamos determinar que recursos serão necessários nesta construção. Na figura 4.12 apresentamos a arquitetura interna básica de um único *PE* e, como pode ser observado, são utilizados diversos somadores, registradores, comparadores de magnitude e multiplexadores.

A seguir, uma breve descrição dos sinais dos sinais do *PE* :

score_n_ant : Escore de entrada, recebido do PE anterior.
s_in: Entrada dos dados (nucleotídeos) da sequência ***"S".***
t_in: Entrada de dados (nucleotídeos) da sequência ***"T".***
CC: Parâmetro referente ao Custo de Comparação.
CI: Parâmetro referente ao Custo de Custo de Inserção.
CR: Parâmetro referente ao Custo de Custo de Remoção.
score_n: Escore de saída, gerado por este PE.
dir_bits: Bits de direção que apontam qual das 3 células (a de cima, a da diagonal ou a da esquerda) definiu o escore gerado pelo PE.

figura 4.12 – **Arquitetura básica do PE**



Fonte: Elaborada pelo autor

Por ser básica, a estrutura apresentada na figura 4.12 não explicita algumas intervenções que objetivaram um ganho maior de desempenho. A principal delas foi a inserção de quatro estágios de *pipeline*, assim, aumentando a frequência máxima do *Clock*. O custo desta estratégia, no entanto, é representado por uma *latência* de 4 ciclos, apenas, na geração do primeiro *escore* de saída de cada PE.

O *PE* foi descrito na linguagem *Verilog* e sintetizado na ferramenta *Altera Quartus II* (QUARTUS,2014) versão 13.0. Esta *IDE (integrated development environment)* gera o arquivo *(bitstream)* responsável pela programação da arquitetura no *FPGA*. A partir do relatório emitido pelo *Quartus,* após a síntese de um *PE,* conclui-se que, teoricamente, haveria recursos suficientes para a implementação de até 1296 *PEs* no dispositivo utilizado no projeto. Este valor é, apenas, uma referência inicial, uma vez que, na medida em que novas instancias destes blocos de processamento são criadas, a ferramenta otimiza o consumo de recursos possibilitando a síntese de uma quantidade de *PEs* superior a este valor inicial. Além disto, a quantidade final de *PEs* implementadas, ainda, irá depender da demanda de recursos exigidos na síntese dos demais blocos da arquitetura.

Com base nesta informação, buscou-se a melhor relação entre o número de Alinhamentos paralelizados e a quantidade *PEs* alocados por Array (ou Unidade de Alinhamento) objetivando a maximização do desempenho.

Como as sequências, normalmente, são longas, dificilmente, haverá recursos suficientes num *FPGA* que possibilitem a construção das *Matrizes PD*, em um único processo. Sendo assim, a solução adotada foi a utilização do paradigma *divide and conquer*, através do qual, dependendo da natureza do algoritmo, o problema é sub-dividido em instancias menores que , ao final, irão compor a solução completa. Ou seja, o problema é dividido em subproblemas que, no caso, significa a divisão ou o particionamento da *Matriz PD* em submatrizes menores, às quais chamaremos de *slices.* O tamanho do *slice* irá depender do número de *PEs* alocados por Array Sistólico. Na figura 4.13 temos um exemplo simples de uma Matriz que recebe sequências pequenas com tamanho igual a 12. Considerando a disponibilização de, apenas, 4 PEs para cada Array, a construção da matriz terá que ser particionada em 3 slices.

figura 4.13 – **Particionamento da Matriz PD**

PE1 PE2 PE3 PE4 PE1 PE2 PE3 PE4 PE1 PE2 PE3 PE4

	-	s1	s2	s3	s4		s5	s6	s7	s8		s9	s10	s11	s12
-	0	-1	-2	-3	-4		-5	-6	-7	-8		-9	-10	-11	-12
t1	-1														
t2	-2														
t3	-3														
t4	-4														
t5	-5	Slice_01					Slice_02					Slice_03			
t6	-6														
t7	-7														
t8	-8														
t9	-9														
t10	-10														
t11	-11														
t12	-12														

Fonte: Elaborada pelo autor

A divisão do processo de construção da matriz introduz um “retardo” ou um *overhead*, o qual é inversamente proporcional ao número de *PEs* alocados/*Array*. A explicação para isto é que durante a geração de um *slice*, na medida em que os

primeiros *PEs* concluem a geração de sua coluna, eles entram em "estado de espera", aguardando que o último *PE* conclua o seu processo. Esta "ociosidade" temporária aumenta o número total de ciclos necessários à construção de uma matriz. A solução seria diminuir o número de *slices*, aumentando o número de *PEs/Array.* No entanto, devido à quantidade limitada de recursos, teríamos que reduzir o número de Unidades de Alinhamento, em paralelo. Diante disto, é necessário identificar o ponto-ótimo da curva que relaciona o desempenho da arquitetura em função do número de *PEs* alocados por *Array* e do número de Unidades de Alinhamento.

A construção desta curva é iniciada pela expressão que define, em número de ciclos, o tempo de construção de um *slice*. Neste cálculo foi levado em conta o número de ciclos necessários ao "carregamento" dos nucleotídeos da sequência ***"S"*** no *Array (ver figura 4.9)* e os 4 ciclos de *latência* na saída de cada *PE* devido ao *pipeline*. Assim, o tempo de construção de um *slice* é definido por:

tempo_slice*(em ciclos)* **= 4.NPE + m – 1 + NPE = 5.NPE + m – 1**

sendo: *NPE = número de PEs / Array*

m = tamanho da sequência ***"T"***.

Por sua vez, o tempo de construção de uma Matriz será, então:

tempo_matriz*(em ciclos)* ***= nr. de slices x tempo_slice***

tempo_matriz*(em ciclos)* ***= (n / NPE) x (5.NPE + m – 1)***

sendo: *n = tamanho da sequência* ***"S".***

Considerando que os tamanhos de ambas as sequências seja igual a *"n"* e que este valor é muito maior que 1, teremos:

tempo_matriz*(em ciclos)* ***= 5n + (n^2 / NPE)***

Com o *FPGA* operando em uma determinada frequencia f_{Clock}, o número de Matrizes geradas por segundo por um único *Array*, seria:

nr_matrizes / s / Array = f_{Clock} / (5n + (n^2 / NPE))

Considerando, finalmente, *tot_PE* como sendo o número total de *PEs* em uso e distribuídos, equitativamente, entre todos os *Arrays*, ver figura 4.10, o número total de Matrizes geradas pela arquitetura seria:

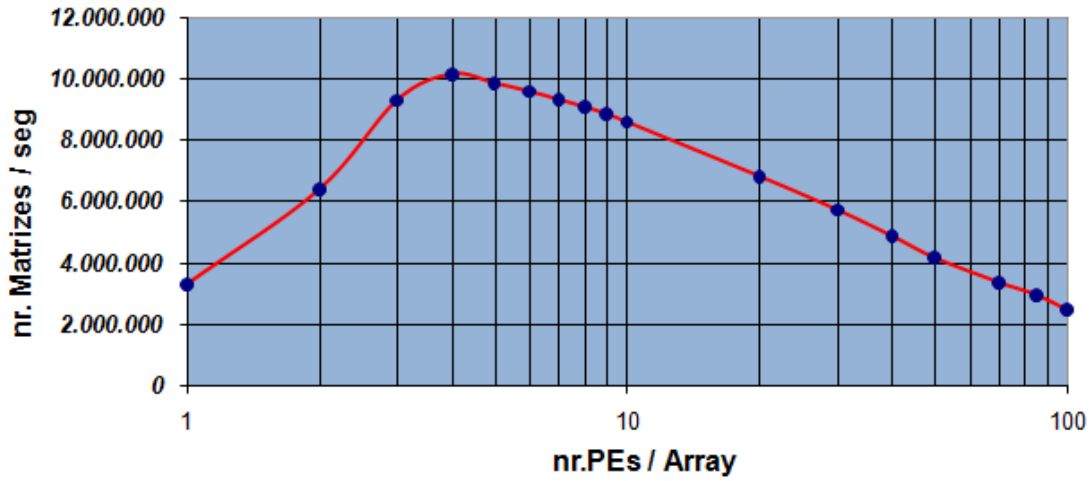
nr_total_matrizes / s = nr_matrizes/s/Array x nr_Arrays

nr_total_matrizes / s = (f_{Clock} / (5n + (n^2 / NPE))) x (tot_PE / NPE)

$$nr\_total\_matrizes / s = (f_{Clock} . tot\_PE) / (5.n. NPE + n^2) \quad \text{eq. 4.3}$$

Na figura 4.14 temos um gráfico-exemplo para a eq. 4.3 que relaciona o número total de matrizes geradas/segundo com o número de *PEs* alocados por *Array*. Na construção do gráfico foi considerado a seguinte configuração: o comprimento das sequências igual a 150, a frequência do *Clock* igual a 200 MHz, o limite máximo de 1296 *PEs* e o número máximo de *Arrays*, em paralelo, igual a 384. Este último valor, corresponde ao número máximo de alinhamentos, em paralelo, dado pela largura máxima do canal de entrada do *multiport* (768 bits) dividido pelos 2 bits usados na codificação dos nucleotídeos.

figura 4.14 – **Nr.matrizes produzidas x Nr. PEs/Array**



Fonte: Elaborada pelo autor

O baixo desempenho na fase inicial da curva deve-se à baixa utilização do quantitativo de PEs disponíveis em função do número máximo de entradas no canal do *multiport*. A partir daí, o desempenho aumenta com o progressivo aproveitamento destas unidades de processamento. Na configuração escolhida, o pico ocorre com 4 *PEs/Array* devido ao aproveitamento dos 1296 *PEs* disponíveis e a utilização de 324 *Arrays* em paralelo. A partir deste ponto, a taxa de geração das matrizes diminui, progressivamente, devido à redução na quantidade de *Arrays* ou de Unidades de Alinhamento e, portanto, na exploração do paralelismo *coarse-grained*. Ou seja, apesar do aumento no número de *PEs/Array* e, portanto, da maior exploração do paralelismo *fine-grained*; a exploração do paralelismo *coarse-grained* é bem mais eficiente no desempenho geral da arquitetura.

Pode ser observado, na figura 4.14**,** que a utilização de 4 PEs/Array nos permite atingir, teóricamente, uma taxa superior a 10 milhões de matrizes/segundo geradas no *FPGA*, o que representa a seguinte taxa em *CUPS*:

150 x 150 x 10 x 10^6 = 225 GCUPS

O *CUPS*, como já afirmado, é a métrica de desempenho, comumente, usada neste tipo de aplicação. Sendo assim, para gerarmos a informação do desempenho em *GCUPS* , usaremos a seguinte relação:

$$\textbf{\textit{desempenho}}(\ GCUPS\ ) = \textbf{nr.matrizes/s} \times \textbf{tamanho\_matriz} \times \textbf{10}^{-9}$$

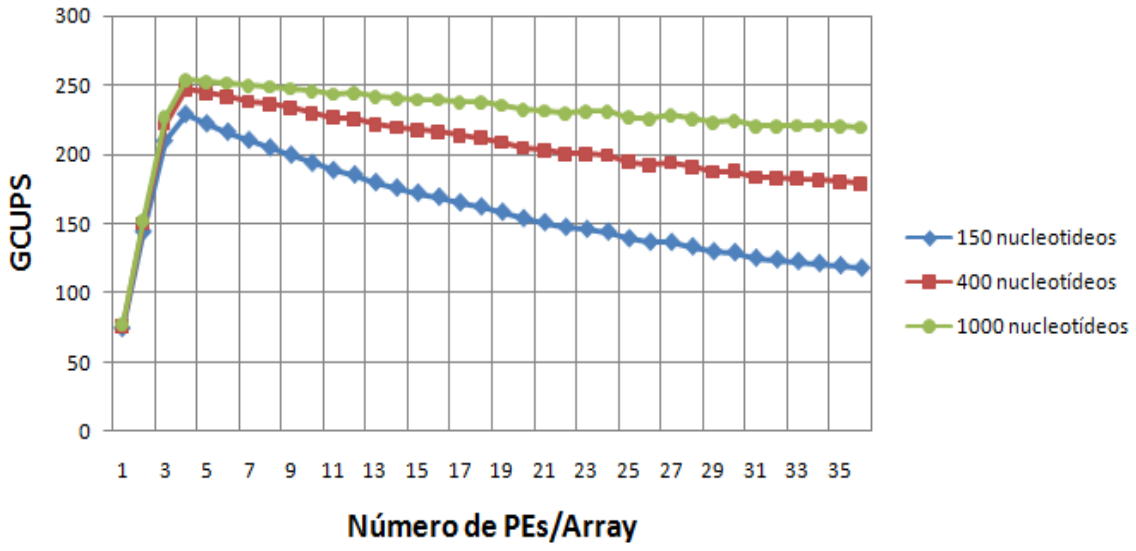
Utilizando a eq. 4.3 , teremos:

$$\textbf{\textit{desempenho}}(\ GCUPS\ ) = [(\ f_{Clock}\ .\ tot\_PE\ )\ /\ (5.n.\ NPE + n^2\ )] \times n^2 \times 10^{-9} \qquad \textbf{eq. 4.4}$$

A figura 4.15 apresenta 3 curvas de desempenho para tamanhos distintos de sequências de *DNA*. Novamente, a frequência do Clock foi de 200 MHz, o limite máximo de 1296 PEs e o número máximo de 384 Arrays.

Pode ser observado que, em todos os casos, o desempenho é máximo na configuração de 4 PEs/Array e os motivos são os mesmos apontados para o gráfico da figura 4.14; ou seja, na fase inicial da curva ocorre uma baixa utilização dos PEs disponíveis e, a partir de 4 PEs/Array, o número de Unidades de Alinhamento diminui, progressivamente.

figura 4.15 – **Desempenho (em GCUPS) x Nr. PEs/Array**



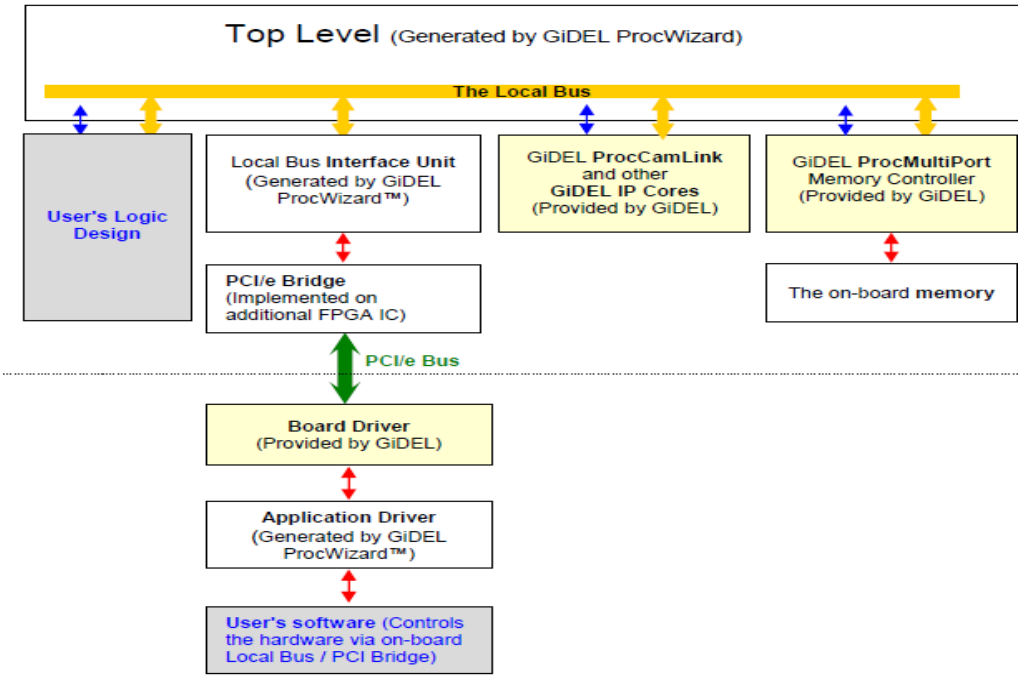
Fonte: Elaborada pelo autor

A escolha de 4 PEs/Array é experimental servindo, apenas, como base na construção da arquitetura do projeto. Mais adiante, faremos uma reavaliação neste número; bem como, no quantitativo total de Unidades de Alinhamento.

4.6 GiDEL *ProcWizard*

Como já mencionado, a placa aceleradora *PROCe III* utilizada no projeto é fornecida pela *GiDEL Ltd*., que disponibiliza em seu ambiente de projeto a ferramenta *ProcWizard* (PROCWIZARD,2014). Esta ferramenta permite a integração do algoritmo implementado no *FPGA* aos recursos disponíveis na placa e à aplicação em execução no *host* (denominada *camada de software).* Ver figura 4.16.

figura 4.16 - **Estrutura do GiDEL ProcWizard.**



Fonte: GIDEL[2],2011

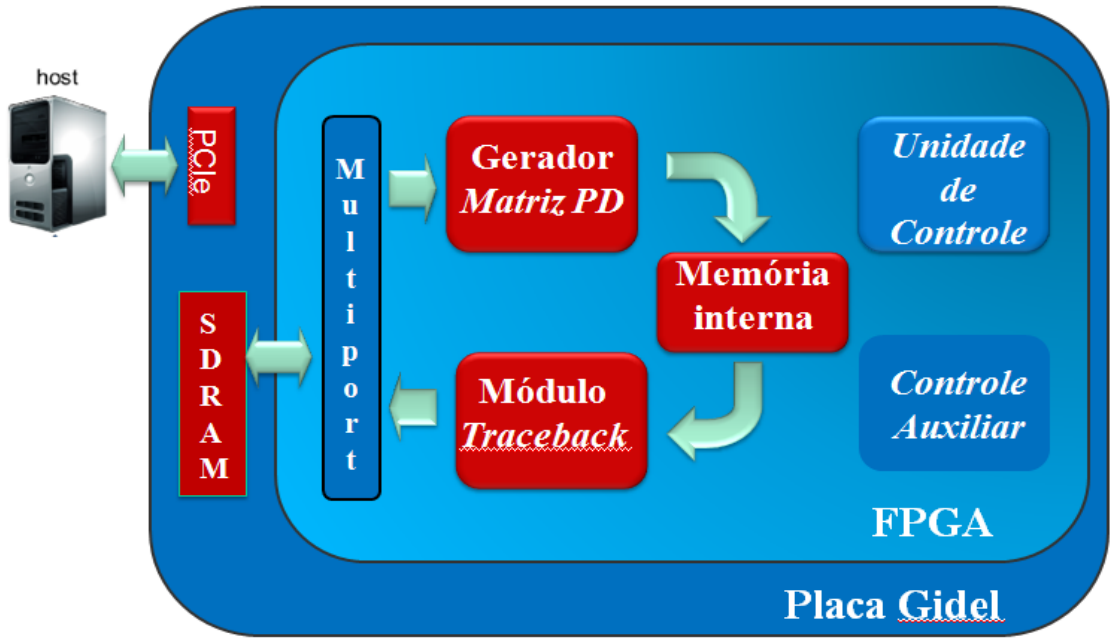
O processo de integração é iniciado a partir de uma *GUI (Graphical User Interface)* que permite a configuração do projeto a ser embarcado na placa, instanciando o seu módulo *top,* as memórias, através de seus *multiports*, e definindo os registradores de controle e outros parâmetros. Em seguida, a ferramenta está pronta para gerar os arquivos referentes ao *hardware* e ao *software*. O arquivo referente ao *hardware* representa um projeto no *Quartus II* com todos os arquivos de configuração e códigos em *HDL* dos módulos a serem sintetizados no *FPGA*.

Por sua vez, o arquivo gerado referente ao *software* possui uma extensão *.h* contendo as especificações de uma Classe em C++ que representa, na *camada de software*, a placa aceleradora. Os objetos desta Classe referenciam uma *API* da *GiDEL (denominada ProcAPI)* que, a partir de chamadas à mesma, permite o gerenciamento e a transferência de dados para a placa. A *GiDEL*, também, fornece os *drivers* do barramento *PCIe.*

4.7 Arquitetura Interna no *FPGA*

A figura 4.17 apresenta uma visão geral da plataforma considerando apenas, um só Alinhamento. O Algoritmo é executado na placa, a partir de uma chamada na *camada de software*, no *host*. As sequências são transferidas, via *DMA*, para as Memórias *SDRAM* da placa, através da interface PCIe 2.0 4x. A interface *hardware/software* foi detalhada no ítem anterior.

figura 4.17 **- Visão Geral da Plataforma (para 1 Alinhamento)**



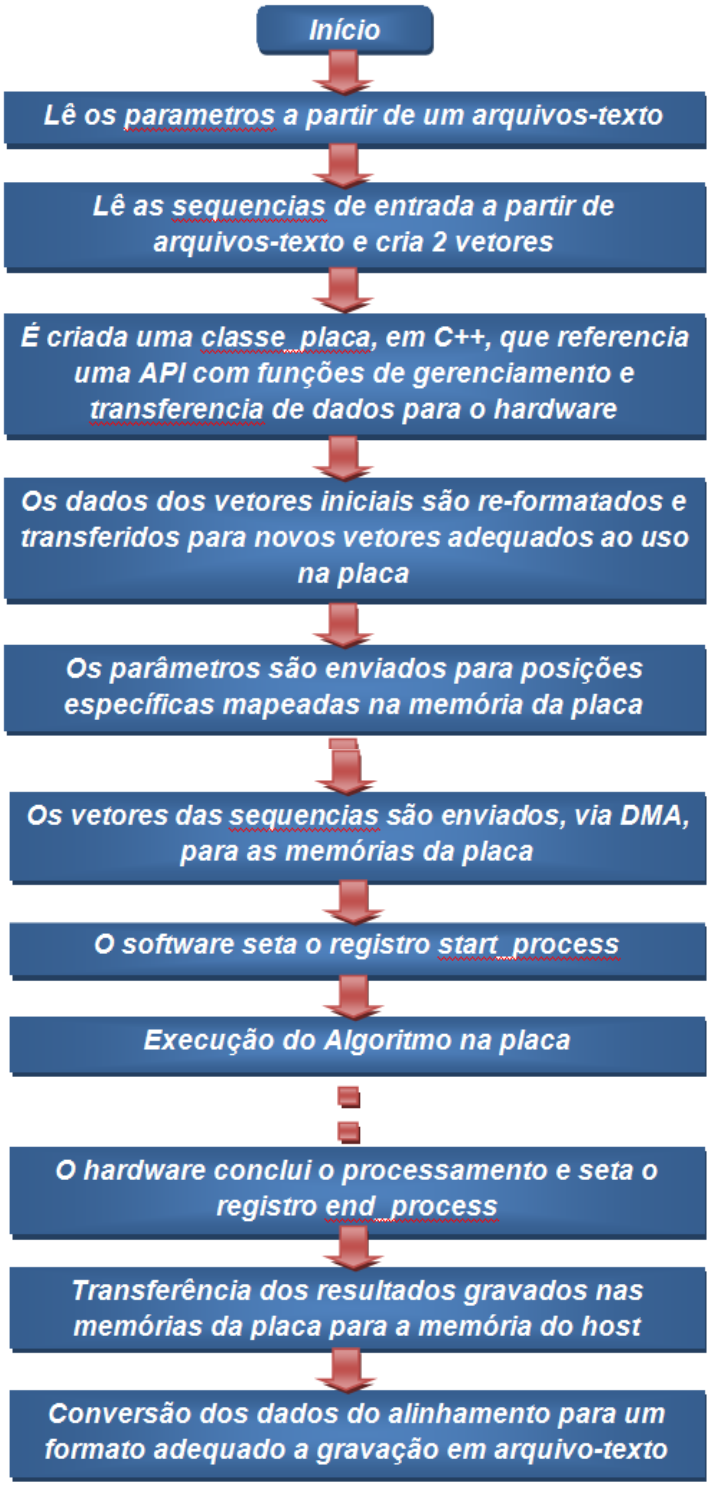
Fonte: Elaborada pelo autor

Toda a estrutura de *hardware* é abstraída pelo programador cabendo a ele, apenas, a inserção das funções da *API*, criada pela ferramenta de integração *hardware/software*, no programa a ser utilizado pelo usuário do sistema. Um pré-processamento ocorre na *camada de software*, mas, cabe à *camada de hardware* a efetiva execução do algoritmo.

Na figura 4.18 é apresentado o detalhamento do fluxo de execução do algoritmo, do ponto de vista, da *camada de software.*

Na sequência, é apresentado todo o fluxo dos dados na *camada de hardware* e a descrição operacional de seus módulos.

figura 4.18 **- Fluxo de Execução do Algoritmo *(camada de software)***

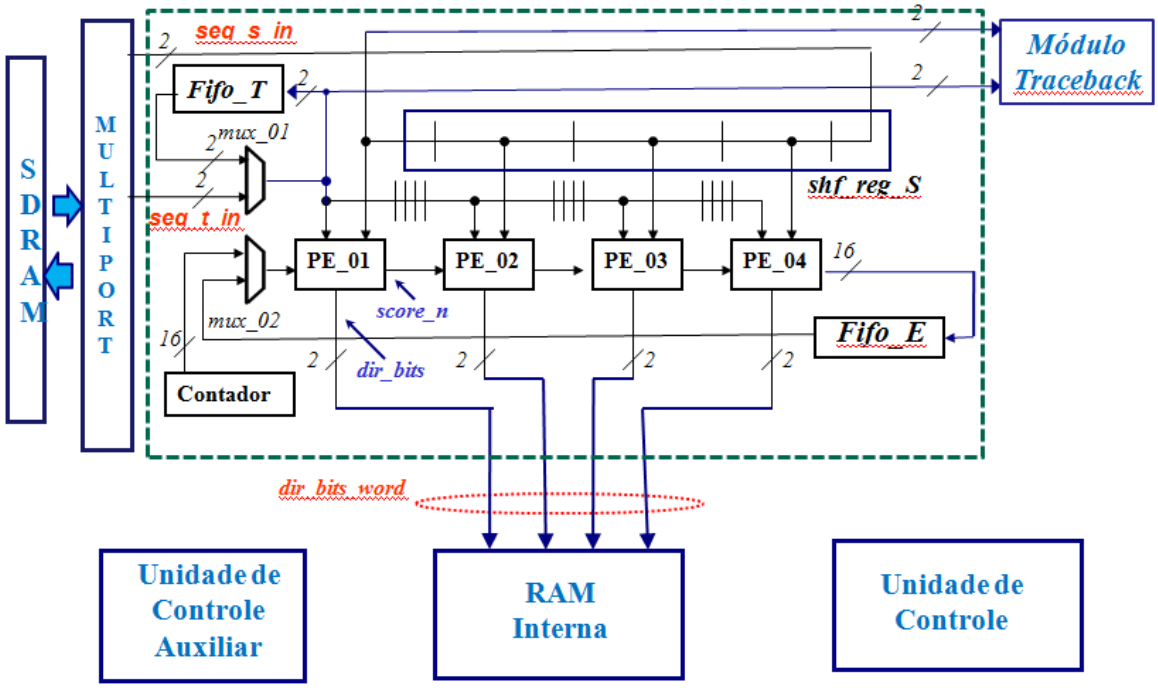


Fonte: Elaborada pelo autor

4.7.1 Módulo Gerador da Matriz PD

A fase inicial do Algoritmo é implementada no Módulo Gerador da Matriz *PD*, ver linha tracejada na figura 4.19. O principal elemento deste módulo é o Array Sistólico que, provisoriamente, será constituído por quatro *PEs*.

figura 4.19 **- Módulo Gerador da Matriz PD**



Fonte: Elaborada pelo autor

Inicialmente, apenas, a primeira linha do *Slice_01* da matriz é gerada, ver figura 4.12. Esta inicialização ocorre a partir de um valor igual a zero na saída do Contador, o qual, através do *mux*_02, é injetado na entrada *score_n_ant* do PE_01, ver figura 4.11. Nesta fase inicial, a Unidade de Controle estabelece que a saída dos PEs seja igual a ***score_n_ant + CR*** *(Custo da Remoção), onde CR* é um parâmetro do projeto que, em nosso caso, foi atribuído o valor -1. Dessa forma, a saída *score_n* do PE_01 será igual a -1, a saída do PE_02 será igual -2 e, assim, sucessivamente.

Em seguida, será iniciada a geração dos demais escores da *Matriz PD*. A seguir, uma breve descrição das principais etapas desta fase:

1. A construção do *slice_01* é iniciada pela leitura, a partir da saída *seq_s_in* do *Multiport,* dos 4 primeiros dados da sequência "S". Estes dados são,

sequencialmente, recebidos e distribuídos pelo *shf_reg_S* nas respectivas entradas *s_in* de cada PE.

2. Em seguida, os dados da sequência "T" são, sequencialmente, lidos a partir da saída *seq_t_in* do *Multiport* e injetados, via *mux_01*, no PE_01. Neste mesmo ciclo, o Contador inicia uma contagem negativa decrescente e, através do *mux_02,* os seus valores são injetados na entrada *score_n_ant* do PE_01. Na verdade, cabe ao Contador a geração da primeira coluna da *Matriz PD*.
3. Os dados de "T", também, são injetados na *Fifo_T*, para o seu futuro reuso na construção dos próximos *slices.* Estes dados são, ainda, injetados na *Lifo_T* do módulo *Traceback*.
4. Devido à presença do *pipeline* interno de 4 estágios nos PEs, foi necessária a inserção de 4 registradores entre eles para garantir a sincronização na chegada dos dados em suas entradas.
5. Além da geração do primeiro escore, o PE_01, também, gera os dois primeiros bits de direção, *dir_bits*. Quando o PE_4 tiver gerado os seus *dir_bits*, a primeira *dir_bits_word* com 8 bits terá sido obtida e será gravada na RAM interna.
6. Os escores gerados pelo PE_04 serão, consecutivamente, gravados na *Fifo_E*. Quando o último escore for gravado, significa que o primeiro *slice* foi concluído.
7. O segundo *slice* é iniciado com a leitura de mais quatro dados de *"S"*. Na medida em que estes dados são deslocados no *shf_reg_S*, eles substituirão os dados anteriores, os quais serão carregados na *Lifo_S*, no módulo *Traceback*.
8. A geração dos *slices* seguintes, segue a mesma sequência do primeiro, porém, com 2 diferenças:

   a) Caso a sequência *"T"* seja a *Query* que está sendo alinhada com as sequências de um determinado banco, ela será reusada. Ou seja, a seqüência *"T"* será lida a partir da *Fifo_T* e, não mais da memória externa.

   b) Os escores, anteriormente, gravados na *Fifo_E* serão injetados, através do *mux_02*, no PE_01.

4.7.2 Módulo *Traceback*

A fase final do algoritmo, o *Traceback,* já apresentada, é iniciada a partir da célula mais inferior à direita na *Matriz PD* e segue em direção à célula (0,0) guiada pelas setas de direção, às quais foram definidas pelos *dir_bits*, ver figura 4.20.

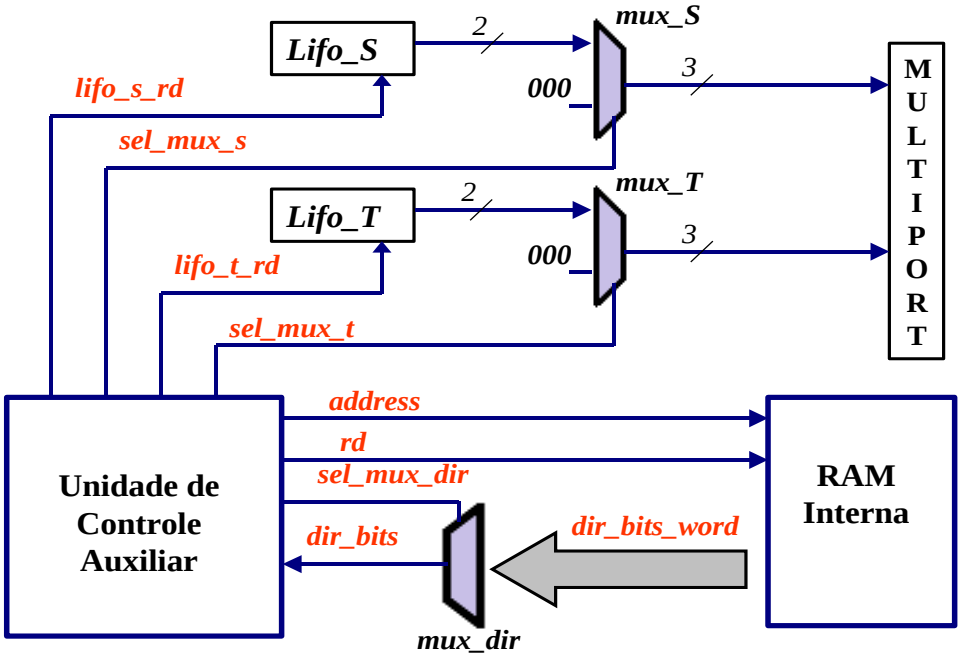
figura 4.20 **- Exemplo de Traceback**

	-	A	A	C	G	T	T	A	C
-	0	-1	-2	-3	-4	-5	-6	-7	-8
C	-1	-1	-2	-1	-2	-3	-4	-5	-4
G	-2	-2	-2	-2	0	-1	-2	-3	-4
A	-3	-1	-1	-2	-1	-1	-2	-1	-2
T	-4	-2	-2	-2	-2	0	0	-1	-2
A	-5	-3	-1	-2	-3	-1	-1	+1	0
A	-6	-4	-2	-2	-3	-2	-2	0	0
C	-7	-5	-3	-1	-2	-3	-3	-1	+1

Fonte: Elaborada pelo autor

Conforme descrito, durante a fase de construção da matriz, a cada escore gerado pelos *PE_s*, 2 bits de direção eram criados. Assim que os 4 PEs compunham a *dir_bits_word*, com 8 bits, ocorria a gravação na RAM interna. O *Traceback* é iniciado, então, pela leitura do último *dir_bits_word* gravado *(ver figura 4.21).*

figura 4.21 **- Módulo do Traceback**



Fonte: Elaborada pelo autor

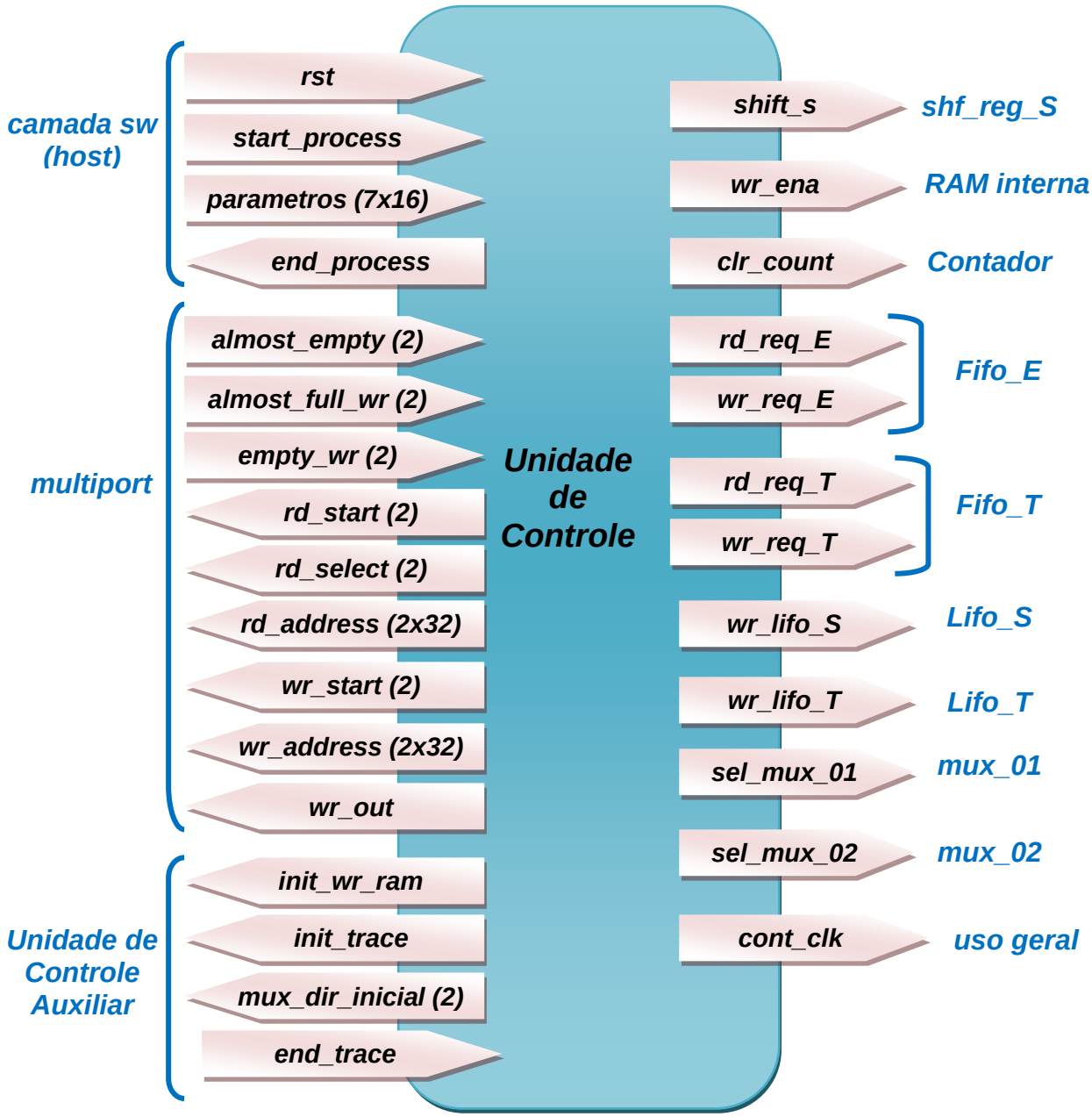
A Unidade de Controle Auxiliar, doravante *UC_Aux*, efetua a leitura da última *dir_bits_word* gravada na RAM. Os dois bits menos significativos desta palavra representam os bits de direção da última célula da Matriz. A leitura destes bits leva a 3 possíveis situações:

1. Os bits indicam que a seta aponta para baixo. Neste caso, o caminho do *traceback* seguirá para a próxima célula de cima na mesma coluna. Na saída dos alinhamentos será inserido um espaço em branco *(gap)* na sequência "S". Para isto, no esquema da figura 4.19, a *UC_Aux* efetuará as seguintes ações:
   a) O sinal *lifo_s_rd* não será ativado e o *sel_mux_s* irá selecionar para a saída do *mux_S* a entrada *"000".* Este dado representa um *gap* na sequência *"S"* e o mesmo será gravado no *Multiport*.
   b) O sinal *lifo_t_rd* será ativado e o *sel_mux_t* irá selecionar para a saída do *mux_T* a entrada conectada à *Lifo_T.* Com isto, no *Multiport* serão gravados 3 bits. O mais significativo é igual a "1" e, os outros dois, serão os bits do último dado gravado na *Lifo_T*; ou seja, o último dado de "T".
2. Os bits indicam que a seta aponta para a direita. Neste caso, o caminho do *traceback* seguirá para a próxima célula à esquerda, na coluna anterior. Na saída dos alinhamentos será inserido um *gap* na sequência "T". Para isto, a UC Auxiliar deverá atuar de forma, exatamente, oposta à anterior.
3. Os bits indicam que a seta aponta para a diagonal. Este caso representa ou um *match* ou um *mismatch* e o caminho do *traceback* segue para a próxima célula de cima na coluna anterior. Assim, os dados das duas sequências serão enviados para as saídas dos alinhamentos e gravados no *Multiport*. Para isto, a *UC_Aux* deverá acionar ambos os sinais *lifo_s_rd* e *lifo_t_rd* e selecionar como entrada dos multiplexadores, as saídas dessas LIFOs.

4.7.3 Unidade de Controle

Os sub-módulos que compõem o Módulo Gerador da Matriz PD, na figura 4.18, são controlados por sinais gerados pela Unidade de Controle (doravante *UC*). A lógica embutida nesta unidade possibilita desde o acionamento da leitura das sequências na Memória da placa e o seu carregamento nas FIFOs do *Multiport,* passando por todo o processo de construção da *Matriz PD*, até a fase do *Traceback*, onde ocorre uma atuação conjunta com a *UC_Aux*. A figura 4.21 apresenta todos os sinais da *UC*.

figura 4.22 **- Sinais da Unidade de Controle**



Fonte: Elaborada pelo autor

A seguir, detalhes dos principais sinais:

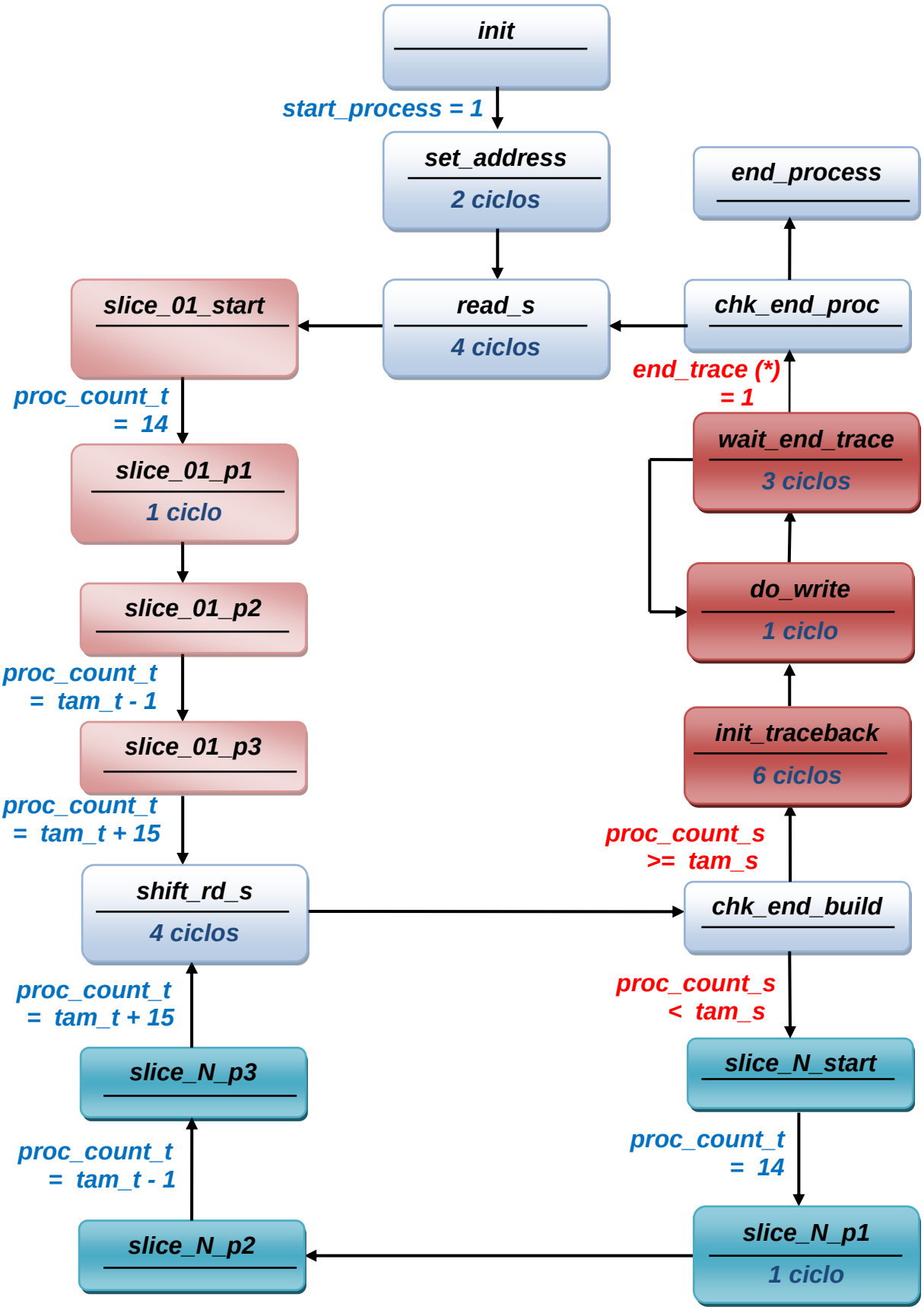
- ***parametros****:* A aplicação na camada de *software* define, em tempo de compilação, sete parametros para arquitetura: o tamanho de cada sequência, o número de alinhamentos consecutivos e os endereços de leitura e de escrita, de ambas as sequências.

- ***end_process:*** reporta à aplicação que o processamento, na camada *hardware*, já está encerrado. Ou seja, os arquivos com as sequências alinhadas, já podem ser transferidos para a memória do *host*.
- ***almost_empty***, ***almost_full*****,** ***empty_wr***, ***rd_start***, ***rd_select, wr_out*** , ***rd_address, wr_start***, ***wr_address***: representam sinais de comunicação e controle entre o *hardware* construído e as FIFOs do *Multiport*.
- ***init_wr_ram:*** sinal enviado à UC Auxiliar para que, a mesma, passe a gerar os endereços de gravação da RAM interna.
- ***init*_trace**, **end**_***trace***: sinais de controle entre a UC Auxiliar e a Unidade de Controle, referentes ao início/fim de um *traceback*.
- ***shift_s***: sinal responsável pelo deslocamento de 4 ciclos dos dados da sequência *"S"* no *shf_reg_S*.
- ***wr_ena***: sinal de gravação da *dir_bits_word* na RAM interna.
- ***cont_clk***: este sinal é acionado sempre que faltarem dados na saída do *Multiport.* Esta informação é fornecida pelo sinal *almost_empty*; assim, a leitura será, temporariamente, suspensa. Para que não ocorra perda de sincronismo entre os diversos módulos, a *UC* determina todos entrem em modo *"stand-by".*

Na figura 4.23 temos o Diagrama da Máquina de Estados referente à Unidade de Controle (UC). Por questões de simplificação, foram omitidos os estados referentes ao modo *"stand-by"*. A seguir, temos uma breve descrição dos estados:

- ***init:*** Estado inicial no qual o *hardware* encontra-se em estado de espera, aguardando a habilitação do sinal *start_process*, pela camada de *software.*
- ***set_address:*** A UC aciona o sinal *rd_start* do *multipor*t para que os dados das sequências gravadas na memória da placa, a partir dos endereços apontados pelo sinal *rd_address*, sejam "carregados" nas FIFOs do *multipor*t.
- ***read_s:*** Os 4 primeiros dados da sequência *"s"* são lidos e injetados nas entradas dos PEs, através do *shf_reg_S*.
- ***slice_01_start***: Inicia a leitura, sequencial, dos dados da sequência *"T"*. A cada novo ciclo de *Clock*, a variável *proc_count_t* é incrementada.
- ***slice_01_p1:*** A FIFO E inicia a gravação dos escores gerados pelo PE_04.

figura 4.23 - **Máquina-de-Estados da Unidade de Controle**



Fonte: Elaborada pelo autor. ***(*) sinal gerado pela UC_Aux.***

- ***slice_01_p2:*** A RAM interna começa a gravação do *dir_bits_word.* Este estado permanece até que seja concluída a leitura da sequência *"T"*.
- ***slice_01_p3:*** Segue a gravação da RAM e da FIFO E, até que o PE_04 gere o último escore do *slice_01 .*
- ***shift_rd_s:*** Os próximos 4 dados de *"S"* serão lidos e substituirão os 4 anteriores que estavam no *shf_reg_S*, os quais serão enviados para a LIFO S. A variável *proc_count_s* é incrementada em 4 unidades.
- ***chk_end_build:*** É verificado o encerramento do processo de construção da *Matriz E*. Esta informação é obtida a partir da variável *proc_count_s.*
- ***slice_N_start, slice_N_p1, slice_N_p2 e slice_N_p3:*** Estes estados são idênticos aos do *slice*_01, com a diferença que, serão usados os escores gravados na FIFO E e a sequência *"T"* será lida a partir da FIFO T.
- ***init_traceback*** *:* A UC habilita o sinal *init_trace*, o qual deflagra, na UC Auxiliar, a inicialização desta fase.
- ***do_write:*** Cabe a UC, em sincronização com a UC Auxiliar, acionar o sinal *write_out*, de gravação dos dados alinhados, na saída do *multiport*
- ***wait_end_trace:*** Ciclos de sincronização com a UC Auxiliar.
- ***chk_end_process:*** Verifica a conclusão dos Alinhamentos.
- ***end_process:*** Com o encerramento de todos os alinhamentos, a UC habilita o sinal *end_process* para a *camada de software*.

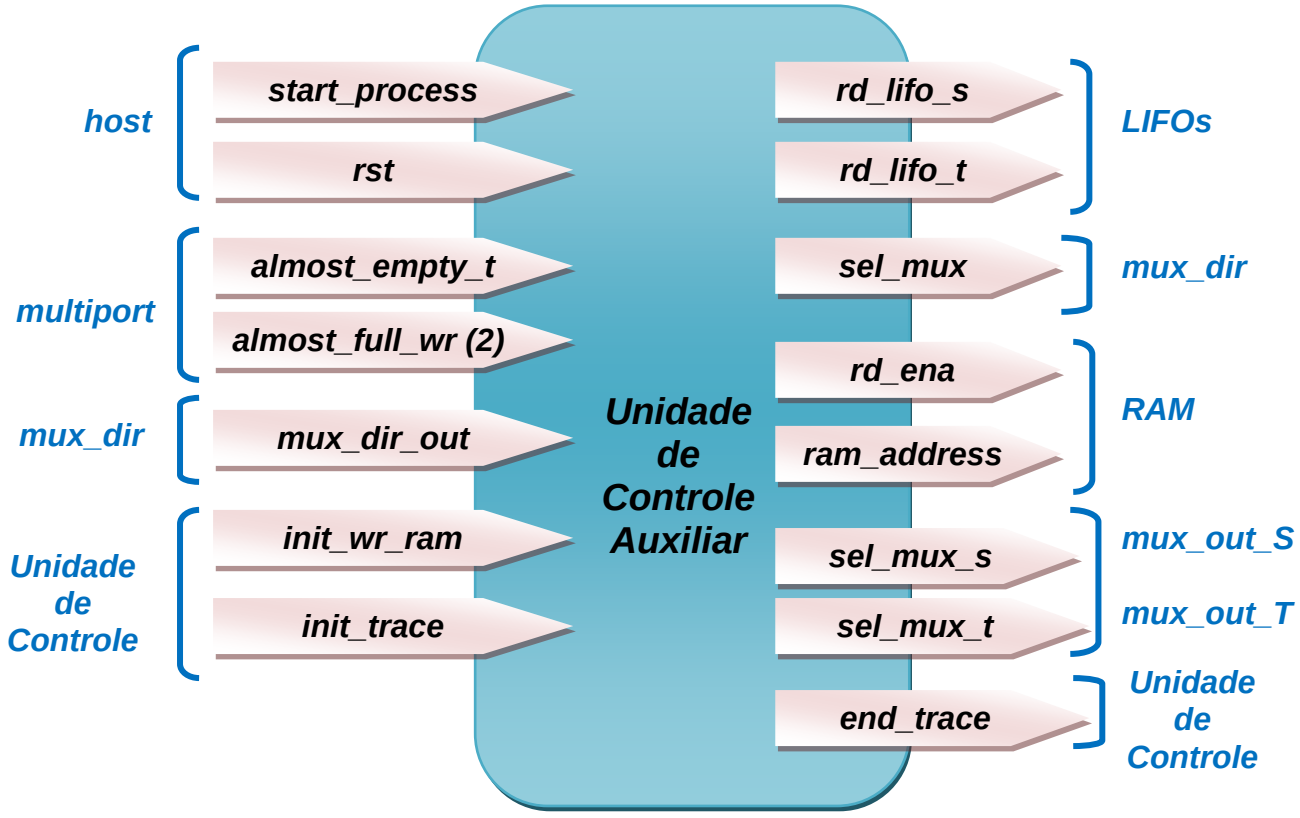
4.7.4 Unidade de Controle Auxiliar

A *UC_Aux*, cujos sinais estão representados na figura 4.24, não somente é a principal responsável pela fase final do Alinhamento, o *Traceback*, como, também, atua de forma sincronizada com a UC na fase de construção da *Matriz PD*. Esta sincronização ocorre através de sinais específicos, os quais, entre outros, serão descritos, a seguir:

- ***start_process:*** de forma idêntica à UC, aguarda a habilitação deste sinal para iniciar o processamento do algoritmo.
- ***almost_empty_t:*** A *UC_Aux* gera os endereços de gravação da RAM, durante à fase de construção da matriz; caso ocorra, algum interrupção na leitura dos dados de *"T"*, a *UC_Aux* entra em um modo *"stand-by"*.

- ***almost_full_wr:*** Informa se as FIFOs de saída do *multiport*, ainda, estão disponíveis para receber as respostas dos alinhamentos das sequências.
- ***mux_dir_out:*** Dado de saída do *mux_dir* que informa os *dir_bits* referentes à célula atual da matriz, durante o *traceback.*
- ***init_wr_ram:*** A UC, através deste sinal, informa, à *UC_Aux*, o início da gravação da RAM para que, esta última, passe a gerar os endereços.
- ***init_trace:*** Sinal de início do *traceback*, gerado pela UC.
- ***rd_lifo_s*** e ***rd_lifo_t:*** Sinais de leitura das LIFOs.
- ***sel_mux*** *:* Sinal de seleção do *mux_dir*.
- ***rd_ena:*** Sinal de leitura da RAM.
- ***ram_address****:* Endereço de escrita, gerado pela UC Auxiliar, durante a fase de construção da matriz.
- ***sel_mux_s*** e ***sel_mux_t:*** Sinais de seleção dos multiplexadores de saída da sequências alinhadas.
- ***end_trace:*** A *UC_Aux* informa a UC o término do *traceback.*

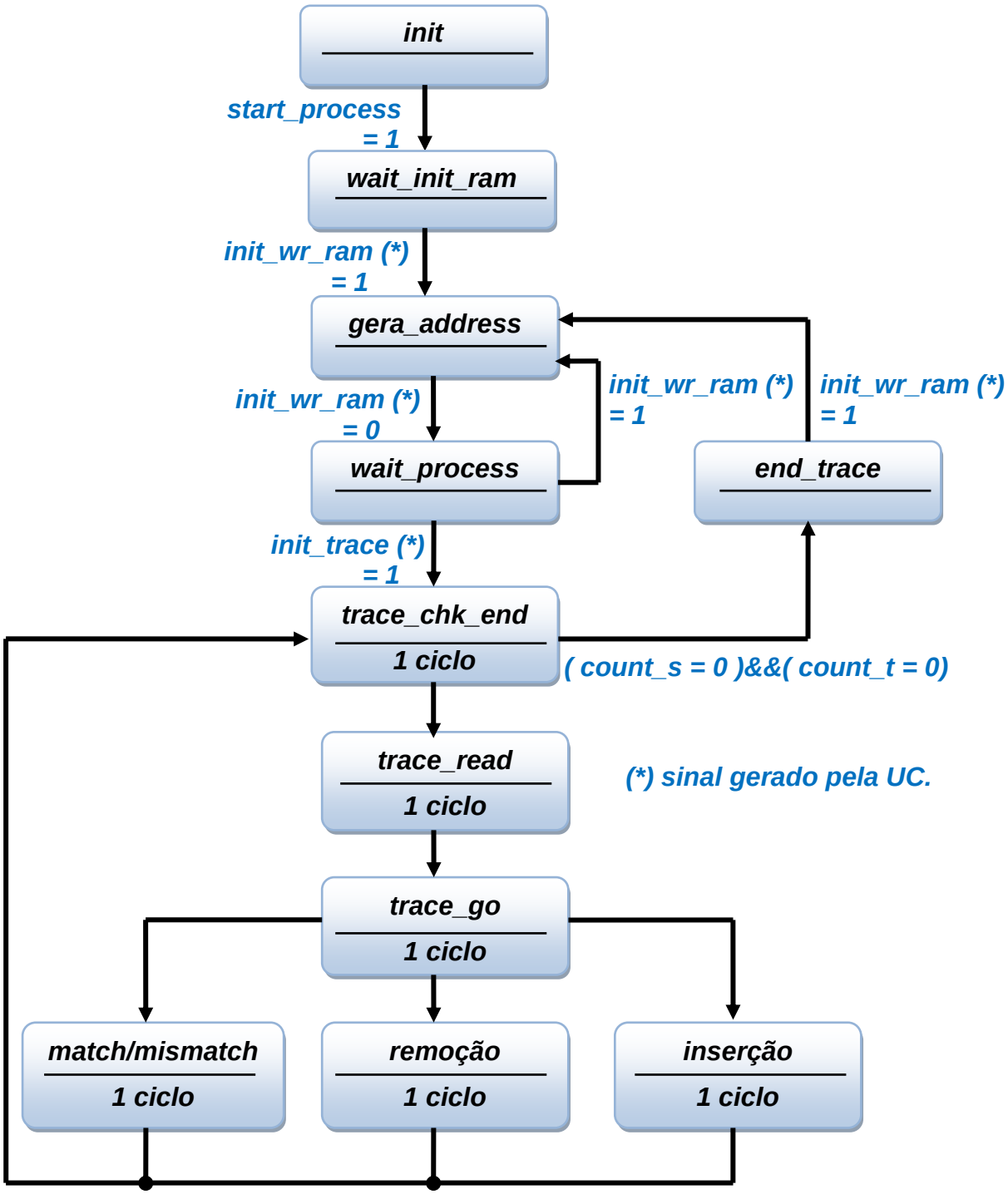
figura 4.24 **- Sinais da Unidade de Controle Auxiliar**



Fonte: Elaborada pelo autor

Na figura 4.25 temos o Diagrama de Estados da *UC_Aux*; nesta, também, foram omitidos todos os estados de espera *(stand-by),* os quais são acionados quando ou faltam dados nas FIFOs do *multiport*, quando num processo de leitura, ou quando as suas FIFOs de saída estão "cheias", durante um processo de escrita.

figura 4.25 **- Máquina-de-Estados da Unidade de Controle Auxiliar**



Fonte: Elaborada pelo autor

A seguir, é apresentada uma descrição sumária de cada um dos estados da *UC_Aux*, apresentados na figura 4.25:

- ***init:*** Estado inicial de espera pela habilitação do sinal *start_process*, pela camada de *software*, no *host*.
- ***wait_init_ram****:* A *UC_Aux* aguarda o sinal *init_wr_ram*, enviado pela UC, para gerar os endereços de gravação da RAM, na fase de construção dos *slices* da Matriz.
- ***gera_address:*** Geração dos endereços de gravação da RAM interna.
- ***wait_process:*** A *UC_Aux* entra em estado de espera aguardando a decisão da UC quanto ao término da fase de construção da matriz.
- ***trace_chk_end*** *:* Verificação do término do *traceback*.
- ***trace_read****:* A *UC_Aux* faz a leituras dos *dir_bits* na saída do *mux_dir* referente à célula atual.
- ***trace_go:*** A *UC_Aux* interpreta os *dir_bits* e define se a célula atual foi gerada: de um *match/mismatch*, de uma *remoção* ou de uma *inserção*.
- ***match/mismatch, remoção*** *e* ***inserção:*** Conforme já explicado, esses estados controlam as LIFOs, os multiplexadores de saída e a leitura da RAM, de forma a efetuar a construção das sequências alinhadas de saída, fazendo a inserção dos *gaps*, quando for necessária.
- ***end_trace*****:** Uma vez concluído o *traceback*, a *UC_Aux* entra em estado de espera pelo início da construção do primeiro *slice* do próximo alinhamento.

4.8 Arquitetura Completa

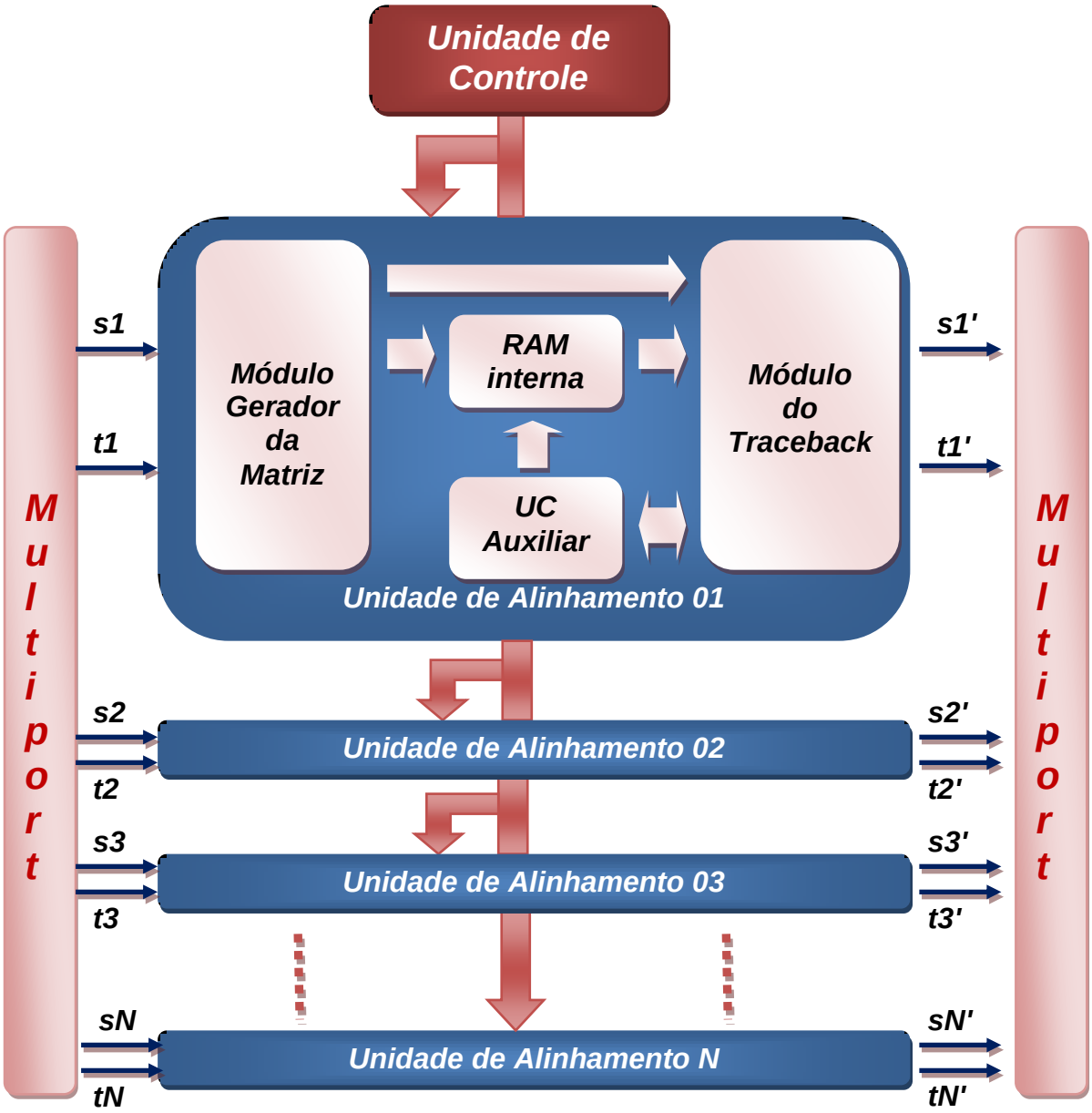
Toda a arquitetura apresentada até este ponto possibilita o Alinhamento de, apenas, duas sequências por vez. Assim, ao conjunto formado pelas micro-arquiteturas apresentadas nas figuras 4.19 e 4.21, chamaremos de Unidade de Alinhamento.

Com a inserção de outras Unidades de Alinhamento, em paralelo, iremos compor a Arquitetura Completa apresentada na figura 4.26, à qual possibilita o Alinhamento, simultâneo, de diversos pares de sequências.

A Unidade de Controle pode ser única, desde que todas as sequências *"S"* tenham o mesmo tamanho; assim como, todas as sequências *"T"*. Os tamanhos das

sequências *"S"* e *"T"* podem ser distintos. Assim, as *Matrizes PD* serão construídas, todas ao mesmo tempo, utilizando o mesmo número de ciclos de *Clock*.

figura 4.26 **- Arquitetura completa**



Fonte: Elaborada pelo autor

O próximo passo do projeto é a definição do número máximo de Unidades de Alinhamento a serem implementadas. Este número, obviamente, depende da quantidade de recursos disponibilizados no FPGA. No ítem 4.5 foi comentado que, à princípio, o dispositivo utilizado na placa possibilitaria a síntese de, no máximo, 1296 PEs. Assim, com 4 PEs/Array poderíamos, teoricamente, implementar até 324 Unidades de Alinhamento.

No entanto, a disponibilidade de recursos para a implementação da Memória interna surgiu como um fator, ainda, mais restritivo. De fato, a síntese da RAM interna, a partir das *macrofunções* do *Quartus 13,* faz uso dos blocos reconfiguráveis da memória SRAM embarcada. Para o dispositivo instalado na placa da GiDEL, o *EP3SE260F1152C2 d*a família *Stratix III* (ALTERA[3],2009), são disponibilizadas 3 tipos de blocos de memória:

a) 5.100 blocos do tipo MLAB, com 320 bits cada, otimizados para a implementação de FIFOs e *shift-registers.*
b) 864 blocos do tipo M9K, com 9 Kbits cada, para uso geral como memória.
c) 48 blocos do tipo M144K, com 144 Kbits cada, idealizados para o armazenamento de código, *bufferização* de pacotes e de *frames* de vídeo.

Dessa forma, foi feita a opção pelo uso dos blocos M9K e M144K para a síntese das RAMs, liberando os blocos do tipo MLAB para a síntese das FIFOs e dos *shift-registers.* Assim, o total de recursos disponibilizados para a implementação da Memória interna do FPGA ficou limitado ao seguinte quantitativo:

Total de bits nos blocos M9K = **864 x 9 kbits = 7.776 kbits**
Total de bits nos blocos M144K = **48 x 144 kbits = 6.912 kbits**
Total de bits disponibilizados pelos blocos M9K e M144K = **14.688 kbits**

Considerando sequências com um tamanho, por exemplo, igual a 250 *bps*, teríamos o seguinte número máximo de Unidades de Alinhamento:

Tamanho da Matriz = 250 x 250 = **62.500 células**
Número de bits de direção / célula = **2 bits**
Número de bits a serem armazenados / Matriz = 62.500 x 2 = **125.000 bits**
Número máximo de Unidades de Alinhamento =
= (Total de bits em M9K e M144K) / (Nr. bits armaz. / Matriz) **eq. 4.5**
= 14.688 x 1024 / 125.000 = 15.040.512 / 125.000
Número máximo de Unidades de Alinhamento **≈ 120 unidades**

Como a profundidade *(o número de células internas)* das memórias têm que ser um número múltiplo de dois, dependendo do tamanho das sequências, poderá ocorrer um baixo aproveitamento das mesmas. Assim, o total acima obtido seria

considerando um tamanho de sequência que possibilitasse a utilização de 100% da memória sintetizada.

Outro ponto importante é que no processo de construção de cada unidade de Memória interna, deve ser feita uma opção entre o uso ou dos blocos M9K ou dos blocos M144K. O tamanho máximo das sequências irá depender desta escolha. Caso fossemos implementar uma Unidade de Alinhamento utilizando todos os blocos M144K disponíveis e escolhendo a profundidade de 64K, que é a máxima disponibilizada no *Quartus* para este dispositivo, teríamos o seguinte valor para o tamanho máximo das sequências:

largura x profundidade **≤** Total de bits nos blocos M144K

largura x 64 x 1024 **≤** 6.912 x 1024 ⟹ largura **≤** 108

Como a memória armazena pares de bits de direção, então:

$largura_{máxima}$ = 108 bits ⟹ Nr. máximo de PEs = 54 PEs

Então, para determinar o tamanho máximo da sequência temos que considerar que a matriz inteira, com um tamanho igual a tam_máx_seq ^{2}, seria segmentada em trechos com uma largura igual a 54. Assim, teremos:

(tam_máx_seq)^2 = Nr. máximo de PEs x 64K = 54 x 65.536

tam_máx_seq = **1.881 *bps***

Então, para este tamanho máximo de sequência pode-se implementar até duas Unidades de Alinhamento. A memória utilizada por uma destas unidades consumiria todos os blocos M144K; enquanto que, a outra consumiria a grande maioria dos blocos M9K.

Caso as sequências sejam maiores que 1.881 *bps*, apenas, a fase 1 do algoritmo será executado no *FPGA*, conforme será explicado no ítem 4.9.

Devido à esta nova restrição de recursos deve ser verificado qual o quantitativo ideal de PEs/Array que irá garantir o máximo de desempenho. Para isto, utilizando-se a equação de desempenho (eq. 4.4), teríamos:

desempenho*(GCUPS)* **=**

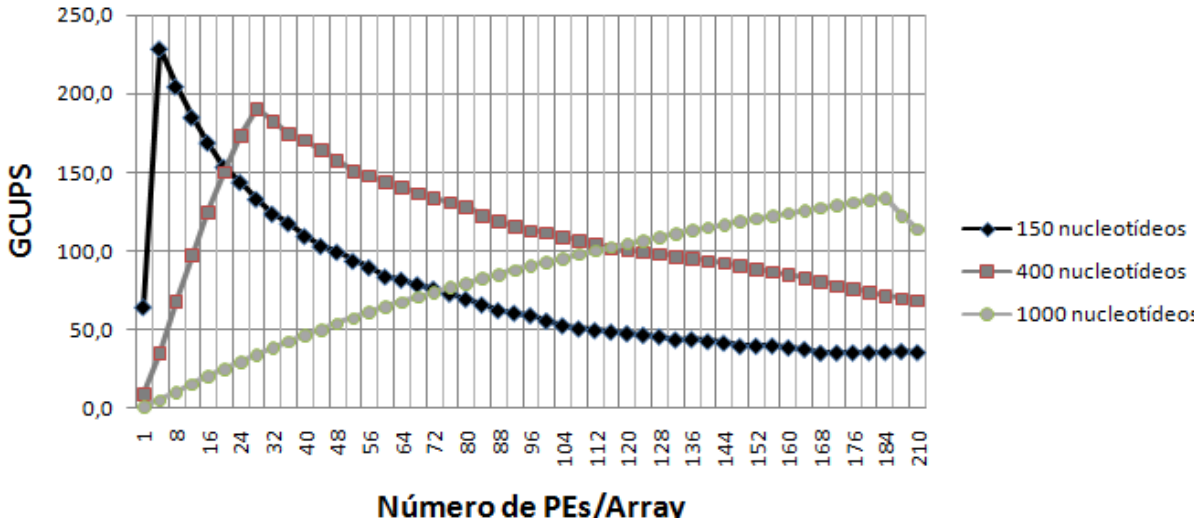
$$= [(f_{Clock} \,.\, tot\_PE) / (5.n.NPE + n^2)] \; x \; n^2 \; x \; 10^{-9} \qquad \textbf{eq. 4.4}$$

Neste novo cálculo, o número máximo de Unidades de Alinhamento será definido pelo tamanho das sequências, conforme estabelecido pela eq. 4.5.

A figura 4.27 apresenta as curvas de desempenho baseada na eq. 4.4 para três tamanhos distintos de sequências: 150, 400 e 1000 *bps*. Em cada caso, foi utilizado o número máximo possível de Unidades de Alinhamento.

Observa-se que conforme o tamanho das sequências aumenta e, portanto, a quantidade de Unidades de Alinhamento diminui, o pico de desempenho é, progressivamente, diminuído e para ser atingido exige quantidades maiores de PEs/Array.

Figura 4.27 **- GCUPS x Nr. PEs/Array**



Fonte: Elaborada pelo autor

Sendo assim, o usuário do sistema deverá ter uma noção do tamanho máximo das sequências a serem alinhadas e, assim, escolher a arquitetura com os respectivos número de PEs/Array que ofereça o maior desempenho possível. Para isto, deverão ser disponibilizados diversas opções de arquitetura de núcleo de processamento, baseado no número de PEs, previamente sintetizado.

A idéia, portanto, é a disponibilização de uma coleção de *templates* referentes a tamanhos máximos padrões de sequências para que, em tempo de compilação, o usuário possa escolher aquela configuração de arquitetura no *FPGA* que pode oferecer o máximo desempenho possível.

4.9 *Traceback* no *host*

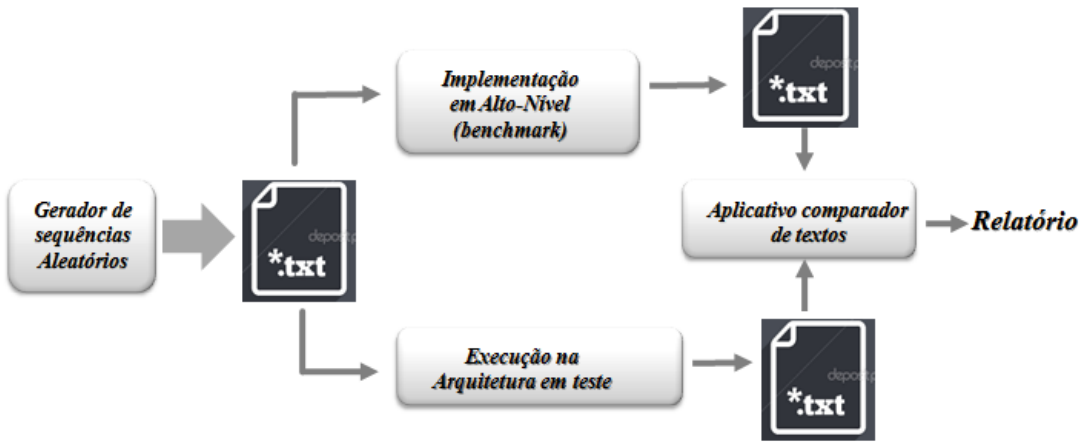
Caso o tamanho das sequências impliquem em um número reduzido de Unidades de Alinhamento devido às restrições de recursos de memória no *FPGA*, o desempenho poderá ser comprometido. A solução é a não utilização desta memória e, portanto, transferir a execução do *traceback* para o *host*. Assim, o número de Unidades de Alinhamento aumentaria, significativamente, e também o desempenho. Neste caso, os bits de direção *(word_dir_bits)* seriam enviados ao *host.*

4.10 Teste e Verificação

Seguindo o fluxo de projeto, a arquitetura em desenvolvimento seguiu para a fase de teste e verificação. Nesta, os resultados gerados pela mesma são comparados aos obtidos a partir de um Modelo de Referência. Este Modelo, evidentemente, deverá gerar resultados 100% confiáveis. Em nosso caso, foi utilizada a implementação do algoritmo *NW* disponibilizada pelo *Rodinia Benchmark Suite* (RODINIA,2015), um site de *benchmarks* focados em aplicações de alto-desempenho para plataformas heterogêneas e *multi-cores*.

A figura 4.28 apresenta a estrutura em blocos usada em nosso *testbench*. O arquivo texto com as sequências, randomicamente, geradas foi submetido a ambas implementações. Os arquivos textos gerados a partir destas foram comparados ao do modelo de referência para ajuste e validação da arquitetura.

figura 4.28 - **Estrutura em blocos do *testbench*.**



Fonte: Elaborada pelo autor

Na fase pré-síntese do projeto foi utilizada a ferramenta *Questa Advanced Simulator* (QUESTA,2015), uma ambiente de desenvolvimento e simulação para

SoCs e *FPGAs* fornecido pela *Mentor Graphics* (MENTOR,2015). Posteriormente, já com a arquitetura instalada no *FPGA*, foi utilizada o analisador lógico *Signal Tap II* (SIGNALTAP,2015), fornecido pela Altera para o *debug* em tempo real de implementações em *hardware*.

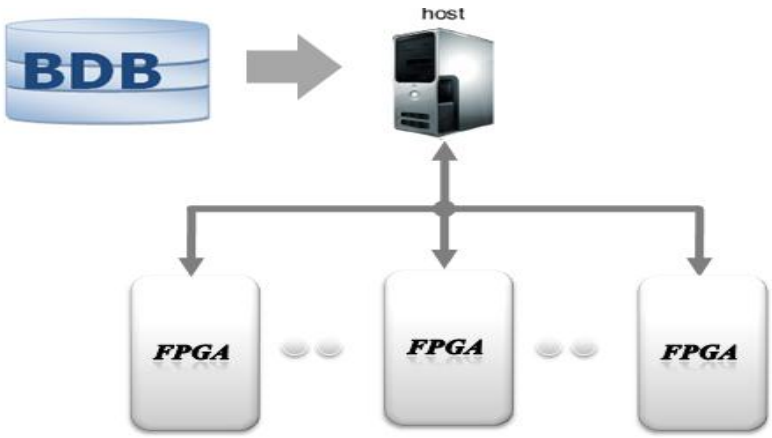
4.11 Escalabilidade

Este projeto apresenta boas possibilidades de escalabilidade possibiltando a expansão da arquitetura através do aumento do número de *FPGAs*.

Numa abordagem semelhante a um *Cluster*, os *FPGAs* atuariam de forma paralela e independente, conforme mostra a figura 4.29, cabendo ao *host* o envio dos *bitstreams* de configuração dos mesmos, a distribuição homogênea das cargas e, posteriormente, a composição final dos resultados. Caberia a cada um dos *FPGAs* a execução dos alinhamentos referentes a uma partição do banco de dados.

O ponto importante a ser avaliado é o gerenciamento do tráfego com o *host,* uma vez que, devido à restrição na largura de banda desta interface a mesma poderá se transformar em um "gargalo" nas comunicações, comprometendo o desempenho em geral.

figura 4.29 - **Expansão paralela da arquitetura.**



Fonte: Elaborada pelo autor

4.12 Conclusão

Neste capítulo, foi apresentada a arquitetura de nosso proposta de implementação e aceleração do algoritmo NW. Foram detalhados todos os módulos de *hardware* e o fluxo de operação das unidades de controle, bem como, as interfaces com a memória *on-board* e com a camada de *software*. Foram

explicitadas as restrições impostas pelas limitações de recursos disponibilizadas pelo *FPGA* e suas implicações no desempenho da arquitetura. Também, foi apresentada a metodologia de teste e verificação adotada na validação da arquitetura. E, finalizamos com uma superficial abordagem referente às possibilidades de escalabilidade desta arquitetura.

5 RESULTADOS EXPERIMENTAIS

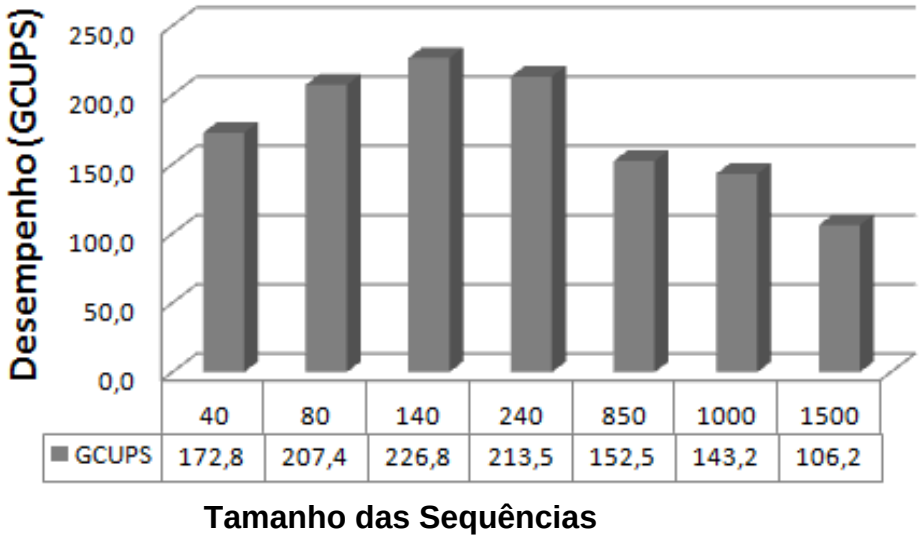
Este capítulo apresenta os resultados de desempenho da plataforma proposta. Serão apresentados os tempos reais de execução dos alinhamentos obtidos com a seguinte configuração: Intel Core i5 @ 2.4 GHZ no *host* operando no S.O. *Debian* (DEBIAN,2014), versão 7.0 *(kernel Linux)*; por sua vez, o *FPGA Stratix III* da *Altera* configurado para operar a 125 MHz.Em alguns casos, quando os resultados de desempenho foram, exclusivamente, referentes ao *FPGA*, serão estimativas em *GCUPS* considerando uma frequencia de operação de 200 MHz.

Inicialmente, é apresentada uma característica peculiar às implementações com *FPGAs* através de uma comparação com uma implementação em *"C"*, não-otimizada. Na sequência, são confrontados os resultados obtidos em nossa proposta com aqueles apresentados pelos artigos do capítulo 3 - Trabalhos Relacionados.

5.1 Desempenho (em GCUPS) x Tamanho das Sequências

A figura 5.1 apresenta o gráfico de desempenho, exclusivamente, do *FPGA* na execução do algoritmo *NW* para diferentes tamanhos de sequências.

figura 5.1 **- Desempenho x Tamanho das sequências**



Fonte: Elaborada pelo autor

A métrica escolhida foi o CUPS por ser a mais utilizada em algoritimos que utilizam matrizes de Programação Dinâmica, conforme já informado.

Para cada um dos casos apresentados no gráfico, buscou-se o número de *PEs/Array* que possibiltasse o máximo desempenho e que não extrapolasse o limite máximo de PEs. O gráfico demonstra que, de fato, o desempenho do *FPGA*, considerando a arquitetura implementada no mesmo, depende do tamanho das sequências a serem alinhadas. A explicação para o comportamento apresentado no desempenho pode ser resumido em duas situações:

a) Para sequências pequenas, o número máximo de Unidades de Alinhamento, e que não extrapolam a capacidade de armazenamento da memória interna, não pode ser implementado, uma vez que, este valor ultrapassa o número máximo de entradas do *multiport* (768 bits). Dessa forma, o paralelismo *coarse-grained* não é explorado ao máximo.

b) Na medida em que as sequências tornam-se maiores, a capacidade de armazenamento de memória interna é mais demandada e em função da restrição de recursos de memória interna do FPGA, o número de Unidades de Alinhamento deverá ser, progressivamente, reduzido e, consequentemente, o desempenho diminui.

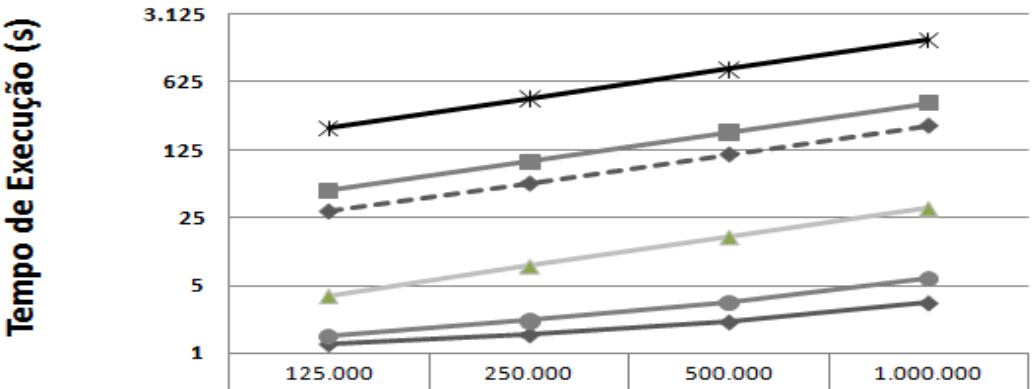
5.2 Tempo de Execução x Tamanho das Sequências

Na figura 5.2, temos o gráfico dos tempos totais de execução em função do número de Alinhamentos considerando diferentes tamanhos de sequências.

Diferentemente do gráfico anterior que se restringia, exclusivamente, ao *FPGA*; na análise atual, os valores apresentados foram obtidos levando-se em conta todos os tempos envolvidos no processo de alinhamentos, tais como:

- Transferência do arquivo de configuração do *FPGA* do *host* para a placa aceleradora.
- Transferência das bases de dados das sequências a serem alinhadas do *host* para a placa.
- Execução dos alinhamentos no *FPGA*.
- Gravação dos resultados nas memórias da placa.
- Transferência dos resultados das memórias da placa para a memória do *host*.

figura 5.2 **- Gráfico do tempo de execução x número de Alinhamentos para diferentes tamanhos de sequências**



	125.000	250.000	500.000	1.000.000
140 bps	1,3	1,6	2,1	3,3
240 bps	1,5	2,2	3,3	5,9
850 bps	3,9	7,9	15,8	31,1
1000 bps	48,0	95,0	190,0	380,0
1500 bps	212,0	424,0	850,0	1.697,0
1500 bps (*)	29,0	56,0	111,0	219,0

Fonte: Elaborada pelo autor (*) FPGA com uma memória interna dez vezes maior que a atual..

Como já esperado, o tempo de execução aumenta com o tamanho das sequências devido ao aumento no tamanho dos arquivos a serem transferidos e, principalmente, à queda de desempenho no *FPGA* provocada pela redução do número de unidades de alinhamento em função das restrições dos recursos de memória, conforme já explicado no ítem anterior. Para efeitos comparativos, a linha tracejada na figura 5.2 apresenta a curva do tempo de execução para sequências de 1.500 nucleotídeos, considerando que a memória interna do *FPGA* fosse dez vezes maior que a disponibilizada pelo *FPGA* atual. Esta curva demonstra o grau de relevância deste recurso do *FPGA* em relação ao desempenho nesta arquitetura.

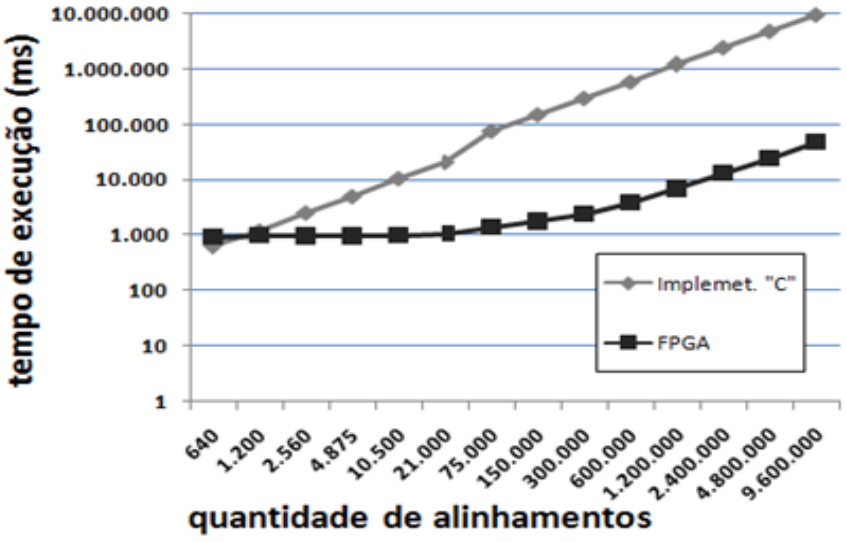
5.3 Comparação com uma implementação sequencial não-otimizada

O gráfico na figura 5.3a apresenta os tempos totais de execução do algoritmo *NW* na plataforma proposta e numa implementação sequencial não-otimizada em linguagem C. Por esta razão, este gráfico não ter por objetivo fazer comparações entre o desempenho destas duas implementações e, sim, apresentar uma característica que deve ser levada em conta nas implementações com um *FPGA*.

No gráfico são apresentados diferentes quantidades de alinhamento e, em todos os casos, foi utilizado o tamanho de 240 *bps* para as sequências. A escolha desse valor deveu-se ao bom desempenho apresentado pela arquitetura nesta faixa de tamanhos, conforme demonstrado na figura 4.25.

Pode ser observado que no início das curvas com pequenas quantidades de alinhamentos, o tempo de execução na plataforma proposta é maior que o apresentado pela implementação sequencial. Á medida em que o número de alinhamentos aumenta, as curvas se invertem e o tempo de execução do algoritmo em *hardware* torna-se, progressivamente, menor.

figura 5.3 **- Tempo de execução x Número de Alinhamentos**



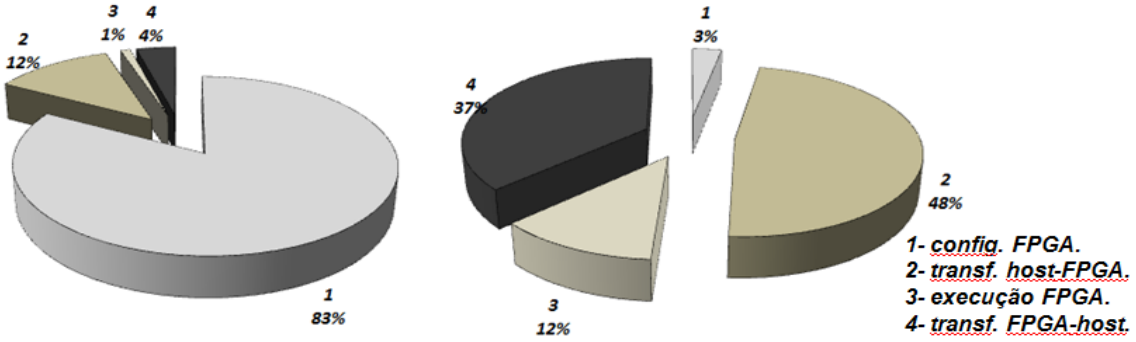
Fonte: Elaborada pelo autor

A explicação para este comportamento está relacionado à seguinte fato: Para uma quantidade pequena de alinhamentos, o tempo de configuração do *FPGA*, que é fixo e ocorre uma única vez, pode ser bastante significativo em relação ao tempo total do processo de alinhamento, comprometendo o desempenho geral. Ou seja, de uma forma geral, o uso do *FPGA* só é justificado a partir de uma determinada quantidade mínima de alinhamentos.

O gráfico na figura 5.4a, o qual apresenta a extratificação dos tempos num processo com uma pequena quantidade de Alinhamentos, demonstra bem este fato.

À medida em que a quantidade de Alinhamentos aumenta e, com isto, a volume de dados a serem processados, o *overhead* causado pela configuração do *FPGA* torna-se desprezível, conforme demonstra a figura 5.4b, reduzindo o comprometimento do tempo total de execução do algoritmo.

figura 5.4 **- Extratificação dos tempos no processo de Alinhamento**



a) 640 alinhamentos. *b) 2.400.000 alinhamentos.*

Fonte: Elaborada pelo autor

5.3 Comparação com os Trabalhos Relacionados

A seguir é apresentado um comparativo dos resultados obtidos com a atual arquitetura proposta nesta dissertação e aqueles apresentados em cada um dos trabalhos descritos no *capítulo 3 - Trabalhos Relacionados*.

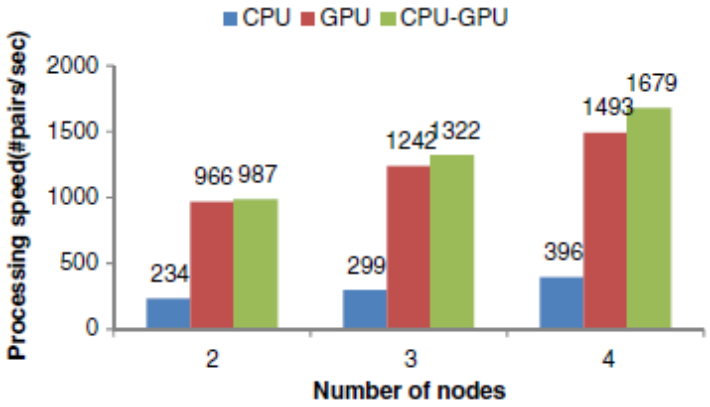
5.3.1 Large-Scale Pairwise Alignments on GPU Clusters: Exploring the Implementation Space

Em (TRUONG,2014) foi apresentado o projeto de um Cluster *CPU-GPU* em um ambiente distribuído *MPI-CUDA*. Também foram apresentadas quatro diferentes *kernels* de implementação do algoritmo *NW* com *gap* linear para as *GPUs*. Por sua vez, os testes foram realizados em duas diferentes estruturas para o *Cluster* (uma *High-end* e uma *Low-end Cluster*), conforme figura 3.3 na página 49.

Nos experimentos foi utilizado um conjunto de dados com cerca de 25.000 sequências únicas do gene bacteriano 16S rDNA, com cerca de 1.536 *bps***,** gerados a partir do *Ribosomal Database Project* (COLE,2009).

Considerando a escalabilidade do *Cluster*, a figura 5.5 apresenta os resultados referentes ao desempenho, em número de pares alinhados/segundo, no *High-end Cluster* considerando diferentes quantidades de *nós.*

figura 5.5 **- Desempenho apresentado pelos *High-end Cluster***



Fonte: TRUONG,2014

Assim, o pico máximo de desempenho obtido foi de 1.679 pares alinhados/segundo tendo sido atingido na configuração *high-end Cluster* com 4 *nós*.

Para efeitos comparativos elaboramos dois experimentos utilizando duas configurações distintas para a nossa arquitetura; em ambos os casos, foram executados 128.000 alinhamentos com uma frequencia de clock de 120MHz.

1) Na primeira implementação em função do tamanho das sequências *(1.536 bps)* e devido às restrições de memória interna já apontados no ítem 4.8, só foi possível a implementação de uma Unidade de Alinhamento com 128 PEs. Nesta configuração foram consumidos cerca de 75% de todos os blocos de memória disponíveis no FPGA. Como só foi implementada uma Unidade de Alinhamento, o *multiport* foi configurado com sua largura mínima (64 bits). O resultado foi um tempo total de execução de, aproximadamente, 56 segundos o que representa uma taxa de:

   # pares/segundo = 128.000 / 56 ≈ 2.304 pares alinhados/segundo

   Portanto, 37% superior à taxa máxima obtida pelo *Cluster* que foi de 1.679 pares/segundo em sua configuração máxima com 4 *nós* composta por quatro *workstations* e nove *GPUs,* conforme Tabela 3.2 na página 50*.* Nas conclusões deste capítulo serão abordados os motivos que levaram a esta significativa diferença de desempenho.

2) Na segunda implementação, o *traceback* foi executado no *host.* Dessa forma, conforme explicado no ítem 4.9, a memória interna não é utilizada, possibilitando uma maior liberdade na quantidade de Unidades de Alinhamento. Assim, configuramos a arquitetura para 128 Unidades de Alinhamento com 2 PEs/Array

e obtivemos um tempo total de execução de, aproximadamente, 240,7 segundos. Com isto, a taxa de geração de pares/segundo ficou igual a:

pares/segundo = 128.000 / 240,7 ≈ 532 pares alinhados/segundo.

Portanto, muito abaixo à taxa obtida em nossa primeira implementação. Essa queda de desempenho é causada pelo elevado tempo de transmissão dos arquivos dos *dir*_bits para o *host* representando 91% do tempo total de execução.

5.3.2 G-DNA - a highly efficient multi-GPU/MPI tool for aligning nucleotide reads

Em (FROHMBERG,2013) foi apresentada uma solução em *software* altamente paralelizada denominada *G-DNA (GPU-based DNA aligner)* para o processo de montagem das sequências do *DNA* a partir das *short-reads* geradas pelos sequenciadores da geração *NGS*. A solução foi desenvolvida para uso tanto em plataformas multi-GPU como em MPI+GPU Clusters.

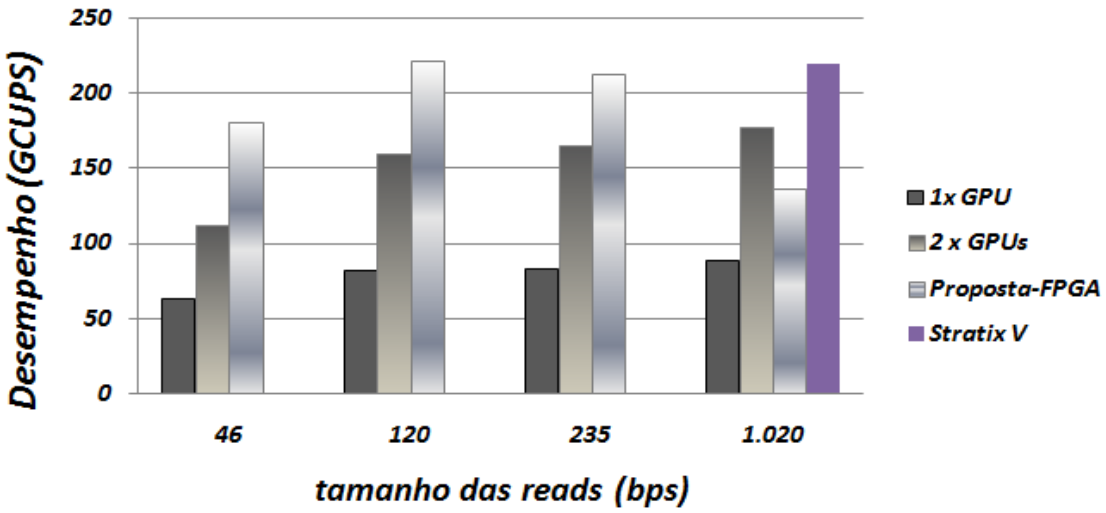
O algoritmo *NW* foi empregado no processo de comparação das regiões de *overlaps* entre as *short-reads*, o qual faz parte da metodologia *overlap-layout consensus* de montagem do DNA.

Para avaliar o desempenho da implementação proposta, foram realizados testes com *reads* de diferentes tamanhos *(46, 120, 235 e 1020 bps)* geradas, respectivamente, a partir dos sequenciadores 454 da *Roche (*ROCHE_454,2015), o *Solexa* da *Ilumina* (ILLUMINA,2015), o *Solid* da *Applied Biosystems* (APPLIED,2015) e o Roche 454 GS FLX Titanium (ROCHE_FLX,2015).

Para a avaliação de desempenho foi utilizada a seguinte estrutura de *hardware*: Uma CPU Intel Quad-Core Q8200 @ 2.33GHz e duas GPUs NVIDIA GeForce GTX 580.

A figura 5.6 apresenta os resultados obtidos em *GCUPS* e, em cada caso, a terceira coluna representa o desempenho gerado em nossa proposta de arquitetura. Para o tamanho de 1020 *bps* foi acrescentada uma quarta coluna considerando a hipótese de substituição do nosso *FPGA* por um *Stratix* V, o qual disponibiliza 5 vezes mais memória *on-chip.*

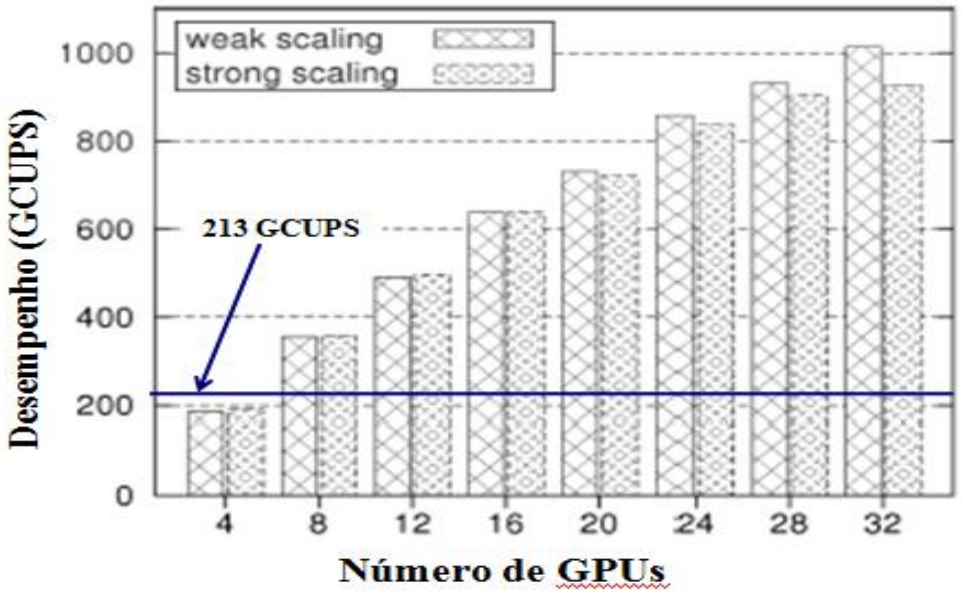
figura 5.6 **- Comparativo de Desempenhos.**



Fonte: FROHMBERG,2013

O segundo gráfico de desempenho apresentado no artigo (figura 3.7) demonstra a boa escalabilidade do *software* proposto para o *Cluster*. Foram utilizadas *reads* com um tamanho médio de 235 *bps* geradas a partir do sequenciador *Roche 454.* O desempenho máximo de 1014 *GCUPS* é obtido quando o *Cluster* comporta 32 *GPUs*. Este gráfico está reapresentado na figura 5.7 com a inserção de uma linha cheia que indica o desempenho de 213 GCUPS obtido em nossa proposta de arquitetura, ultrapassando, portanto, o atingido pelo *Cluster* quando configurado com até quatro *GPUs*.

figura 5.7 **- Quadro comparativo de Desempenho.**



Fonte: FROHMBERG,2013

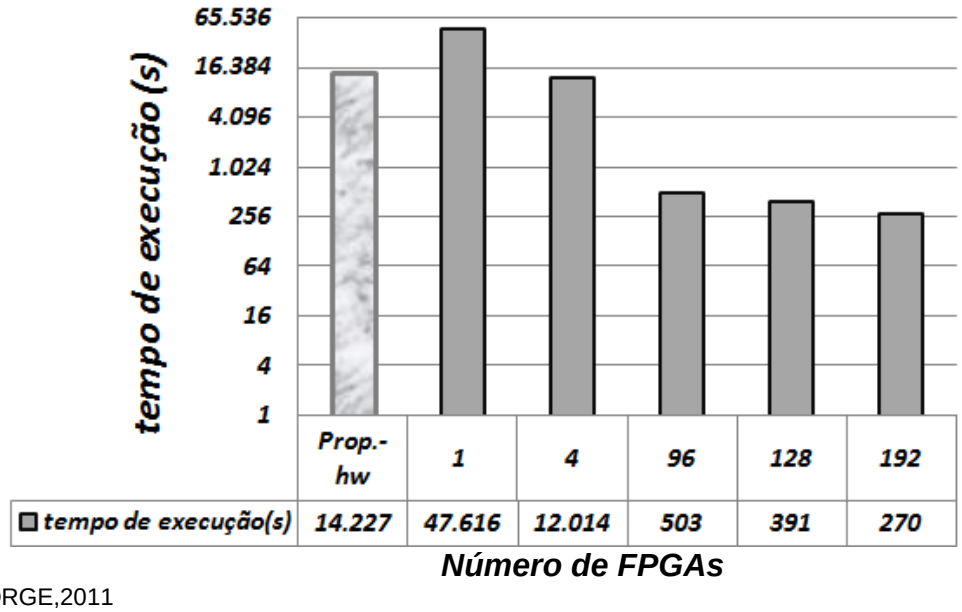
5.3.3 Novo-G : At the Forefront of Scalable Reconfigurable Supercomputing

EM (GEORGE,2011) foi apresentado o Novo-G, um poderoso computador reconfigurável baseado num *Cluster* de FPGAs com uma estrutura escalável em até 192 *FPGAs*, em operação desde julho/2009. Na implementação do algoritmo *NW*, a abordagem usada foi através de um Array Sistólico único com 850 *PEs* em cada *FPGA*, conforme figura 3.9. Além disto, foi empregada uma técnica denominada *in-stream control* que, conforme o nome já diz, introduz caracteres de controle no próprio *streaming* de dados. Com isto, a complexidade dos blocos de controle e sua àrea ocupada no chip são reduzidos, liberando recursos para a implementação de mais blocos de processamento.

Na figura 3.10 foi apresentado um quadro comparativo de desempenho na implementação do algoritmo *NW* entre um único *FPGA* do *Novo-G* e uma implementação otimizada em *C* para diferentes tamanhos de sequências e quantidades diversas de alinhamentos.

Uma tabela foi apresentada com os resultados referentes ao tempo de execução e *speedup* considerando sequências com um tamanho de 850 *bps*, num total de 32 milhões de alinhamentos e um crescimento escalar na quantidade de *FPGAs.* A figura 5.8 apresenta um gráfico comparativo entre estes resultados e o tempo de execução obtido com nossa proposta sob as mesmas condições de testes.

figura 5.8 - **Quadro comparativo do tempo de execução.**



Fonte: GEORGE,2011

Analisando os resultados, verifica-se que o tempo de execução apresentado por nossa proposta foi cerca de 70% menor que o apresentado pelo *Novo-G* quando este utiliza, apenas, um *FPGA,* o que representa um *speedup* de 3.35x*.* E, apenas, 18% maior que o apresentado pelo *Novo-G* quando este foi configurado com 4 *FPGAs*. A explicação para este bom desempenho apresentado por nossa proposta está, diretamente, relacionada ao fato de termos explorado o paralelismo *coarse-grained* através do uso de dez unidades de alinhamento que atuaram em paralelo. Ao passo que na implementação apresentado pelo artigo, apesar dos uso de 850 *PEs*, foi utilizado, apenas, um único array por *FPGA*.

É importante frisar que o *FPGA* utilizado neste artigo para o *Novo-G* é o mesmo utilizado em nosso projeto e, em ambos os casos, a frequencia de *Clock* foi igual a 125 MHz.

5.3.4 Bioinformatics Applications on the FPGA-Based High-Performance Computer RIVYERA

Em (WIENBRANDT,2013) foi apresentado o RIVYERA, uma plataforma computacional de alto-desempenho baseada num *Cluster* reconfigurável de *FPGAs*. O artigo apresentou duas versões da plataforma: a RIVYERA S3-5000 e a RIVYERA S6-LX150. A primeira foi configurada para a aceleração dos algoritmos Smith-Waterman e BLAST. Enquanto que, a segunda foi dedicada ao algoritmo BWA.

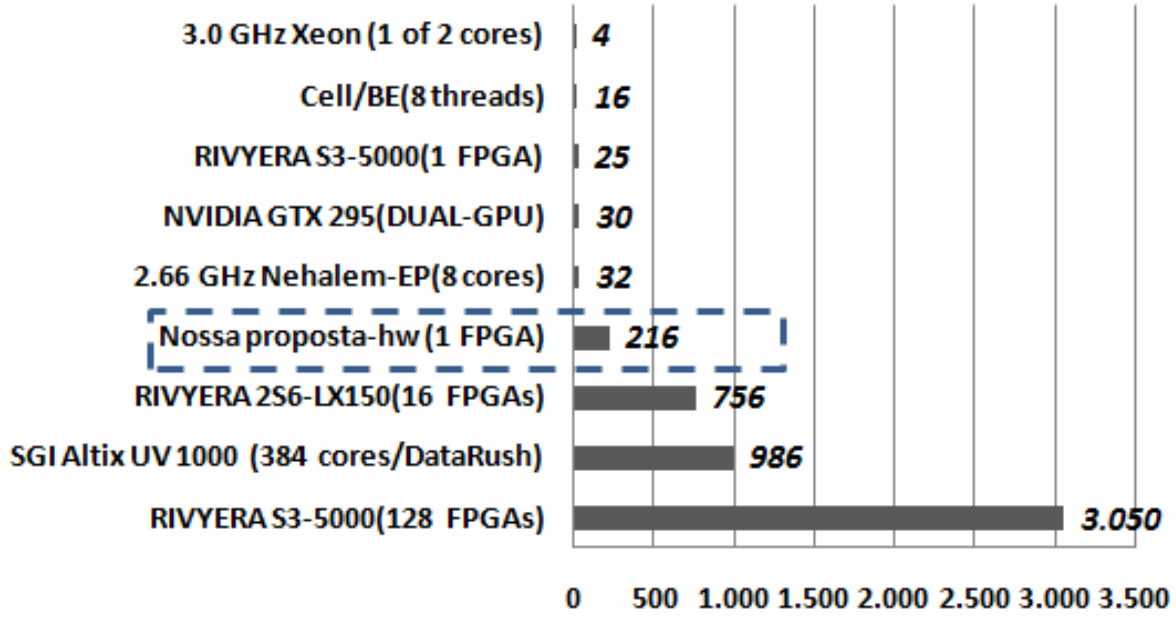
Os resultados apresentados para o RIVYERA S3-5000, e que se referem à fase 1 do algoritmo SW, podem ser comparados com os obtidos em nossa proposta de arquitetura devido à semelhança deste algoritmo com o NW, em sua fase inicial.

Na avaliação de desempenho, o algoritmo *SW* foi utilizado no alinhamento de *short-reads* com um tamanho médio de 100 *bps*. A abordagem empregada na solução foi através do uso de *arrays sistólicos* cujas células foram denominadas *SWcell*. Os *FPGAs* foram configurados com 4 *arrays* contendo, cada um deles, cem *SWcells*, possibilitando o alinhamento simultâneo de até 4 sequencais. Assim, nesta configuração com 400 *SWcells*, cada *FPGA* atingiu o desempenho de 25 *GCUPS*.

Em nossa plataforma, sob as mesmas condições de testes, fazendo uso, também, de 400 *PEs*, foi obtida uma taxa de 66,67 *GCUPS*. A diferença em nossa estratégia, e que maximiza o desempenho, foi o uso de 100 Unidades de Alinhamentos e, apenas, 4 *PEs/Array*. Na verdade, em nossa proposta de solução

para este tamanho de sequência, pode-se chegar a 324 unidades de alinhamento o que implica num desempenho de 216 *GCUPS*.

figura 5.9 - **Quadro comparativo de desempenho (em GCUPS)**



Fonte: Elaborada pelo autor

O desempenho de nossa proposta de *hardware* foi inserido no quadro da figura 5.9, o qual foi baseado no apresentado em (SCIENGINES, 2012) que faz um comparativo de desempenho na implementação do algoritmo SW em diferentes plataformas.

5.4 Conclusão

Este capítulo dedicou-se à comparação dos resultados, essencialmente, referentes a desempenho obtidos com nossa arquitetura proposta para a aceleração do algoritmo NW.

Inicialmente, foram apresentados alguns gráficos que demonstraram estes resultados em relação ao tamanho das sequências a serem alinhadas. Em seguida, foram apresentados gráficos que demonstraram a inadequação do uso do *FPGA* numa situação com um número reduzido de alinhamentos.

Por fim, os resultados apresentados nos trabalhos relacionados foram, individualmente, confrontados com os nossos. E, dessa forma, podemos concluir que, apesar de nossa arquitetura utilizar, apenas, um *FPGA* e das restrições impostas pelo poucos recursos da memória *on-chip*, dependendo do tamanho das

sequências, o seu desempenho foi bastante competitivo e, alguns casos, ultrapassando aqueles apresentados nos diversos *Clusters* de *GPUs* e *FPGAs*, até uma determinada quantidade de *nós*.

Destacando-se o resultado obtido na comparação com o primeiro artigo (ítem 5.3.1), no qual uma das configurações de nossa arquitetura com o *FPGA* operando a 120 MHz gerou uma taxa de alinhamento 37% superior à obtida com o *Cluster CPU-GPU* composto por quatro *workstations* operando a 2.0 GHz e mais nove *GPUs.* Esta diferença de desempenho pode ser atribuída aos seguintes motivos:

a) Em nossa implementação com uma única Unidade de Alinhamento com 128 PEs, cada um destes ficou responsável pela geração de forma continuada de uma coluna inteira da matriz de alinhamento em cada *slice* com um *overhead* de 128 ciclos de *Clock* entre um *slice* e o próximo. Por sua vez, a implementação no *Cluster CPU-GPU*, também, recebeu uma larga distribuição entre os diversos *cores* de processamento; no entanto, o tamanho dos *Tiles* estava limitado ao tamanho da memória local (número de registradores e à *shared memory*) exigindo constantes comunicações com a memória global e, por sua vez, invocações do *kernel* por questões de sincronização entre blocos. Estes dois tipos de operações introduzem significativos *overheads* devido á relativa baixa largura de banda da memória global e, principalmente, da interface com o *host*.

b) Os algoritmos de Alinhamento de sequências biológicas trabalham com dois tipos de dados: Nucleotídeos e Aminoácidos. No primeiro caso são 4 tipos que podem ser codificados com, apenas, 2 bits; enquanto que, o segundo caso, são 20 tipos codificáveis com 5 bits. A baixa granularidade dos recursos disponibilizados em um FPGA permitem um melhor aproveitamento da largura desse tipo de dado possibilitando a implementação de elementos de processamento que operem, exatamente, na largura mínima necessária de bits. Esta compactação permite uma eficiente utilização dos recursos de memória; bem como, um máximo aproveitamento de sua largura de banda e da interface com o *host.* O que não ocorre nas *GPUs* com sua arquitetura fixa, na qual os *cores* operam com larguras padronizadas (normalmente, 32 ou 64 bits) em seus dados de entrada, não permitindo que suas memórias se beneficiem desta compactação possibiltada pelos tipos de dados biológicos.

6 ESTUDO DE CASO

Este capítulo tem como objetivo apresentar os resultados referentes ao desempenho obtido com a plataforma desenvolvida neste projeto em uma determinada situação num contexto real. Assim, através da demonstração de sua aplicabilidade para uma determinada àrea do conhecimento e, por extensão, para outras, buscamos a validação deste trabalho.

Neste estudo de caso, optamos pelo uso das sequências obtidas a partir do genoma de uma família de vírus que atacam vários tipos de plantas. Este estudo é relevante em função da importância econômica destas culturas.

6.1 A doença do algodeiro

O algodão está entre as mais importantes culturas de fibras no mundo e uma das mais antigas, haja vista o fato do algodeiro já ser conhecido 8.000 anos A.C. e tecidos de algodão já existirem na Índia 3.000 A.C. Todos os anos, uma média de 35 milhões de hectares de algodão é plantada por todo o planeta com uma produção anual estimada em 27 milhões de toneladas. A demanda mundial tem aumentado gradativamente desde a década de 1950, a um crescimento anual médio de 2%. O comércio mundial do algodão movimenta anualmente cerca de US$ 12 bilhões e envolve mais de 350 milhões de pessoas em sua produção, desde as fazendas até a logística, o descaroçamento, o processamento e a embalagem. Atualmente, o algodão é produzido por mais de 60 países, nos cinco continentes. China, Índia, Estados Unidos e o Paquistão são os maiores produtores responsáveis por mais de 70% da produção mundial da fibra (ABRAPA,2015).

Existe, no entanto, uma doença cuja sigla é *CLCuD (Cotton leaf curl disease)* e ocorre na folha do algodeiro provocando a sua deformação, inchaços e escurecimentos em suas veias, provocando o nanismo total da planta e a, consequente, redução na produtividade e na qualidade do algodão produzido.

Na figura 6.1, do lado esquerdo, temos um algodeiro atacado pelo vírus com sua folhagem totalmente murcha, no canto à direita uma folha saudável.

figura 6.1 - **A doença *CLCuD* na folha do algodeiro.**

fonte : National Institute for Biotechnology ang Genetic Engineering(NIBGE),Pakistan.

Esta doença é causada por alguns dos vírus da família dos *geminivírus* (gênero *Begomovirus*) cujo *vetor*, que é o agente responsável pela disseminação, é a mosca branca *(Bemisia tabaci)*. Nos anos 90, esta doença devastou boa parte das plantações do Paquistão provocando prejuízos da ordem de U$ 5 bilhões (BROWN,2013). Além do Paquistão, na Índia, em 1997, ocorreu uma epidemia que afetou uma área de 0,1 milhões de hectares no estado de Rajasthan (KHAN,2014). Ainda há registros do *CLCuD* no Egito, na Nigéria, na Tanzânia e no Sudão. Além do algodão, outras culturas, também, são afetadas por esta doença, tais como, o tomate, o feijão e a mandioca.

Um dos vírus causadores da *CLCuD* é o *Cotton leaf curl Burewala*, o qual encontra-se catalogado no banco de genomas do *NCBI* com o número de acesso NC_012137 (NCBI,2015). Este vírus é constituído por uma única fita de *DNA* com 2.759 nucleotídeos. Na página do *NCBI* existem diversos outros vírus que provocam a *CLCuD*.

Este estudo de caso está baseado em uma situação hipotética na qual, em uma determinada região de plantações suscetível à doenças virais, começassem a ocorrer danos atípicos à folhagem das plantas. Uma primeira ação de busca para o diagnóstico do problema seria a identificação do agente causador da doença, o qual

poderia ser algum microorganismo, tais como bactérias, fungos ou vírus, entre outros. Caso as suspeitas recaíssem sobre algum agente de natureza viral, ainda, desconhecido, duas ações distintas e não-excludentes poderiam ser tomadas objetivando a sua identificação:

1) Testes biológicos de suas amostras, em laboratório.
2) Sequenciamento do seu DNA seguido de uma pesquisa em bancos de dados.

Baseados nesta segunda ação, o objetivo deste estudo de caso foi simular esta situação utilizando a nossa plataforma para acelerar a pesquisa deste vírus em um banco biológico. Ou seja, a idéia foi executar um experimento com o genoma do *Cotton leaf curl Burewala virus*, como se o mesmo fosse o tal vírus desconhecido, baixando a sua sequência do banco do *NCBI* e submetê-lo ao alinhamento com todas as sequências de um banco gerado em nosso computador.

No entanto, como o tamanho deste vírus *(2.759 bps)* ultrapassa o tamanho máximo de 1.881 *bps* suportável por nossa arquitetura, confome demonstrado no ítem 4.8, não foi possível utilizá-lo. A solução, então, foi executar o mesmo procedimento utilizando o seu vírus-satelite, o *Cotton leaf curl Burewala virus betasatellite* cujo o número de acesso NC_013802 no banco do *NCBI* e que possui o tamanho igual a 1.354 *bps.* Entendemos que esta mudança de planos não invalida o nosso experimento, pois, a identificação positiva deste vírus-satélite poderá apontar a presença do seu *helper vírus*, o *Cotton leaf curl Burewala*.

Na verdade, o nosso objetivo foi testar o desempenho de nossa plataforma fazendo uso de uma situção hipotética; porém, utilizando uma sequência real como nossa sequência de busca *(Query)*.

Observando os tamanhos das sequências dos *vírus satelites* cadastrados no *NCBI* verifica-se que eles são próximos dos 1.500 nucleotídeos. Como no ítem 5.3.1 apresentamos a comparação de desempenho de nossa arquitetura com a obtida pelo *Cluster CPU-GPU* (TRUONG,2014) e as sequências tinham um tamanho fixo de 1.536 nucleotídeos, poderíamos, portanto, utilizar a mesma configuração para a nossa arquitetura utilizada naquela comparação.

Assim, a nossa arquitetura foi configurada com uma única Unidade de Alinhamento com 128 PEs e com uma frequência de *Clock* de 120 MHz. Em seguida, executamos 128.000 alinhamentos com a sequência do *Cotton leaf curl Burewala virus betasatellite*. O resultado foi um tempo total de execução de 56 segundos. Para efeito de comparação, se este mesmo número de alinhamentos for

executado no *Cluster CPU-GPU* do ítem 5.3.1 em sua configuração de 4 *nós* e com sua taxa máxima de 1.679 matrizes/segundo, o tempo total de execução seria de 76,24 segundos. Esta diferença nos desempenhos já foram abordadas no ítem 5.4.

No Apêndice A é apresentada uma listagem final de um dos alinhamentos com a respectiva taxa de similaridade; no caso, é apresentado o resultado dos alinhamentos entre a sequência do *Cotton leaf curl Burewala virus betasatellite* com o *Cotton leaf curl Bangalore virus betasatellite*. A taxa de similaridade apresentada foi obtida, simplesmente, pela relação entre o número total de coincidências (*matches)*, representadas pelo símbolo "|", e o tamanho de nossa *Query*.

6.2 Conclusão

O presente capítulo apresentou um estudo de caso com o objetivo de demonstrar de que forma a arquitetura apresentada neste projeto poderia contribuir na solução de um problema real. Este estudo apresentou a doença *CLCuD (Cotton leaf curl disease)* provocada por um vírus, o qual ataca não, apenas, as plantações de algodão como, também, outras culturas comercialmente importantes.

7 CONCLUSÕES e TRABALHOS FUTUROS

Nesta dissertação, propusemos e avaliamos uma abordagem de execução do algoritmo de Alinhamento Global Needleman-Wunsch, neste caso, aplicado a sequências de nucleotídeos em cadeias de DNA, através de uma plataforma reconfigurável de alto-desempenho. A plataforma híbrida usada foi constituída por um PC *(host)* e uma placa aceleradora baseada em um FPGA. O principal objetivo foi demonstrar as potencialidades do uso de um dispositivo reconfigurável e, particularmente, da plataforma proposta no campo da Biologia Computacional.

Pôde ser observado que os processos de transferência de dados entre as memórias do *host* e da placa, bem como, a largura-de-banda entre a memória da placa e o FPGA são fatores restritivos ao desempenho e, em nosso trabalho, influenciaram na escolha de uma arquitetura interna baseada em *Arrays Sistólicos*. Além disto, demonstramos que numa arquitetura, totalmente, paralelizada, a quantidade de recursos internos disponibilizados pelo dispositivo reconfigurável, também, é um fator determinante quando o foco principal é o desempenho. Particularmente, em nosso caso, os recursos de memória interna disponibilizados pelo *FPGA Stratix III* estabeleceram um limite superior ao tamanho das sequências, na faixa do 1.900 nucleotídeos, e restringiram, bastante, o número de unidades de alinhamento na medida em que o tamanho das sequências se aproximou deste valor.

Ainda assim, conforme demonstrado, as taxas de desempenho obtidas com nossa plataforma foram bastante competitivas quando comparadas a de outros trabalhos do estado-da-arte, inclusive, aquelas implementações que fizeram uso, simultâneo, de diversas GPUs e FPGAs. Além disto, este desempenho, também, se refletiu num significativo *speedup* no tempo de execução dos alinhamentos em relação a uma implementação, puramente, sequencial.

As elevadas taxas de desempenho internas ao FPGA foram alcançadas, em virtude da exploração do paralelismo em dois níveis de granularidade: o *fine-grained*

entre os PEs pertencentes ao mesmo *Array Sistólico* e o *coarse-grained* entre as unidades de alinhamento.

7.1 Trabalhos Futuros

Ao término deste projeto, é possível apontar algumas ações de melhorias, como, por exemplo:

- A migração da placa atual para uma outra cujo FPGA instalado disponibilize mais recursos, principalmente, de blocos de memória para a síntese de RAMs.
- A partir de sua boa escalabilidade, expandir a arquitetura atual através da incorporação de outros FPGAs. Esta ação, dependendo da tecnologia de interface com o *host* e do seu gerenciamento, poderá aumentar, proporcionalmente, a capacidade de processamento e o desempenho da arquitetura.
- Ampliar a atual arquitetura para a execução do Alinhamento Global de Sequências de Proteínas, inclusive, utilizando o *affine gap*.
- Reconfigurar a arquitetura atual para que, também, seja possível a implementação do Alinhamento Local Smith-Waterman.
- Reestruturar a arquitetura atual para que a mesma não mais dependa da memória interna do FPGA e, sim, utilize a memória *off-chip* (da placa) para o armazenamento temporário dos *dir_bits.* Dessa forma, poderemos trabalhar com sequências muito maiores.

Estas três últimas ações irão ampliar, significativamente, a gama de aplicações para a arquitetura proposta não, apenas, na Biologia Computacional como, também, em outras áreas de pesquisa.

Particularmente, o êxito na última ação possibilitaria a operação com sequências de *DNA* (ou de proteínas) com dezenas ou centenas de *Kbps* permitindo, por exemplo, o alinhamento de trechos genômicos humanos, acelerando processos relacionados à pesquisas de doenças, como o câncer, ou, ainda, no levantamento de perfis genéticos individuais relacionados à medicina genômica citada na introdução desta dissertação.

REFERÊNCIAS

ABRAPA. Associação Brasileira dos Produtores de Algodão. *O Algodão no Mundo*. Disponível em: <http://www.abrapa.com.br/estatisticas/Paginas/Algodao-no-Mundo .aspx**>**. Acesso em: 5 Jul. 2015.

ALTERA. Altera Corporation. Disponível em: <http://www.altera.com/>. Acesso em: 10 Maio. 2015.

ALTERA[1]. Altera Corporation. *Accelerating High-Performance Computing With FPGAs*. Out. 2007. Disponível em: <https://www.altera.com/content/dam/altera-www/global/ en_US/pdfs/literature/wp/wp-01029.pdf>. Acesso em: 11 Fev. 2015.

ALTERA[2]. Altera Corporation. *Implementation of the Smith-Waterman Algorithm on a Reconfigurable Supercomputing Platform**.*** Set. 2007. Disponível em: < https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ wp/wp-01035.pdf >. Acesso em: 14 Maio. 2015.

ALTERA[3]. Altera Corporation. *TriMatrix Embedded Memory Blocks in Stratix III Device Handbook***.** Volume 1, Chapter 4, Maio. 2009. Disponível em: <https://www.altera.com/en_US/pdfs/literature/hb/stx3/stx3_siii51004.pdf>. Acesso em: 17 Abr. 2015.

ALTERA[4]. Altera Corporation. *FPGA Architecture.* Jul. 2006. Disponível em: <https://www.altera.com/en_US/pdfs/literature/wp/wp-01003.pdf>. Acesso em: 22 Abr. 2015.

ALTERA[5]. Altera Corporation. *Cyclone V Device Overview.* Jun. 2015. Disponível em: < https://www.altera.com/en_US/pdfs/literature/hb/cyclone-v/ cv_51001.pdf >. Acesso em: 22 Jul. 2015.

ALTSCHUL, S.F.; RERICKSON, B.W. *Optimal Sequence Alignment Using Affine Gap Costs*. In: Bulletin of Mathematical Biology 48.5-6, pp. 603-616.1986.

ALTSCHUL, S. F. et al. *Gapped BLAST and PSI-BLAST: a new generation of protein database search programs*. Nucleic Acids Research, 25:3389–4202. 1997.

ALWAN, N.A.S.; IBRAHEEM, I.K.; SHUKR, S.M. *Fast Computation of the Shortest Path Problem through Simultaneous Forward and Backward Systolic Dynamic Programming.* International Journal of Computer Applications – Volume 54 – No. 1. Set. 2012.

APPLIED. Applied Biosystems Corporation. Disponível em: <www.appliedbiosystems.com>. Acesso em: 10 Jul. 2015.

BENKRID, K. et al. *High Performance Biological Pairwise Sequence Alignment: FPGA versus GPU versus Cell BE versus GPP**.*** Hindawi Publishing Corporation International Journal of Reconfigurable Computing Volume 2012, Article ID 752910. Fev. 2012.

BOUKERCHE, A. et al. *Reconfigurable Architecture for Biological Sequence Comparison in Reduced Memory Space**.*** Proceedings of the International Parallel and Distributed Processing Symposium – Nature Inspired Distributed Computing Workshop, p.1-8. 2007.

BRIDDON, R.W. et al. *Diversity of DNA β, a satellite molecule associated with some monopartite begomoviruses**.*** ScienceDirect, Virology, Volume 312, Issue 1, Pages 106–121, Jul. 2003.

BROWN, J. K. *Cotton Leaf Curl Viral Disease Complex**.*** School of Plant Sciences, The University of Arizona, Tucson, Arizona 85721 USA, 2013.

COLE, J. R.; WANG, Q.; CARDENAS, E*. The ribosomal database project: improved alignments and new tools for rRNA analysis.* Nucleic Acids Research, 37(Database issue), D141–D145. 2009.

CUDA. Disponível em: <http://www.nvidia.com.br/object/cuda_home_new_br.html>. Acesso em: 11 Jun. 2015.

CLCBIO. *High-Speed Smith–Waterman*. Disponível em: <http://www.clcbio.com/index.php?id=1254>. Acesso em: 12 Mar. 2012.

CLUSTALW. Disponível em: <http://www.ebi.ac.uk/Tools/msa>. Acesso em: 22 Fev. 2015.

DEBIAN. Sistema Operacional Debian. Disponível em: <http://www.debian.org>. Acesso em: 11 Maio. 2014.

DURBIN,R. et al.*Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press. 1998.

EMBL. European Molecular Biology Laboratory. Disponível em*:* <http://www.ebi.ac.uk>. Acesso em: 10 Maio. 2015.

FERLIN, E. P. *Arquitetura Paralela Reconfigurável baseada em fluxo de dados implemenetada em FPGA.* Universidade Tecnológica Federal do Paraná. Tese de Doutorado. Curitiba. Nov. 2008.

FERMI, *NVDIA's Next Generation CUDA Compute Architecture : Fermi.* NVIDIA's White Paper. 2009.

FISCHETTI, V. et al. *Identifying Periodic Occurrences of a Template with Applications to Protein Structure Combinatorial Pattern Matching.* Lecture Notes in Computer Science, v. 644, p. 111-120. 1992.

FROHMBERG, W. et al. *G-DNA - a highly efficient multi-GPU/MPI tool for aligning nucleotide reads.* BULLETIN OF THE POLISH ACADEMY OF SCIENCES TECHNICAL SCIENCES, Vol. 61, No. 4. 2013.

GEORGE, A.; LAM, H; STITT G. *Novo-G: At the Forefront of Scalable Reconfigurable Supercomputing.* IEEE magazine Computing in Science and Engeneering. Fev. 2011.

GIDEL. Gidel Company. Disponível em: <http://www.gidel.com>. Acesso em: 10 Maio. 2014.

GIDEL[2]. *ProcWizard User Manual v8_9*. Disponível em: <http://www.gidel.com/procwizard.htm>. Acesso em: 13 Ago. 2014.

GOTOH, O. *An Improved Algorithm for Matching Biological Sequences.* In:Journal of Molecular Biology 162.3, pp. 705-708. Dez. 1982.

GHR. *Genetics Home Reference/Handbook.* Disponível em: *<http://ghr.nlm.nih.gov/handbook/hgp?show=all>.* Acesso em: 13 Fev. 2015.

HOSSAIN, M. A.; KABIR, U.;TOKHI, M. O. *Impact of data dependencies in real-time high performance computing.* Microprocessors and Microsystems 26 p. 253–261. Abr. 2002.

INTEL. Intel Corporation. Disponível em: <http://www.intel.com>**.** Acesso em: 10 Abr. 2015.

ILLUMINA. Illumina Inc. Disponível em: <http://www.illumina.com/systems/miseq.html>. Acesso em: 8 Ago. 2015.

JANDA, J. M.; ABBOTT, S.L. *16S rRNA Gene Sequencing for Bacterial Identification in the Diagnostic Laboratory: Pluses, Perils, and Pitfalls.* Journal of Clinical Microbiology, 45(9): 2761–2764. Set. 2007.

KHAN, J. A.; SHWETA, M. K. *In silico* prediction of cotton (*Gossypium hirsutum*) encoded microRNAs targets in the genome of *Cotton leaf curl Allahabad virus.* Bioinformation Journal. 2014; 10(5): 251–255. Maio. 2014.

KINDATRENKO, V.V. et al. *GPU Clusters for High-Performance ComputingIn.* Proc. Workshop on Parallel Programming on Accelerator Clusters - PPAC'09. 2009.

KHRONOS. Khronos group. Disponível em: <https://www.khronos.org/>. Acesso em: 5 Abril. 2015.

KUNG, H. T.; LEISERSON, C.E*. Systolic Arrays for VLSI*. Interim report, Department of Computer Science,Carnegie Mellon University. Dez. 1978.

KUNG, H. T.; LEISERSON, C.E. *Systolic Arrays for VLSI Sparse Matrix*. Proceedings of the Society for Industrial and Applied Mathematics, p. 256 – 282, Santa Monica, CA, USA. 1979.

KUNG, H. T. *Why Systolic Architetures?.* Computer, vol.15, p.37 – 46. 1982

KUNG, S.Y. et al. *Wavefront Array Processors-Concept to Implementation.* Computer, vol.20, p.18 – 33. 1987.

LI, D. et al. *A Distributed CPU-GPU Framework for Pairwise Alignments on Large-Scale Sequence Datasets.* IEEE International Conference on Application-specific Systems, Architectures and Processors. 2013.

LIN,H. et al. *Massively Parallel Genomic Sequence Search on the Blue Gene/P Architecture.* Proceedings of the 2008 ACM/IEEE conference on Supercomputing (Piscataway, NJ, USA), SC '08, IEEE Press. 2008.

LUSCOMBE, N. M.; GREENBAUM D.; GERSTEIN M. *What is bioinformatics? An introduction and overview.* Yearbook of Medical Informatics 2, pp. 83. 2001.

LIMA, M. F. *Virus que infectam plantas.* Disponível em: <http://ainfo.cnptia. embrapa.br/digital/bitstream/item/91513/1/VIRUS-QUE-INFECTAM-PLANTAS.pdf>. EMPRAPA. Acesso em: 11 Jul. 2015.

MANAVSKI, S.; VALLE, G*. CUDA Compatible GPU cards as E_cient Hardware Accelerators for Smith-Waterman Sequence Alignment.* BMC Bioinformatics. 2008.

MAXFIELD, C. *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows.* Newnes. 2004.

MAZURKIEWICZ, J.; ZAMOJSKI, W. *Systolic-Based Hardware Realisation of Hopfield Neural Network.* Institute of Engineering Cybernetics, Wroclaw University of Technology, Revista do DETUA, vol.3, nº 5. Jan. 2002.

MEYERS,R.A*. Molecular Biology and Biotechnology: A comprehensive desk reference.* New York Vch Publishers. 1995.

MOORE, G. E. *The Microprocessor: Engine of the Technology Revolution*. In: Communications of the AMC. 1997.

MENTOR, Mentor Graphics Company. Disponível em: <http://www.mentor.com/>. Acesso em: 11 Jul. 2015.

NASH, J. G. *High-Throughput Programmable Systolic Array FFT architecture and FPGA Implementations*. Computing, Networking and Communications (ICNC), 2014 International Conference, Honolulu, HI, Publisher: IEEE. Fev. 2014.

NCBI. National Center for Biotechnology Information. Disponível em: <http://www.ncbi.nlm.nih.gov>. Acesso em: 10 Fev. 2015.

NVIDIA. Nvidia Corporation. Disponível em: <http://www.nvidia.com.br/page/ home.html>. Acesso em: 5 Maio. 2015.

NVIDIA[2]. Nvidia White Paper*. NVIDIA Solves the GPU Computing Puzzle.* Disponível em: http://www.nvidia.com/content/PDF/fermi_white_papers/N. Brookwood_NVIDIASolvesTheGPUComputingPuzzle.pdf. Set. 2009. Acesso em: 14 Jun. 2015.

NEEDLEMAN, S. B.; WUNSCH, C. D. *A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins.* Journal of Molecular Biology, 48;443-453. 1970.

OWENS, J. D. et al. *Graphics Processing Units- powerful, programmable, and highly parallel- are increasingly targeting general-purpose computing applications*. In: Proceedings of the IEEE, 2008. Journal IEEE. Vol.96, number 5. p. 879 – 899. 2008.

OPENCL. Disponível em: <*https://www.altera.com/products/design-software/ embedded-software-developers/opencl/overview.html*>. Acesso em: 5 Fev. 2015.

PAGLIOSA, P. A. *Tópicos sobre GPGPU em CUDA.* Faculdade de Computação Universidade Federal de Mato Grosso do Sul. Maio. 2013.

PASCOE, C. et al. *Reconfigurable Supercomputing with Scalable Systolic Arrays and In-Stream Control for Wavefront Genomics Processing.* Proc. Symp. Application Accelerators in High-Performance Computing. 2010.

PATTERSON D.A.; HENNESSY J.L. *Computer Organization and Design : The Hardware/Software Interface.* Morgan Kaufmann Publishers, Inc., 2nd.Edition. 2000.

PEARSON, W. R.; LIPMAN, D. J. *Improved tools for biological sequence comparison.* PNAS – Proceedings of the National Academy of Sciences, 85:2444–2448, 1988.

PRADO, A. C. *ARM e FPGA em um chip! A nova geração de FPGAs.* Disponível em: <http://www.embarcados.com.br/armefpga/>. Acesso em: 4 Fev. 2014.

PROCWIZARD. Gidel ProcWizard. Disponível em: <http://www.gidel.com/ ProcWizard.htm>*.* Acesso em: 3 Julho. 2014.

PROSDOCIMI, F. *Introdução à Bioinformática*. Disponível em : <http://www2. bioqmed.ufrj.br/prosdocimi/FProsdocimi07_CursoBioinfo.pdf>. Acesso em: 2 Maio. 2015.

QUARTUS. Quartus II IDE. Disponível em: http://www.altera.com/products/software/ quartus-ii/web-edition/qts-we-index.html. Acesso em: 13 Fev. 2014.

QUESTA. Questa Advanced Simulator. Disponível em: <http://www.mentor.com/ products/fv/questa/>. Acesso em: 4 Julho. 2015.

RACANIELLO, V. *Satellites - the viral kind.* College of Physicians and Surgeons of Columbia University. Disponível em: <http://www.virology.ws/2015/01/22/satellites-the-viral-kind/>. Acesso em: 13 Jul. 2015.

ROGNES, T. *Faster smith-waterman database searches with inter-sequence simd parallelisation*. BMC bioinformatics, 12(1):221. 2011.

ROCHE_454. Disponível em: <http://www.454.com/>. Acesso em: 15 Fev. 2015.

ROCHE_FLX. Disponível em: <http://my454.com/downloads/my454/documentation/technical-bulletins/TCB-09004_UsingMultiplexIdentifierAdaptorsForTheGSFLXTitaniumSeriesChemistry-BasicMIDSet.pdf>. Acesso em: 2 Jul. 2015.

RODINIA, Rodinia Benchmark Suite. Disponível em:: http://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Rodinia:Accelerating_Compute-Intensive_Applications_ with_ Accelerators. Acesso em: Jan. 2015.

SALVADOR, R. et al. *Self-Reconfigurable Evolvable Hardware System for Adaptive Image Processing*. Computers IEEE Transactions on (Vol. 62, Issue; 8), Abr. 2013.

SAVRAN, I.;GAO, Y.;BAKOS, J. D. *Large-Scale Pairwise Sequence Alignments on a Large-Scale GPU Cluster.* University of South Carolina, Copublished by the IEEE CEDA, IEEE CASS, IEEE SSCS, and TTTC. Fev. 2014.

SETUBAL, J.; MEIDANIS, J. *INTRODUCTION TO COMPUTATIONAL MOLECULAR BIOLOGY. Livro,* PWS PUBLISHING COMPANY. 1997.

SCHAUER, Bryan. *Multicore Processors – A Necessity*. Pro Quest – Discovery Guides. Released. Set. 2008.

SCALIONE, S. Disponível em: <http://www.laboratoriogene.com.br/ downloads/reportagem_exoma_ufmg.pdf>. Acesso em: 10 Jul. 2015.

SCIENGINES, SciEngines GmbH. Disponível em: <http://www.sciengines.com>. Acesso em: 12 Maio. 2015.

SCIENGINES_WP, Sciengines White Paper v2.1. *Accelerated Smith-Waterman on RIVYERA Hardware.* SciEngines GmbH 24118, Kiel, Germany. 2012.

SCITABLE, Scitable by Nature Education. Disponível em: < http://www.nature.com/scitable/ebooks/essentials-of-genetics-8/126430897#bookContentViewAreaDiv>. Acesso em: 2 Jun. 2014.

SIGNALTAP, SignalTap II Logic Analyzer Editor. Disponível em: <http://quartushelp.altera.com/13.1/mergedProjects/program/ela/ela_view_using.htm>. Acesso em: 11 Jul. 2014.

SHRIMANKAR, D. D.; SATHE, S. R. *Implementation of Parallel Algorithms on Cluster of Workstations.* 2nd IEEE International Conference on Parallel, Distributed and Grid Computing. 2012.

SOTIRIADES, E.; KOZANITIS, C.; DOLLAS, A*. FPGA based Architecture for DNA Sequence Comparison and Database Search.* Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International. Abr. 2006.

STERCK, L. et al. *How many genes are there in plants (... and why are they there)?.* Current Opinion in Plant Biology , 199-203. Fev. 2007.

THERMO, Thermo Fisher Scientific Inc. Disponível em: <https://www.thermofisher.com/br/en/home.html>. Acesso em: 11 Maio. 2015.

TOCCI, Ronald; WIDMER, Neal; MOSS, Gregory.*SISTEMAS DIGITAIS: princípios e aplicações.* Tradução Cáudia Martins.10. ed. São Paulo: PEARSON, 2007.

TRUONG, H. et al. *Large-Scale Pairwise Alignments on GPU Clusters: Exploring the Implementation Space.* Journal of Signal Processing Systems, volume 77, issue 1-2, pp 131-149. Out. 2014.

UNIPROT. Universal Protein Resource. Disponível em: <http://www.uniprot.org>. Acesso em : 5 Fev. 2015.

VARUZZA, L. *Introdução à análise de dados de sequenciadores de nova geração. Versão 2.0.1.* Disponível em: <http://lvaruzza.com/files/apostila_bioinfo_2.0.1.pdf>. Acesso em: 11 Jun. 2014.

VIANA, V. R.; MOURA, H. A. *Algoritmos para Alinhamentos de Sequências*. Revista Científica da Faculdade lourenço Filho - v. 7, n.1, 2010.

VENTER, J.C. et al. *The Sequence of the Human Genome.* Science 291, no. 5507, 1304-1351, Fev. 2001.

WIENBRANDT, L. *Bioinformatics Applications on the FPGA-Based High-Performance Computer RIVYERA.* W. Vanderbauwhede and K. Benkrid (eds.), High-Performance Computing Using FPGAs, pp 81-103, 2013.

XILINX, Xilinx Inc.Disponível em:<http://www.xilinx.com/>. Acesso em: 10 Fev. 2015.

XILINX[2]. Xilinx White Paper. *7 Series FPGAs Configurable Logic Block - User Guide*. Disponível em: < http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf>. Acesso em: 4 Nov. 2014.

YANG, C.T.; HAN T.F.; KAN H.C. *G-BLAST: a Grid-based Solution for mpiBLAST on Computational Grids.* Concurrency and Computation: Practice and Experience 21, 2009.

WATSON,J. D. ; CRICK,F.H.C. *Molecular Structure of Nuclec Acids".* Nature, Cavendish Laboratory, Cambridge, England, 1953.

WATERMAN,M.S.; SMITH,T.F. *Identification of common molecular subsequences.* Journal of Molecular Biology, 147(1):195-197, 1981.

ZAHA, A. et al. *Biologia Molecular Básica - 3ª edição.* São Paulo: Mercado Aberto, 2003.

GLOSSÁRIO

ASIC (Application Specific Integrated Circuits)
São circuitos integrados orientados para implementação de tarefas específicas

Bistream
Conjunto de bits gerados a partir de uma ferramenta de síntese que são carregados no FPGA para a sua programação.

DMA (Direct memory access)
Permite que certos dispositivos de hardware num computador acessem a memória do sistema para leitura e escrita independentemente da CPU.

DSPs (Digital Signal Processor)
São microprocessadores especializados em processamento digital de sinal usados para processar sinais de áudio, vídeo, etc., quer em tempo real quer em off-line.

FIFO (First-In-First-Out)
Estrutura de dados baseados em fila, onde o primeiro dadoarmazenado é o primeiro a ser lido.

Memória SRAM (Static Random Access Memory)
É um tipo de memória de acesso aleatório que mantém os dados armazenados desde que seja mantida sua alimentação.

Pipeline
Técnica de hardware que posibilita a aceleração das operações de um sistema digital através da utilização paralela de seus recursos.

Speedup
Indica ganho de desempenho entre arquiteturas ou implementações de um código. É definido pela relação entre os seus tempos de execução.

APÊNDICE A

Listagem final de um alinhamento

```
Sequência 01: Cotton leaf curl Burewala betasatellite
Sequência 02: Cotton leaf curl Bangalore betasatellite

Perc.= 39.71 %

Seq.01: 1   accgtgggcgagcggtgcc_cggtggccgcg__c____agtgggt_cgt_ 41
            |||||||||||||||||| |: :|::::||  :    ::::::| |:|
Seq.02: 1   accgtgggcgagcggtgcctct_ggcgaacgcagtgggtcccactgcttg 49

Seq.01: 42  _c_ccactgct_tgtc___ttgac_t_ggatttgacttctaatgg_gcca 82
             : ::| :: : ::|:   : ::: :::|::: : : : :  |:: ::::
Seq.02: 50  tttgga_ct_gga_tttgac_ttgtattgggc_c_a_a_g__ttaatggg 89

Seq.01: 83  atttaatgggcttggaagtgattgggcctttgaagatg_gccttttatg_ 130
            :| |::::|::||:|  |:::| : : : :::::|::  ::: ||| ::
Seq.02: 90  ct_tggaagtgtttg__ggcct_t_t_g_aaattggc__c_t_ttt_atg 127

Seq.01: 131 gat___ta___tgg_g_tctgtttgttgttggtga___tatg_t__g_tg 165
            :::   ::   | : | ::::|:|: :||::::::   : |: |  : ::
Seq.02: 128 tggtttagggct_ctgtctgttgta_agtgatatgtgtg_taattatgcc 174

Seq.01: 166 ttaaatatgcattgc_tggtttgtggttggaa_tttaaacggtgaacttc 213
            ||:::::: ::||:: : | | : |:: ::: :::  :: :::|::||:
Seq.02: 175 ttgctg_g__tctgtg_t_t_t_g_gaa_tttaaac__gg_tgaatgttt 212

Seq.01: 214 tt___attgaat__atgta____t__gg_ttcg_a_tt_ac_a_tc_cat 245
            |:   |:|:::|  :::|:    :  :: |::: : :: |: ::|: ::|
Seq.02: 213 tattgaatacgtacggttcgattacatt_tattcctagaatatctgggtt 261

Seq.01: 246 tcccaatatctctgggtt_ttcaa_gtacaagtatatcaagtctgtgaaa 292
            |:| :|:::|:::::: : ::::: :| : :: ||  :| :  ::::::
Seq.02: 262 ttc_tagtacaagtat_atcaagcctt_t_ga_at__ta_t__gtcttc_ 300

Seq.01: 293 _ctatatct_tctatct_cg_atctcttctatctttgccccgttgtatgc 338
             :|:|: ::: ||:::: :: :|  ::  : :|    |   |||||:|||
Seq.02: 301 tatctc_gat_ctcatctatttt__tg__c_cc____c___gttgtttgc 336

Seq.01: 339 gaataggaaattcgctatgatg___ctccct_tcaaagccgttgaagtcg 384
            |||||||||||||||||||||  || |  : ||||||||||||| :: :
Seq.02: 337 gaataggaaattcgctatgatgtt_c_c__cttcaaagccgttga_ag_t 380

Seq.01: 385 aa_tg__ga_acgtgta_tgtcttcg_tacgtgtact_ggac_g_____a 421
             : ::  || :::::::: |:|| ::: :: :::::::| :::: : :   :
Seq.02: 381 _cgaatggaacgtgcatgtctc_cgtacg_tgtacttgacgat_cccttc 426

Seq.01: 422 _tccc_t_tcatacttg_attagcga_tgg_tga_ctttgtggata_gta 463
             |:|: : |:| : ::: ::|::|:: ::| ::| |: :::::| : ::|
Seq.02: 427 atacttgatga_t_cgctgatgacttggtggatacca_acctca_tgtga 472

Seq.01: 464 _tcc_tcatgtgaatga_ag_atct_tcatattct__ccatgatgcgaac 506
             |:: | ||  ::: :: :| : |: :|||  ::|  ::|:  ::||| |
```

```
Seq.02: 473 atgaat_at___ct_tcatg_t_cctccat__gatgcgtac__gtcga_c 510

Seq.01: 507 gtcgact_atgaacctgactccc_tccttgt_tt_gtt_ccgctcgttgt 551
            :  |::: ::  : ::::   || |:| |:: || :::::::: ||:||||
Seq.02: 511 t__gtgaacc__t_gact___ccctgc_ttggttgtccc_gc_tccttgt 549

Seq.01: 552 catttctgcttatttgatggaa_ttgttt__agtgtgcattcatttatag 598
            |||||:||||||||:||||| | |||:||  :: ::||:: ::|||||||
Seq.02: 550 catttttgcttattggatgg_aattggttatta_tggctg_tgtttatag 596

Seq.01: 599 _ac_ttaatgaac_agatattttgtgtagttgtgtggttgtgagtgattt 645
             :|: :| :::|: ||||||||||||||:|||:||||||||||:|||||| :
Seq.02: 597 atcc__a_atgatgagatattttgtgttgttatgtggttgttagtgat_c 642

Seq.01: 646 attatt_atgtgattg_tc__cattaaagggataaagtg_atgatggaga 690
            :||::| ||||||| ::|:  : ||||||||:||||| : ||||||||||
Seq.02: 643 cttgatcatgtgat_tgtg_ga_ttaaagggttaaag_atatgatggaga 688

Seq.01: 691 cgt_a__ttac_ac___gt_gttgtca_tg_ttggcttg_a_aatcttta 728
            :|| :  :::: ::   :| :|:|  : || :::::|:: :::||:::|:
Seq.02: 689 tgtacttcatgtgtttgatgttgg__tttgagac_ttctatacatgggtt 735

Seq.01: 729 tacatgggtttgtaccgtatatacc_tatata_tacggatagaaaacgga 776
            |::  ::::|: ||:::: |:||:: :::::: :::|:|: :|| |:
Seq.02: 736 tgt__accgta_tatgcc_tctatatacggataggagtag_aaa_aa___ 776

Seq.01: 777 tgag_aa_aaggaaaaacgg_aactgaag__aggaaaa_acaaagaaaa_ 819
             | |::: | |:|||| |: ||::::::  : |||:| : ||||||||
Seq.02: 777 _g_gat_aa_gaaaaa__gagaaaactgaact_gaagagg_aaagaaaaa 818

Seq.01: 820 _g_aaaca_ag_g_a__ta_tattattta__tgaaagaaatggg_agcg_ 857
             | || :: || : |  :: ::| :: ::: : || :|| :::: :|::
Seq.02: 819 agaaa_ataaggaaagaactact__a_atcaa_aa_aaa_gaaatgggag 861

Seq.01: 858 cagcgaat_cgaaacaggaaaacccaaggaaagagaaaaaa_t_aa_aaa 903
            |:::|::: |::||:::|: || :|  ::   | :::|| | : ||:|:|
Seq.02: 862 cgcagcgacccgaaacagg_aa_ac__cc___g_aggaa_agagaaaa_a 901

Seq.01: 904 gtaaagagaaaaaaata_taa_att___ctaaaacgtcatcg_t_ttgag 946
            :||||:|||||:|||:| ::| |::   ::||| |||||||| | ::|||
Seq.02: 902 ataaaaagaaacaaaaaaataaaaactcgaaaa_cgtcatcgtttgagag 950

Seq.01: 947 ag_taat_aaaaaaaa_aaga_aaaaacaaa_aa_acatatc_cgaaaac 989
            :: ::|: ||:|| |::|| | |::|| ||: :| :| :|:: || |:|:
Seq.02: 951 gagataaaaa_aa_agaaa_aaactaa_aatagaacc_gagaacg_acat 994

Seq.01: 990 gtt_gt_cgtttgaaggtgactgtgtg_gtttta_ccatttactgtgtgg 1035
            ::| :: :|:|::::|::| :::|:|: ::: |:::|::: :|:||::::
Seq.02: 995 cgttcgaaggtgactgagg_tgtttta_cca_tttactgg_gcggtaaat 1040

Seq.01: 1036 taaatggtaagtg__at_t_gaagaatagttaaaaatggagacaccgatt 1080
             :   ::: ||::|  |: : ||:|:|: ::  ||:| : :||:::|:|:
Seq.02: 1041 g___gta_aatgggtaaaaagaggtaa_aa__aata_t_tgagtaccaa_ 1080

Seq.01: 1081 _aggtaaattgta_ctccaa__tatattggg__g_tacaattggggactc 1123
              |||||||||::: ::|:|:  :::::|:::  : ||:|:|:: :::: :
Seq.02: 1081 taggtaaattgagtaccgatatatcggtactcaattagagtct_catt_t 1128
```

```
Seq.01: 1124 ataaattacctttccta_aaatacccc__tatttctgtgtctggaaggtg 1170
             :::| :  |:||:|:::: |||||||| : |||:::::::::| ||:|
Seq.02: 1129 tata_c__ctttccg_ta_aataccccgc_ttttatgtctaata_ggcg 1171

Seq.01: 1171 cg_tggaa_gtgcgctgaaaaaggt___gttcttc_tctcacctaaaacc 1214
             || :|||: |:|  ||||:|||| |   : ::|:|::  :::|:  ::::
Seq.02: 1172 cgtcggagtgcg__ctgataaag_tgaac_ctt_ctc__tctcc__gttt 1212

Seq.01: 1215 c_gcc_g_g___aacg_cca_aaa___ctg_gctgt_tg___ccggcatc 1248
             : | : : :   :||: ::: :::   :|: | :|: ::   :::: ::|
Seq.02: 1213 tgg_gactctaatacaatttcccggttatcgg_cggcgaatttttc_gac 1259

Seq.01: 1249 aattt_a__cgac___ac__gc_gcggcgg_tgtgtacccctgggagggt 1288
             |:::: :  ||::   :: ::| |:|::|| : :||| ||:: ::  :|:
Seq.02: 1260 a_cgcgcggcggtgtgtacccctgggagggta_ggta_ccac_ta__cgc 1303

Seq.01: 1289 agaaaccactacgctacgcagcagccttagc_t_acgccggagcttagct 1336
              : |::   :|: |:  ||  |::    ||| | ||||||||||||||
Seq.02: 1304 _t_acg___cag_ca__gc__ctt____agc_t_acgccggagcttagct 1337

Seq.01: 1337 cgcc_cacgtt_tta__atatt 1354
             | :| |||| | :||  || |
Seq.02: 1338 c_gcccacg_ttctaatat__t 1355
```