UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

EVALDO BEZERRA DA COSTA

UM FRAMEWORK UTILIZANDO COMPUTAÇÃO PARALELA PARA MONTAGEM DE SEQUÊNCIAS DE DNA

Rio de Janeiro
2022

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

EVALDO BEZERRA DA COSTA

UM FRAMEWORK UTILIZANDO COMPUTAÇÃO PARALELA PARA MONTAGEM DE SEQUÊNCIAS DE DNA

Tese de Doutorado submetida ao Corpo Docente do Departamento de Ciência da Computação do Instituto de Matemática, e Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para obtenção do título de Doutor em Informática.

Orientador: Marcello Goulart Teixeira
Coorientador: Gabriel Pereira da Silva

Rio de Janeiro
2022

CBIB Costa, Evaldo Bezerra da

Um Framework Utilizando Computação Paralela para Montagem de Sequências de DNA / Evaldo Bezerra da Costa. – 2022.
134 f.: il.

Tese (Doutorado em Informática) – Universidade Federal do Rio de Janeiro, Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Programa de Pós-Graduação em Informática, Rio de Janeiro, 2022.

Orientador: Marcello Goulart Teixeira.
Coorientador: Gabriel Pereira da Silva.

1. Bioinformática. 2. Sequenciamento. 3. Framework. 4. Análise. 5. Montagem. 6. Computação Paralela. – Teses. I. Teixeira, Marcello Goulart (Orient.). II. Silva, Gabriel Pereira da (Coorient.). III. Universidade Federal do Rio de Janeiro, Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Programa de Pós-Graduação em Informática. IV. Título

CDD

EVALDO BEZERRA DA COSTA

Um Framework Utilizando Computação Paralela para Montagem de Sequências de DNA

Tese de Doutorado submetida ao Corpo Docente do Departamento de Ciência da Computação do Instituto de Matemática, e Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para obtenção do título de Doutor em Informática.

Aprovado em: Rio de Janeiro, 10 de outubro de 2022.

Prof. Dr. Marcello Goulart Teixeira (Orientador)

participação por videoconferência

Prof. Dr. Gabriel Pereira da Silva (Coorientador)

participação por videoconferência

Profa. Dra. Maria Luiza Machado Campos - PPGI-UFRJ

participação por videoconferência

Profa. Dra. Silvana de Cássia Paulan - UNESP

participação por videoconferência

Prof. Dr. Diogo Antonio Tschoeke - UFRJ

Rio de Janeiro
2022

"Feliz a nação que tem o Senhor por seu Deus, e o povo que ele escolheu para sua herança."
Salmos 32:12

AGRADECIMENTOS

Ao Senhor meu Deus, que sempre está ao meu lado e me fortalece durante todos os momentos da minha vida.

Desde pequeno que sempre escutei dos meus pais, que estudar é a única forma de transformar vidas. Que só através da educação é possível construir uma nação forte, onde todos somos iguais. Obrigado mãe, Obrigado pai.

A minha doce Rosalice, mulher que escolhi para viver ao meu lado por toda a vida. Pelas horas que não pude estar ao seu lado, dedicando-me aos meus estudos. Ao meu anjo Maria Eduarda, amor da minha vida.

Aos meus orientadores os professores Marcello G. Teixeira e Gabriel P. Silva. Por seguirem comigo nesta longa jornada, que foi desafiadora e ao mesmo tempo fantástica. Durante este período construímos mais que uma relação de aluno e professores, tenho grande satisfação em tê-los como amigos.

A todos os professores do PPGI, os quais tive a honra e o privilégio de conhecer e ter sido aluno. Aos colegas que conheci durante o curso e que me ajudaram durante esta jornada. A todos os membros do PPGI, meu muito obrigado.

Gostaria de fazer um agradecimento especial ao Dr. Daniel Zerbino e todos que contribuíram para o desenvolvimento da nova versão do Velvet usando OpenACC com sugestões, testes e validação dos módulos do programa.

RESUMO

Costa, Evaldo Bezerra da. **Um Framework Utilizando Computação Paralela para Montagem de Sequências de DNA**. 2022. 118 f. Tese (Doutorado em Informática) - PPGI, Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2022.

O uso de recursos computacionais em Biologia teve início na década de 80, possibilitando o sequenciamento de pequenos fragmentos de DNA. A partir desse período, o uso de recursos computacionais ganhou importância no estudo da Biologia.

Com o desenvolvimento de novas técnicas e métodos de sequenciamento, alinhamento e montagem de genoma, a quantidade de dados gerados tem aumentado consideravelmente, principalmente nas últimas décadas, exigindo o desenvolvimento de *frameworks* de programação capazes de processar e analisar um montante cada vez maior de informações. Atualmente, existem *frameworks* que buscam utilizar de maneira mais eficiente os recursos computacionais disponíveis para alcançar o maior desempenho possível, tornando, assim, o processo de manipulação dos dados mais simples e intuitivo para os usuários.

Nesse contexto, a presente tese tem por objetivo o desenvolvimento de um *framework* que, através da utilização de técnicas e métodos de computação paralela que façam uso de aceleradores do tipo *manycore* ou unidade de processamento gráfico (GPU), melhore o desempenho dos programas utilizados para montagem de DNA com a melhor utilização dos recursos computacionais.

Palavras-chave: Bioinformática, Sequenciamento, Framework, Análise, Montagem, Computação Paralela.

ABSTRACT

Costa, Evaldo Bezerra da. **Um Framework Utilizando Computação Paralela para Montagem de Sequências de DNA**. 2022. 118 f. Tese (Doutorado em Informática) - PPGI, Instituto de Matemática, Instituto Tércio Pacitti, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2022.

Computational resources applied in the Biological area has began in the 1980s, making possible the sequencing of small DNA fragments and gaining importance since then.

The development of new techniques and methods for genome sequencing, alignment, and assembly, resulted in a considerable increase in generated data, mainly in the last decades, requiring the development of programming frameworks capable of processing and analyzing large amounts of information. Currently, there are frameworks aiming the more efficient use of the computational resources available, to achieve the highest performance, thus making the data manipulation process easier and user-friendly.

In this context, this thesis presents a framework that uses parallel computing techniques and hardware accelerators, such as manycore and graphics processing units (GPU), to improve the performance and accuracy of the results of programs used for DNA assembly.

Keywords: Bioinformatics, Sequencing, Framework, Analysis, Assembly, Parallel Computing.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

No início da década de 80, um método desenvolvido por Sanger e colaboradores possibilitou o sequenciamento de fragmentos pequenos de DNA através da incorporação de nucleotídeos alterados quimicamente (SANGER; COULSON, 1975). A partir de então, diferentes inovações tecnológicas, algumas delas baseadas na técnica de Sanger, possibilitaram o sequenciamento de genomas completos. Essas inovações tecnológicas receberam o nome de Sequenciamento de Nova Geração (do inglês, *Next Generation Sequencing* - NGS) (SALZBERG et al., 2011). Além de possibilitarem o sequenciamento de genomas completos, as técnicas NGS requerem menor quantidade de DNA purificado e descartam processos laboriosos e complexos, como a clonagem dos fragmentos de DNA-alvo em colônias bacterianas para a construção das bibliotecas.

Os principais métodos de sequenciamento processam milhares ou até milhões de fragmentos de DNA, chamados de sequências de leitura (*reads*), que apresentam tamanho desde curtas (menos de 100 pares de bases) até longas (com milhares de pares bases), dependendo do método utilizado (COSTA; SILVA; TEIXEIRA, 2020).

Se, por um lado, esse avanço tecnológico propiciou importantes avanços em diversas áreas, como produção animal e vegetal, parasitologia, oncologia etc., por outro, a enorme produção de informação gerada através dos sequenciamentos genômicos resultou em desafios para o armazenamento, processamento e análise desses dados. Tais desafios necessitam da ação conjunta entre especialistas das áreas biológicas e de cientistas da computação, culminando com o desenvolvimento de diversas ferramentas que possibilitam o pleno uso das informações contidas no genoma de todo ser vivo. Esse trabalho conjunto deu início à área que hoje conhecemos como

Bioinformática.

A bioinformática é uma área de estudo interdisciplinar que envolve outras áreas, entre elas esta a biologia, matemática e a ciência da computação. O uso da computação na biologia tem por objetivo ajudar a resolver problemas que podem ser desde no entendimento de questões simples a solução de grandes desafios.

Quando a computação começou a ser utilizada na biologia, o conhecimento e as técnicas para análise e geração de dados eram reduzidos. Com o passar dos anos novas técnicas e descobertas foram feitas, resultando em um aumento exponencial na quantidade de informação a ser processada. Os recursos computacionais também precisam ser utilizados de forma mais eficiente e em maior quantidade, para entender e buscar respostas para essa nova demanda.

O marco inicial da bioinformática se deu com o trabalho desenvolvido por Fleischmann e colaboradores (FLEISCHMANN et al., 1995), que resultou na criação de um banco de dados biológicos para a espécie *Haemophilus influenzae*.

Quando falamos dos desafios associados à bioinformática, uma dos primeiros em que pensamos é como vamos interpretar os dados que são gerados. A quantidade de dados gerados a partir do sequenciamento genômico está em ampla expansão. Tomando como exemplo a espécie humana, estima-se que até 2025 sejam sequenciados entre 100 milhões e 2 bilhões de genomas.Para processar todos esses dados é necessário uma grande quantidade de recursos computacionais.

Nos últimos anos, surgiram novas arquiteturas paralelas com alta capacidade de processamento, que têm por característica principal o uso de unidades de processamento ou processadores para a realização de várias tarefas no qual um problema pode ser subdividido, alcançando um elevado grau de paralelismo através da utiliza-

ção de um número de processadores cada vez maior, como as arquiteturas *multicore* e *manycore*.

Um outro tipo de arquitetura com destaque pelo seu alto poder computacional são os aceleradores, que foram desenvolvidos com o objetivo de oferecer uma arquitetura massivamente paralela e têm capacidade de cálculo muito superior à de um processador convencional. Os aceleradores são projetados para o uso em operações de cálculo de ponto flutuante e possuem memórias cache em sua estrutura, permitindo que todas as *threads* acessem os mesmos dados sem utilizar a memória principal do computador. Entre os tipos de aceleradores temos os modelos da NVIDIA GPU, AMD GPU e o Intel Xeon Phi. Cada um desse modelos possuem características específicas, mas todos possuem uma grande capacidade de processamento de dados.

Quando desenvolvemos um programa ou utilizados um recurso computacional, uma das questões que surgem é saber o quanto paralelizar na aplicação, quanto o recurso utilizado oferece de paralelismo e as possibilidades de obtenção de um ganho de desempenho significativo. Para avaliação do desempenho de um sistema de processamento paralelo, são consideradas, por sua maior importância, as métricas *speedup* (isto é, ganho de desempenho ou aceleração), eficiência e escalabilidade (SILVA, 2018).

Atualmente, o desenvolvimento de recursos computacionais com capacidade de armazenar, processar, analisar os dados biológicos e realizar a montagem de genomas com robustez e agilidade tem configurado o maior desafio da bioinformática ao longos dos últimos anos. Nessa busca, computadores com elevado poder computacional e *frameworks* mais eficientes vêm sendo utilizados, aumentando a capacidade de paralelização dos processos, o que eleva não apenas a capacidade de processamento, como também reduz o tempo necessário para a execução de cada trabalho.

Uma outra demanda importante é a necessidade do uso de *frameworks* mais eficientes, de modo a aumentar a capacidade de paralelização e reduzir o tempo necessário para a execução de cada trabalho. Os *frameworks* têm como principal objetivo a execução de um maior número de tarefas relacionadas ao tratamento de dados sequenciados, como análise de qualidade, alinhamento e montagem. Têm como características o uso de interface gráfica ou acesso por navegador, com isso os usuários não necessitam usar a linha de comando para a execução de uma determinada tarefa, além de serem intuitivos e amigáveis.

Entre os *frameworks* comumente utilizados em bioinformática temos o Galaxy, que é um sistema de fluxo de trabalho científico e uma plataforma de integração de dados para dados biológicos baseada em três pontos: o primeiro são os dados, que podem ser enviados ou utilizar os dados já existentes no banco de dados do Galaxy (GIARDINE et al., 2005). O segundo ponto é a análise que é feita através das ferramentas disponíveis no *framework*. Por último a execução completa de todo o fluxo de trabalho necessário como por exemplo, a execução de uma montagem.

Também temos o *frameworks* BioExtract Server que é um sistema aberto, baseado na Web, em que os usuários podem construir seus próprios *pipelines* (LUSHBOUGH et al., 2010). Os fluxos de trabalho do BioExtract Server podem ser compartilhados. Os usuários podem exportar um fluxo de trabalho como um arquivo XML e compartilhá-lo com outros colaboradores.

E o *frameworks* MELC Genomics 1.0 que foi desenvolvido integrando alguns dos programas mais usados para um *pipeline* de montagem completo. Com uma interface web simples, intuitiva e amigável, o MELC Genomics podia ser executado em qualquer navegador, além de ser simples e rápido (COSTA, 2018).

O uso desses *frameworks* tornam os processos de bioinformática como ali-

nhamento, verificação da qualidade de dados gerados e montagem mais simples e rápidos, pois é possível em um único sistema executar vários processos ao mesmo tempo.

A proposta desta tese é desenvolver uma versão do MELC Genomics 2.0 que tem como objetivos apresentar uma interface de usuário renovada, com autenticação dos usuários, com opções adicionais que incluem funcionalidade como verificação da qualidade dos dados antes e depois da montagem, acompanhamento da utilização dos recursos da máquina em tempo real, além da modificação do código do montador de sequências para realizar a montagem utilizando aceleradores do tipo *manycore* ou GPU, com o objetivo de diminuir o tempo de montagem.

Para a escolha do montador utilizado no *framework* apresentado nesta tese, foi feita uma análise comparativa avaliando diversos montadores de genoma *de novo*, utilizando critérios tais como a utilização de recursos computacionais e o tempo de montagem. Baseado nos resultados apresentados nesses estudos, optou-se por utilizar o montador Velvet, pois, além de ter apresentado bons resultados na montagem de genomas, trata-se de um programa livre, e seu código permite fazer a implementação utilizando modelo de programação para acelerador. Como parte principal do *framework* será o desenvolvido uma versão do montador de sequências Velvet usando o paradigma de programação OpenACC.

O montador Velvet pode ser usado para construir rapidamente sequências longas e contínuas a partir de conjuntos de dados de sequências curtas (ZERBINO; BIRNEY, 2008). Essa função é útil, principalmente ao se estudar dados de um novo organismo para o qual um genoma de referência ainda não foi montado ou ao tentar determinar a origem de *reads* não mapeadas.

O Velvet consiste em dois programas que são usados para gerar os resultados:

velveth e velvetg. Velveth lê os arquivos de sequências que foram passados como entradas de dados e cria dois arquivos de saída chamados Roadmaps e Sequences, que serão utilizados pelo velvetg. O velvetg usa os arquivos de saída gerados pelo velveth e constrói o grafo *de Bruijn*.

O OpenACC é um modelo de programação aberta para computação paralela, desenvolvido com o objetivo de simplificar a programação paralela, oferecendo alto desempenho e portabilidade entre diversos tipos de arquiteturas: *multicore*, *many-core* e GPUs. O OpenACC é compatível com os modelos de programação OpenMP e MPI, ambas as abordagens podem ser combinadas com o OpenACC. Em geral, as diretivas do OpenACC são muito semelhantes às do OpenMP. Em relação ao CUDA, OpenACC é totalmente compatível, tornando a necessidade de alteração do código a menor possível (COSTA; SILVA, 2019). OpenACC é baseado em diretivas e cláusulas. Também é possível realizar a movimentação de dados entre a GPU e o servidor com isso é possível executar tarefas diretas na GPU e devolvendo o resultado dos dados processados para o servidor.

O montador Velvet é composto por vários arquivos. Alguns desses arquivos de código não utilizam nenhum paradigma de paralelismo. Os arquivos do montador que utilizam programação paralela têm como caraterística principal uma grande quantidade de laços, sendo esta uma das condições ideais para o uso do OpenACC.

Para a execução dos testes e verificação da funcionalidade e eficiência da versão em OpenACC do montador Velvet foram utilizados três tipos de genomas que variam em tipo e tamanho, sendo dois genomas de bactérias e o cromossomo humano 21. Os resultados apresentados durante a execução da nova versão do montador mostram que o uso do OpenACC tem um melhor desempenho em relação a versão do montador usando OpenMP.

Esta tese está dividida em capítulos. No Capítulo 2, são apresentados alguns conceitos básicos. No Capítulo 3, é feita uma descrição das principais técnicas e métodos de sequenciamento, alinhamento e montagem. No Capítulo 4 serão apresentados alguns *frameworks* utilizados em bioinformática e será descrito o *framework* desenvolvido nesta tese. O Capítulo 5 descreve o montador desenvolvido para o uso em acelerador. No Capítulo 6, são apresentados os resultados de avaliação dos montadores, e o Capítulo 7, serão apresentadas as conclusões e trabalhos futuros.

2 CONCEITOS BÁSICOS

Este capítulo tem por objetivo apresentar alguns conceitos de arquiteturas paralelas e bioinformática.

2.1 Bioinformática

O termo biologia significa basicamente "estudo da vida". Informática é um termo usado para descrever o conjunto das ciências relacionadas à coleta, armazenamento, transmissão e processamento de informações em meios digitais [1]. A bioinformática é uma campo de estudo interdisciplinar que envolve diversas áreas, tais como biologia, ciência da computação, engenharia, matemática e estatística, com o objetivo de analisar e interpretar dados biológicos, desenvolver novas técnicas ou ajudar a resolver problemas biológicos com o uso de recursos computacionais, que de outra forma não seria possível solucionar (Figura 2.1).

De uma forma mais simples, a bioinformática é uma ciência que une os dados e as informações fornecidos pela biologia e os programas desenvolvidos na computação para interpretar e processar esses dados de forma significativa, ou ainda, a bioinformática faz uso da computação para uma melhor compreensão da biologia (SPENGLER, 2000). A bioinformática tem diversas aplicações dentro da biologia: desde o armazenamento de dados de DNA e a modelagem matemática de sequências biológicas, até a análise de mecanismos envolvidos em doenças humanas, além da modelagem e compreensão da história evolutiva da vida (SEARLS, 2010).

[1] https://pt.wikipedia.org

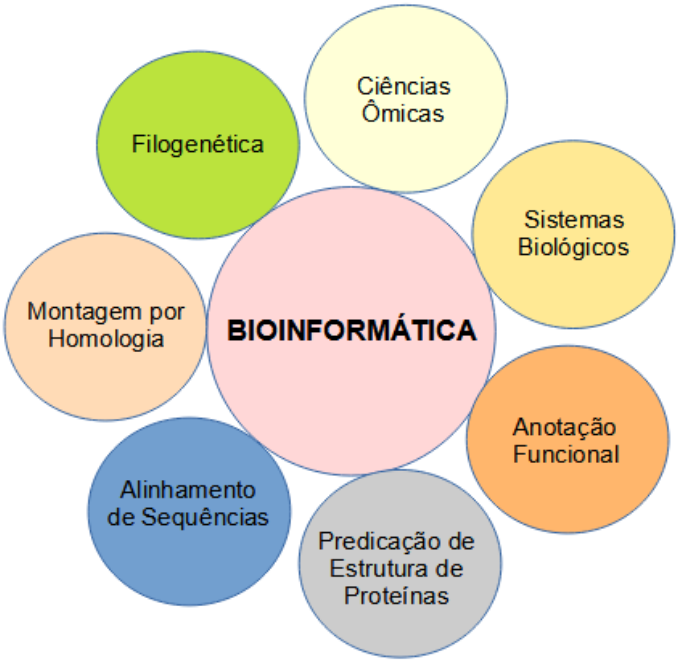

Figura 2.1: Uma relação de aplicações que podem se beneficiar com uso da bioinformática (SPENGLER, 2000)

No início da década de 80, o aumento significativo da quantidade de dados gerados através do sequenciamento de genomas completos obrigou o uso de recursos computacionais em diversas áreas da Biologia (HOGEWEG, 2011). Essa demanda por recursos computacionais aumentou enormemente a partir da década de 90, em virtude da massiva quantidade de dados biológicos gerados em grande parte pelo Projeto Genoma Humano e pelos rápidos avanços na tecnologia de sequenciamento de DNA. Como consequência, a bioinformática é uma das áreas da ciência de maior desenvolvimento atualmente, superando paulatinamente todos os desafios que têm sido apresentados durante essa evolução.

Desafios em Bioinformática

Um dos primeiros desafios associados à bioinformática que podemos destacar é a análise e interpretação da enorme quantidade de dados que são gerados, e o desenvolvimento de ferramentas e recursos computacionais tem sido a forma de conseguirmos atingir esses objetivos.

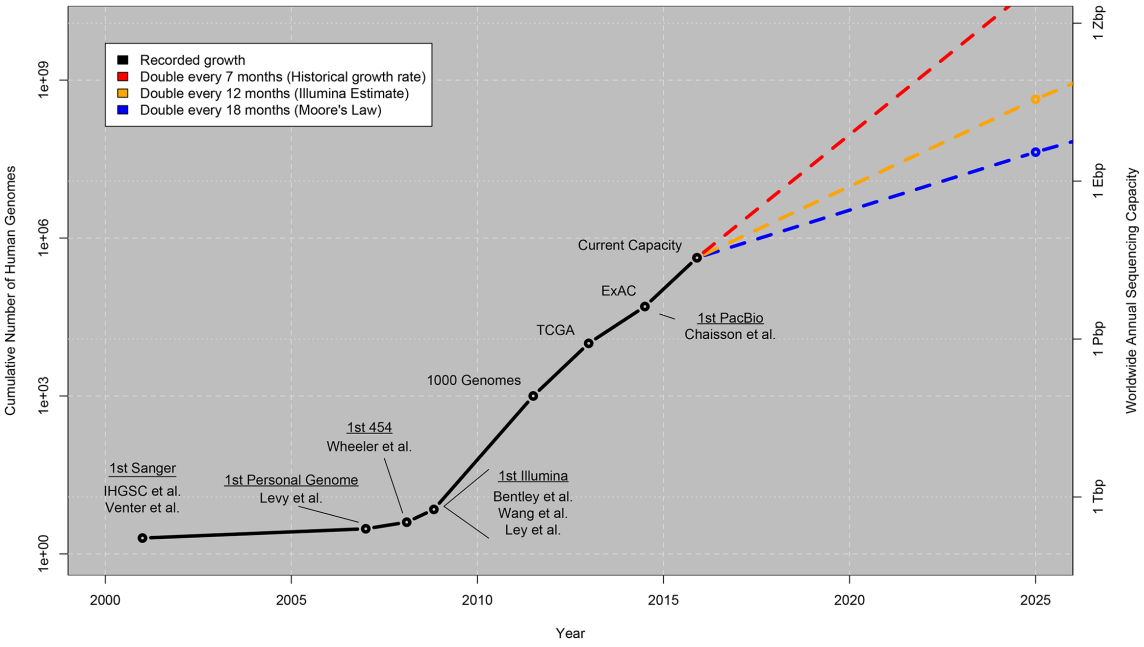

Figura 2.2: Crescimento da quantidade dos dados de DNA gerados de acordo com a evolução das técnicas de sequenciamento (linhas inteiras) e uma estimativa desse crescimento até o ano de 2025 (linhas tracejadas). Tbp: pares de base em *terabyte*; Pbp: pares de base em *petabyte*; Ebp: pares de base em *exabyte*; Zbp: pares de base em *zettabyte* (Zbps) (STEPHENS et al., 2015)

A quantidade de dados gerados a partir do sequenciamento genômico é sempre crescente: tomando como exemplo a espécie humana, estima-se que até 2025 sejam sequenciados entre 100 milhões e 2 bilhões de genomas, cujo tamanho dos dados armazenados pode ser entre 30 e 100 vezes o tamanho do próprio genoma (BATLEY; EDWARDS, 2009). Esse cenário conduz à chamada era do Big Data,

exigindo programas e linguagens de programação capazes de analisar quantidades de informação cada vez maiores.

A Figura 2.2 mostra o crescimento na geração de dados de sequenciamento de DNA, tanto no número total de genomas humanos sequenciados, apresentado no eixo esquerdo, quanto na capacidade anual mundial de sequenciamento apresentada no eixo direito. Notem que ambos eixos estão em escalas logarítimicas. Os valores a partir de 2015 representam uma projeção sob três possíveis curvas de crescimento, estimadas até 2025.

2.2 Arquiteturas Paralelas

As arquiteturas paralelas têm por característica o uso de diversas unidades de processamento ou processadores para a execução em paralelo das múltiplas tarefas nas quais um problema pode ser subdividido. Um elevado grau de paralelismo é alcançado através da utilização de um número cada vez maior de processadores. As arquiteturas *multicore*, *manycore* e GPU são exemplos que utilizam processamento paralelo maciço e que têm sido aplicadas em áreas diversas, como geofísica, sequenciamento genético, simulações de modelos matemáticos, previsão do tempo, entre outras (SILVA, 2018). Apresentamos a seguir os conceitos e suas características referentes aos tipos de arquiteturas paralelas:

- **Multicore**
  É o termo definido para descrever a utilização de mais de um núcleo em um processador, normalmente em único encapsulamento compartilhando uma mesma memória global, mas onde cada núcleo possui uma cache independente para execução em paralelo dos programas.

- **Manycore**

  Os processadores *manycore* são uma extensão do conceito do *multicore*, com um alto grau de processamento paralelo, variando de dezenas a centenas de núcleos de arquitetura independente e simplificada, que coexistem em um único encapsulamento.

- **GPU**

  São implementadas na forma de aceleradores, acoplados a um processador hospedeiro através de um barramento dedicado, onde são utilizados milhares de núcleos de processamento paralelo maciço com foco na eficiência energética e para aplicações com demandas que melhorem o *throughput*.

2.2.1 Aceleradores

Os aceleradores foram desenvolvidos com o objetivo de oferecer uma arquitetura massivamente paralela e têm capacidade de cálculo muito superior a de um processador convencional, embora tal poder seja apenas aproveitado em problemas paralelizáveis em nível de dados. Outra característica é a grande quantidade de *threads* que podem ser executados, existindo, atualmente, aceleradores que possuem mais de 5000 núcleos de processamento.

Os aceleradores são projetados para o uso em operações de cálculo de ponto flutuante, e quando utilizados para operações convencionais podem apresentar baixo desempenho em relação ao uso de arquitetura convencional. Os aceleradores possuem memórias cache em sua estrutura, permitindo que todas as *threads* acessem os mesmos dados sem utilizar a memória principal do computador.

GPU NVIDIA

Desenvolvida pela NVIDIA nos anos 90, essa GPU se baseia em um grande número de núcleos para processamento paralelo maciço com foco na eficiência energética e aplicações com demandas que melhorem o *throughput*. Inicialmente desenvolvidos com o objetivo de atender à área de jogos, rapidamente mostrou-se muito eficiente em outras áreas, incluindo a bioinformática (Figura 2.3).

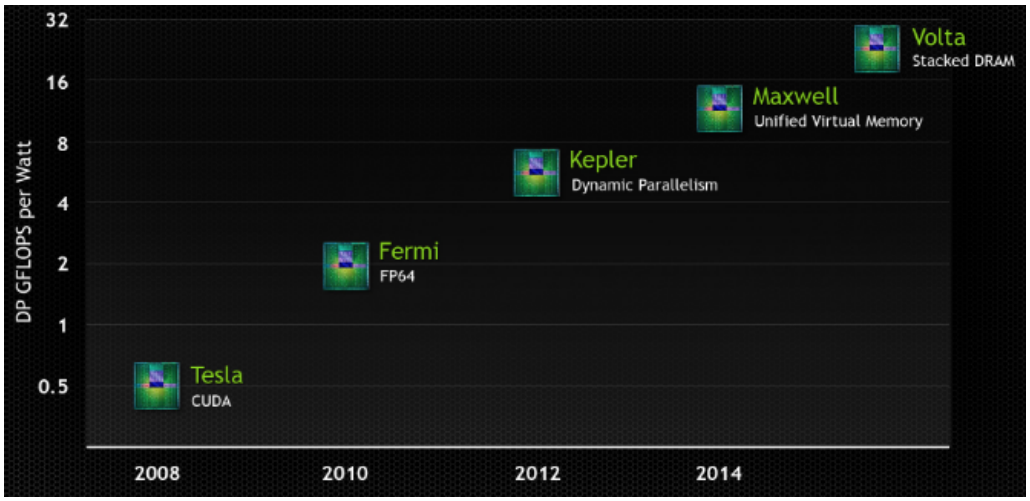

Figura 2.3: Relação de família de GPU da NVIDIA (NVIDIA, 2014)

As arquiteturas dos aceleradores gráficos (GPUs) são bem diferencidas das arquiteturas dos processadores convencionais. O paralelismo nos aceleradores gráficos é explorado através de um conjunto maçico de multiprocessadores de fluxo (*streaming multiprocessors (SM)*), executando em paralelo e de forma sincronizada trechos computacionalmente intensivos, chamados de *kernels*, das diversas aplicações (NVIDIA, 2014).

Na Figura 2.4 verificamos que o acelerador gráfico possui uma arquitetura distinta, com diversos níveis de hierarquia de memória, algumas delas compartilhadas, outras exclusivas de cada processador de fluxo (SM). Analisamos esses e outros detalhes a seguir.

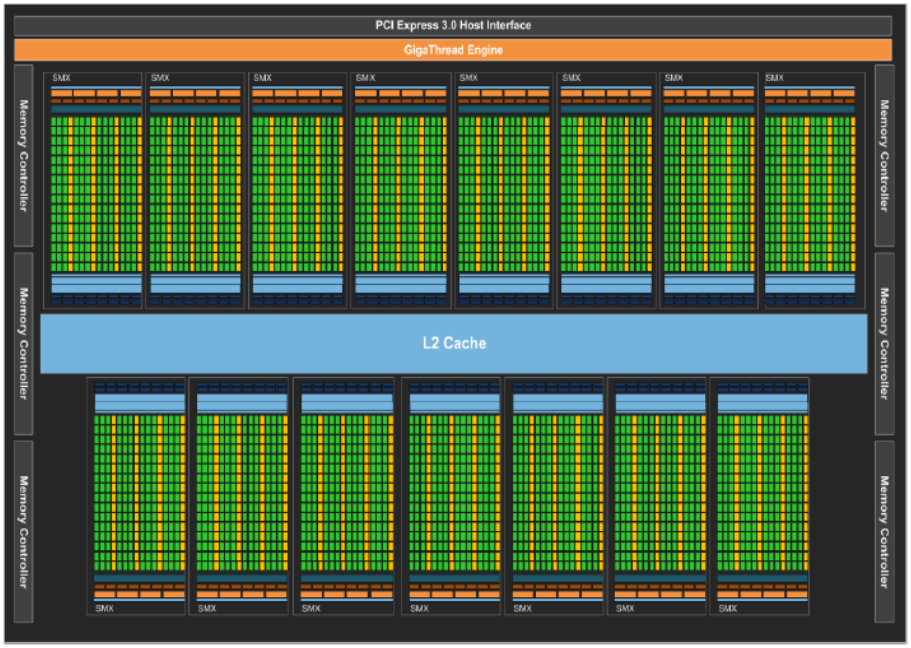

Figura 2.4: Arquitetura NVIDIA Kepler (NVIDIA, 2014)

O escalonador do multiprocessador de fluxo (SM) dispara as *threads* em grupos de 32 *threads* chamadas de *warps*. Cada SM possui quatro escalonadores de *warp*, permitindo um máximo de quatro *warps* disparadas e executadas concorrentemente. O número de registradores pode chegar até 255 registradores utilizados simultaneamente por cada *thread*.

Para melhorar ainda mais o desempenho, a arquitetura Kepler apresenta uma instrução de *shuffle* que permite às *threads* dentro de uma mesma *warp* compartilhar dados. Anteriormente, o compartilhamento de dados entre *threads* demandava o acesso à memória compartilhada, com operações de *load* e *store*, impactando em muito o desempenho de aplicações como a transformada de Fourier (FFT). Com essas características, a GPU da NVIDIA tem um grande destaque no uso de diversas aplicações de bioinformática.

Como exemplo, em 2010 o programa CUDAlign foi desenvolvido com o objetivo de alinhar longas sequências de DNA utilizando o algoritmo *Smith-Waterman* em uma arquitetura de GPU. Esse algoritmo é amplamente utilizado para o alinhamento local de sequências de DNA, isto é, para determinar regiões semelhantes entre a sequência-alvo e a referência (LIU et al., 2014).

Em 2014, a versão do programa foi atualizada (CUDAlign v.3.0), melhorando sua eficiência na execução do algoritmo, através da estratégia de paralela em plataformas com várias GPUs. Assim, o programa se tornou eficiente para o alinhamento de longas sequências de DNA (O. SANDES et al., 2014).

GPU AMD

Os aceleradores AMD *Radeon* foram projetados para fornecer altos níveis de desempenho para sistemas de aprendizagem, computação de alto desempenho, computação em nuvem e renderização. Esses aceleradores foram projetados com operações otimizadas de aprendizagem profunda, excepcional desempenho de precisão dupla e memória HBM2 hiper-rápida, oferecendo velocidades de largura de banda de memória de 1 TB/s (ZHU et al., 2018).

A memória de alta velocidade (HBM) é um tipo de memória DRAM empilhada que integra verticalmente várias matrizes de memória. As GPU da AMD possuem quatro pilhas de chips de memória entorno do chip de processamento, composto por 4 GB de memória global (Figura 2.5). Com a utilização de memória HBM não apenas aumenta a largura de banda da memória do dispositivo, mas também melhora a eficiência energética em comparação com a tecnologia GDDR5 tradicional (A.M.D., 2013).

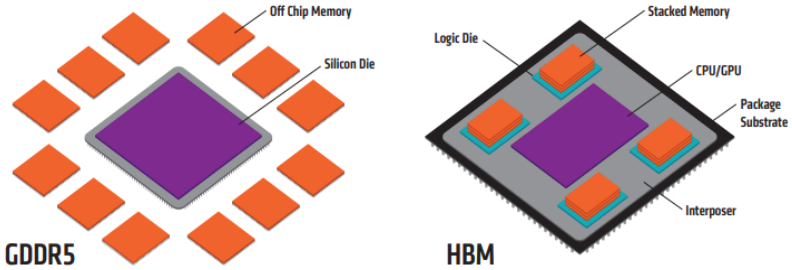

Figura 2.5: Comparação entre as arquiteturas de memória tradicional e memória HBM (A.M.D., 2013)

Para o uso desses aceleradores são necessários *softwares* específicos que garantam o máximo de seu desempenho. A AMD utiliza o *ROCm*, uma plataforma de software aberta que permite aos pesquisadores aproveitar o poder dos aceleradores AMD *Instinct*. A plataforma *ROCm* é construída baseada na portabilidade aberta, compatível com ambientes em várias arquiteturas de aceleradores (Figura 2.6). O *ROCm* incorpora *drivers* otimizados, compiladores, bibliotecas, compatibilidade para OpenMP e as ferramentas necessárias para grandes sistemas HPC (PEREZ, 2018).

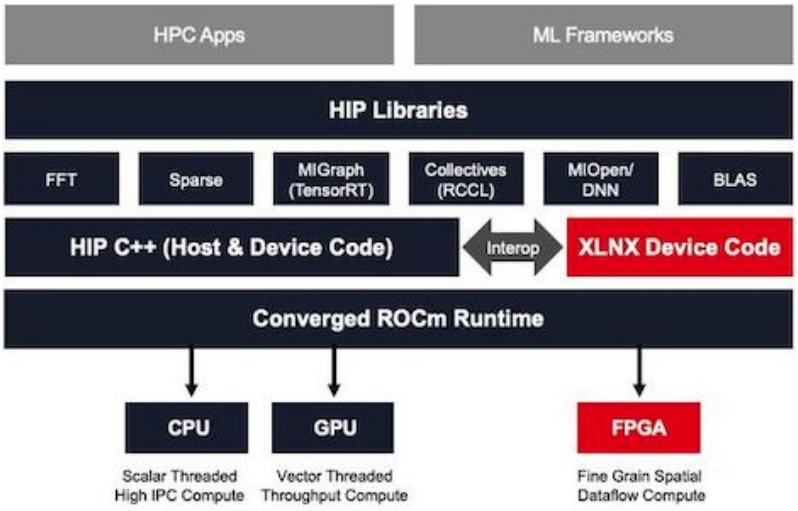

Figura 2.6: Bibliotecas e modelos de programação suportadas pelo *ROCm* (PEREZ, 2018)

INTEL MIC

Arquitetura do Intel MIC (do inglês, *Many Integrated Core*) é voltada para a computação de alto desempenho, ou seja, que utiliza grandes demandas de dados para processamento paralelo em uma variedade de áreas, tais como a Física Computacional, Química, Biologia e Finanças (RAHMAN, 2013). Neste caso, o co-processador é suportado por um ambiente de desenvolvimento que inclui diversos produtos, como compiladores, diversas bibliotecas, ferramentas de *tuning* e depuradores.

O *Knights Landing* (KNL) é o nome do modelo do processador Intel Xeon Phi de segunda geração que foi amplamente utilizado em implantações de soluções de HPC. A principal diferença em relação a geração anterior é o uso mais eficiente do recurso de memória (Figura 2.7).

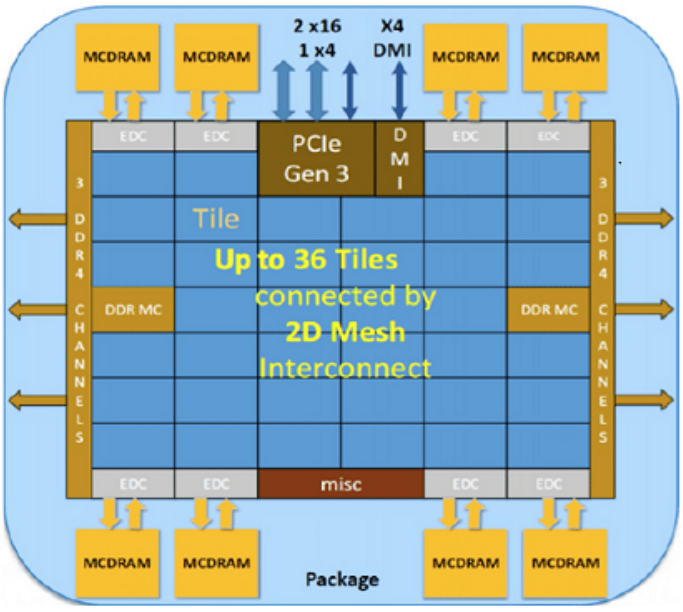

Figura 2.7: Representação geral da arquitetura do Intel Xeon Phi Knights Landing (KNL) (CODREANU; RODRíGUEZ; SAASTAD, 2017)

O sistema de memória Intel Xeon Phi KNL é composto por 16 GB acessados por 8 controladores de memória, bem como até 384 GB de DDR4 acessados por 2 controladores de memória de 3 canais. Um aspecto importante do desenvolvimento do Knights Landing, é a largura de banda de memória alcançada. É possível obter mais de 400 GB/s de largura de banda dos 16 GB de MCDRAM e mais de 90 GB/s de DRAM normal (Figura 2.8).

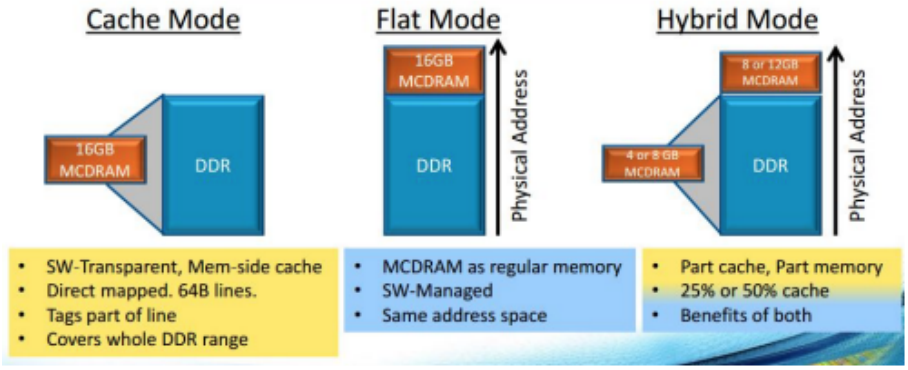

Figura 2.8: Representação geral da arquitetura do Intel Xeon Phi Knights Landing (KNL) (CODREANU; RODRíGUEZ; SAASTAD, 2017)

- **Modo Flat:** Neste modo, modificações de software são necessárias para usar o DDR e o MCDRAM na mesma aplicação. A memória MCDRAM é mapeada no mesmo espaço de endereço que a memória DDR e atua da mesma forma em termos de leitura e escrita. A vantagem é que os 16 GB de MCDRAM são vistos como endereçáveis, aumentando assim a memória endereçável total.

- **Modo Cache:** Usar este modo não requer nenhuma alteração de software e funciona bem para muitos aplicações. Nesse modo, o cache é gerenciado pelo hardware e as aplicações funcionam perfeitamente e se beneficiam da memória HBM.

- **Modo Híbrido:** Este modo usará parte da MCDRAM tanto para cache como memória comum. Esta configuração tem um bom desempenho para aplicações

que podem se beneficiar de um maior armazenamento em cache, bem como utilizar de forma mais eficiente a memória HBM.

Como exemplo, a arquitetura Intel *Xeon Phi* foi utilizada num estudo que avaliou o desempenho do programa *DALIGNER*, desenvolvido para alinhar longas sequências de DNA. O programa apresentou melhoria no desempenho em 14% ao ser executado em *Xeon Phi*, quando comparado a um processador *multicore* convencional (COSTA; SILVA; TEIXEIRA, 2019).

Em 2014, o desenvolvimento dos programas *SWAPHI* e *SWAPHI-LS*, utilizando coprocessadores Intel *Xeon Phi*, possibilitou a aceleração do algoritmo *Smith-Waterman* para o acesso e pesquisa em bancos de dados de proteínas e também no alinhamento de sequências de DNA longas, respectivamente (LIU et al., 2014).

2.3 Avaliação de Desempenho

Uma das questões que surgem ao se elaborar um programa paralelo é saber o quanto ele pode ser paralelizado e se apresenta um desempenho adequado quando executado em ambiente paralelo. Para avaliação do desempenho de um sistema de processamento paralelo, são consideradas, por sua maior importância, as métricas *speedup* (isto é, ganho de desempenho ou aceleração), eficiência e escalabilidade (SILVA, 2018).

São diversos fatores que influenciam essas métricas, tais como o custo de sincronização e comunicação, a distribuição das tarefas entre os processadores e o percentual do tempo de execução do programa passível de paralelização.

2.3.1 Speedup

O *speedup*, ou aceleração, mede a razão entre o tempo gasto para execução de um algoritmo ou aplicação em um único processador ($T(1)$) e o tempo gasto na execução com P processadores ($T(P)$), como visto na Equação 2.1:

$$S(P) = \frac{T(1)}{T(P)} \tag{2.1}$$

Em condições ideais, quando o *speedup* é sempre igual a P , onde P é o número de processadores em uso, temos o chamado *speedup* linear. Mas, na prática, o *speedup* é geralmente menor do que P, devido, principalmente, à sobrecarga de comunicação entre os diferentes fluxos de execução do programa, perdas por sincronização e decomposição de tarefas malfeitas. Quando isso acontece, chamamos o *speedup* de sublinear. Essa situação pode se deteriorar até o ponto em que a adição de mais processadores diminui o ganho obtido, caracterizando o que se chama de "retorno negativo ".

Em algumas situações especiais, *speedups* superiores a P podem ser obtidos (denominados *speedups* superlineares). Exemplos dessas situações são aplicações em que o volume de dados manipulados é grande o suficiente para exceder o tamanho da memória cache de um único processador. Nesse caso, ao dividir a aplicação entre P processadores, o volume de dados manipulado por cada processador é aproximadamente dividido por P, sendo esse volume agora pequeno o suficiente para que seja armazenado integralmente, ou com baixo nível de interferência destrutiva na memória *cache* de cada processador. Dentro desse quadro, e respeitadas as condições mencionadas anteriormente, é razoável se esperar um *speedup* superior a P no processo de paralelização, já que o desempenho com um único processador fica muito prejudicado pela baixa taxa de acerto nos acessos à memória *cache*.

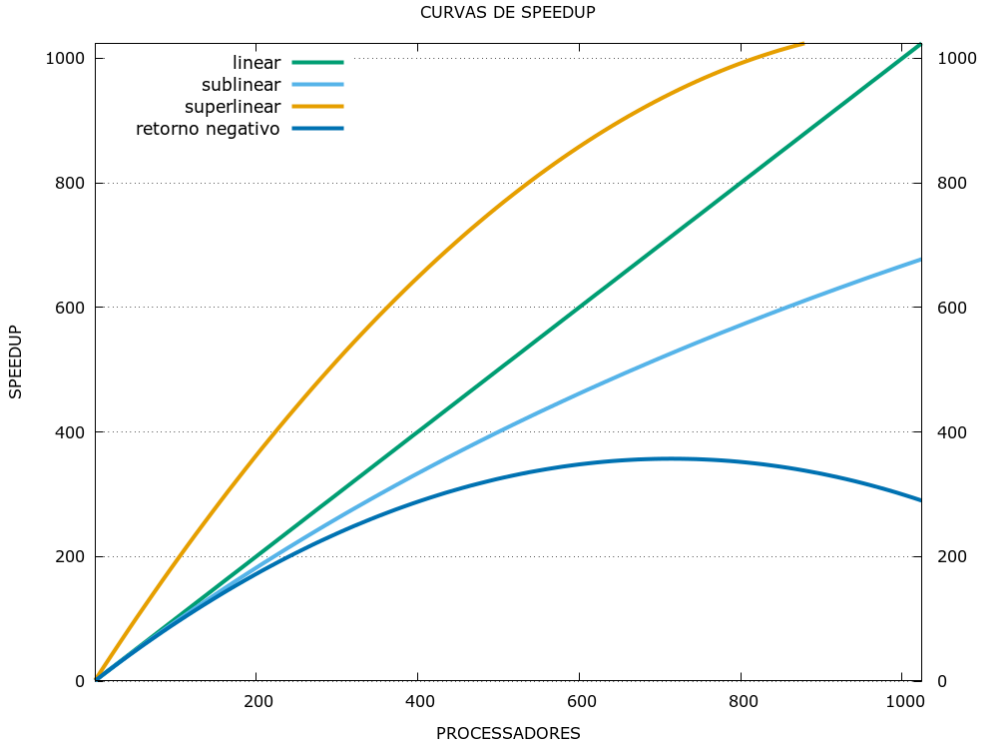

Figura 2.9: Curvas de *speedup* demonstrando situações de linearidade, quando o *speedup* é igual ao número de processadores em uso (P); sublinearidade, quando o *speedup* é menor que P ; supralinearidade, que ocorre quando o speedup é maior que P ; e o retorno negativo, quando o aumento de P não aprimora o *speedup*

.

Uma situação análoga ocorre quando são feitas buscas em grandes bases de dados, tais como a busca de dados genômicos realizada por diversas implementações paralelas do programa BLAST (ALTSCHUL et al., 1990). Neste caso, a quantidade de memória RAM acumulada de cada um dos nós em um *cluster* permite que a base de dados se mova do disco inteiramente para a memória, reduzindo, portanto, dramaticamente o tempo requerido para o mpiBLAST percorrer toda a base de dados, por exemplo (CORREA; SILVA, 2012). O *speedup* superlinear pode ocorrer também sob determinadas condições quando da execução de algoritmos de *backtracking* e *branch and bound* paralelos. A Figura 2.9 ilustra as diferentes curvas de *speedups*.

2.3.2 Eficiência

A eficiência é a medida de quão efetiva é a adição de novos processadores para ajudar na resolução de um problema. Seu valor é obtido pela razão entre o *speedup* ($S(P)$) e o número de processadores (P) utilizados para obter esse speedup, conforme podemos observar na equação 2.2:

$$E(P) = \frac{S(P)}{P} \tag{2.2}$$

Novamente, como o *speedup* em geral é menor do que P, por conta da sobrecarga do processamento paralelo, a eficiência tipicamente assume um valor menor do que 1. Entretanto, o ideal é que se obtenha o valor de eficiência próximo a 1. Para isto, é necessário atender às seguintes condições:

- no código a ser paralelizado, o percentual de código não paralelizável (que continuará sendo executado de forma sequencial) deve ser mínimo;
- no processo de paralelização, a distribuição de carga de trabalho entre os P processadores deve ser homogênea;
- independência dos processadores que trabalham nos trechos de código executados em paralelo, requerendo mínima comunicação ou sincronização entre eles.

2.3.3 Escalabilidade

Um sistema é dito escalável quando sua eficiência se mantém constante à medida que o número de processadores P aplicados à solução do problema aumenta.

Se o tamanho do problema é mantido constante e o número de processadores aumenta, o *overhead* de comunicação tende a crescer, e a eficiência a diminuir. Na prática, uma análise da escalabilidade deve considerar a possibilidade de se aumentar proporcionalmente o tamanho do problema a ser resolvido à medida que P cresce, contrabalanceando naturalmente o aumento do *overhead* de comunicação.

Considere como exemplo um problema de tamanho S. Usando P processadores, esse problema leva um tempo T para ser executado. O sistema é dito escalável se um problema de tamanho $2S$ executado em $2P$ processadores leva o mesmo tempo T . Desta forma, a escalabilidade é frequentemente uma propriedade mais desejável quando comparada ao *speedup*.

3 TÉCNICAS E MÉTODOS DE SEQUENCIAMENTO, ALINHAMENTO E MONTAGEM DE DNA

Este capítulo tem por objetivo descrever as principais técnicas e métodos utilizados no sequenciamento, alinhamento e montagem de DNA.

3.1 Sequenciamento

Grande parte das informações que constituem um ser vivo estão contidas no seu DNA, mas existem informações que não são codificadas no DNA (GRIFFITHS, 2005). São as informações epigenéticas. O sequenciamento é um processo que determina a ordem dos nucleotídeos na fita de DNA, permitindo acesso à informação estrutural, por exemplo, as posições e o número de cópias dos genes, polimorfismos de base única, deleções e inserções. Por consequência, essas informações possibilitam a identificação de regiões cromossômicas ou mutações relacionadas a determinadas características do espécime em questão.

Entretanto, o sequenciamento é um processo computacional complexo, pois vários genomas possuem regiões com sequências repetitivas não codificantes e de funções desconhecidas até então. Estas sequências apresentam unidades que variam entre 5 a 250 pares de nucleotídeos, que se repetem centenas de vezes. Na Figura 3.1 é visto o processo de evolução do sequenciamento de DNA ao longo dos últimos 10 anos.

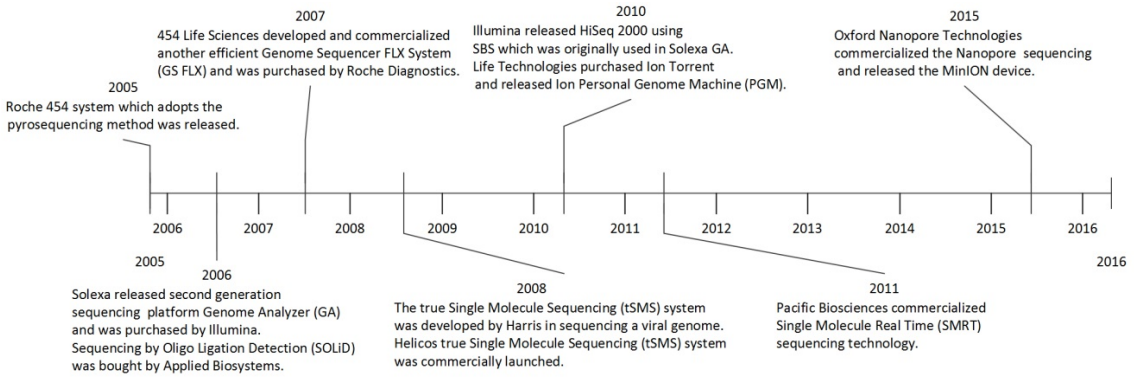

Figura 3.1: Evolução das técnicas de sequenciamento de DNA nos últimos 10 anos (WONG et al., 2019)

3.1.1 Método de Sanger

Os nucleotídeos que compõem a molécula de DNA, os desoxirribonucleotídeos, são formados por uma base nitrogenada (Adenina, Citosina, Timina e Guanina), pelo açúcar desoxirribose e por 3 grupos fosfato. O método tradicional de sequenciamento de DNA, proposto por Frederick Sanger e colaboradores na década de 1970, consiste na adição de nucleotídeos quimicamente modificados, os dideoxinucleotídeos (ddNTPs), que apresentam um grupo hidroxila no lugar do grupo fosfato no Carbono 3' da desoxirribose. Essa alteração impede a continuidade do processo de extensão da molécula pela DNA polimerase permitindo, portanto, a leitura do fragmento sequenciado (SANGER; COULSON, 1975). Esse método é ainda utilizado em projetos que não necessitem da robustez dos processos de sequenciamento mais recentemente desenvolvidos (Figura 3.2), entretanto, com algumas modificações descritas na próxima seção.

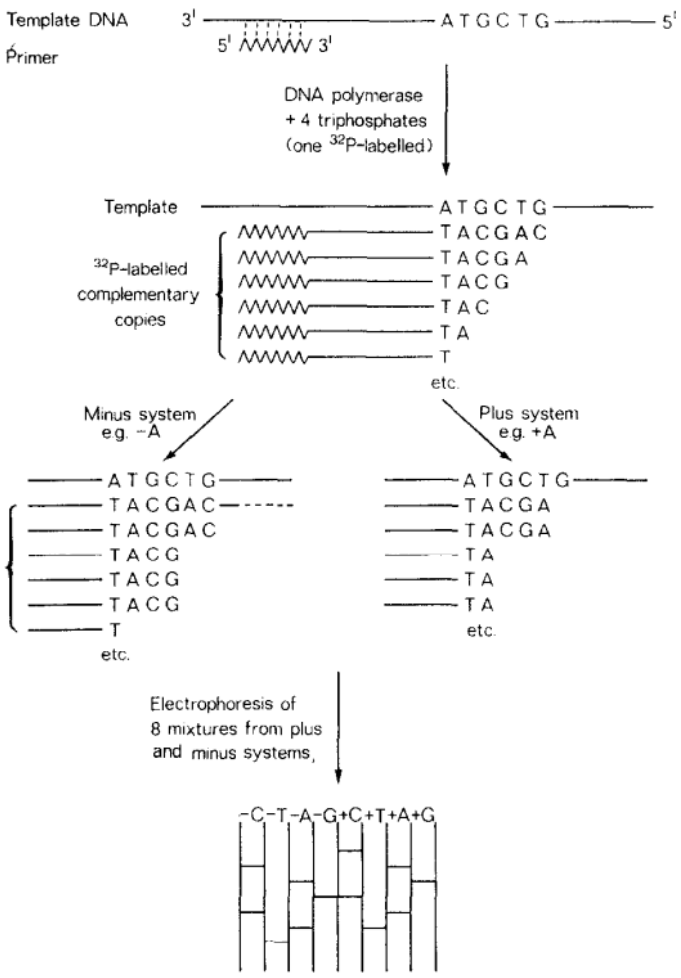

Figura 3.2: O princípio do método de sequenciamento de DNA desenvolvido por Sanger e colaboradores (SANGER; COULSON, 1975)

3.1.2 Sequenciamento por Capilaridade (automatização do método de Sanger)

Assim como o método de Sanger, esse processo utiliza o princípio de sequenciamento do DNA através da adição de dideoxinucleotídeos. Os pontos que diferem da técnica desenvolvida por Sanger e colaboradores é a marcação dos dideoxinucleo-

tídeos com fluoróforos (diferente para cada base), permitindo a realização da reação em um único ensaio. Após a reação, a amostra percorre um capilar contendo gel, e ao final a fluorescência é detectada e filtrada por um fotomultiplicador, sendo convertida na leitura das bases que compõem o fragmento de DNA sequenciado (Figura 3.3).

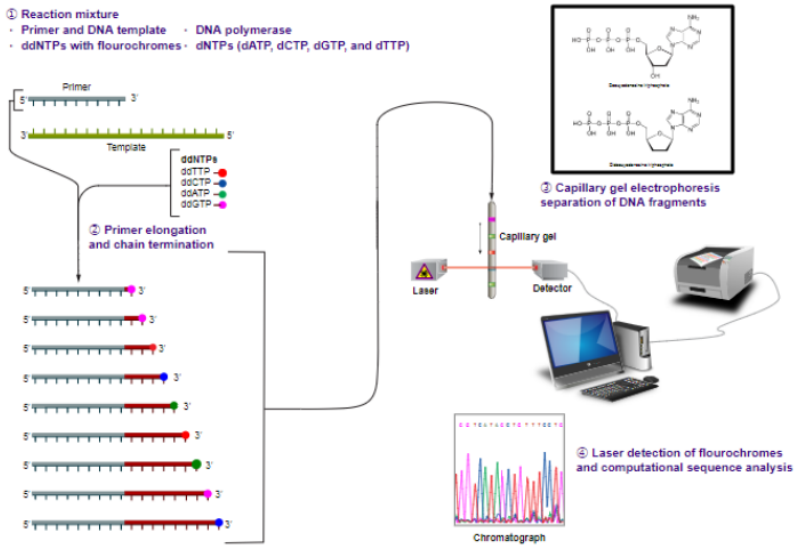

Figura 3.3: Esquematização do método de sequenciamento por capilaridade (OLIVEIRA et al., 2014)

3.1.3 Pirosequenciamento

O método de Pirosequenciamento (454 Roche) consiste em uma nova abordagem molecular do sequenciamento, em que o DNA extraído é fragmentado, com cada uma das extremidades ligadas a adaptadores que atuarão na adesão às nanoesferas para a continuidade do processo. Nessa técnica, a amplificação dos fragmentos para a formação dos *clusters* ocorre através da reação em cadeia da polimerase (PCR) em emulsão. Em seguida, há o processo de sequenciamento em que a cada nucleotídeo adicionado é liberado pirofosfato (PPi), em quantidade diferente de acordo com cada

tipo de nucleotídeo, e que será convertido em adenosina trifosfato (ATP) pelas enzimas ATP sulfurylase e adenosina 5' fosfossulfato. O produto dessa reação (ATP) será utilizado na reação de conversão da luciferina em oxiluciferina, mediada pela enzima luciferase, gerando luz (RONAGHI, 2001). Essa luz é captada por sensores e convertida em sinais ilustrados graficamente como picos. Como a intensidade da luz gerada é de acordo com a quantidade de ATP liberado, é possível distinguir quais nucleotídeos foram incorporados a cada ciclo (Figura 3.4).

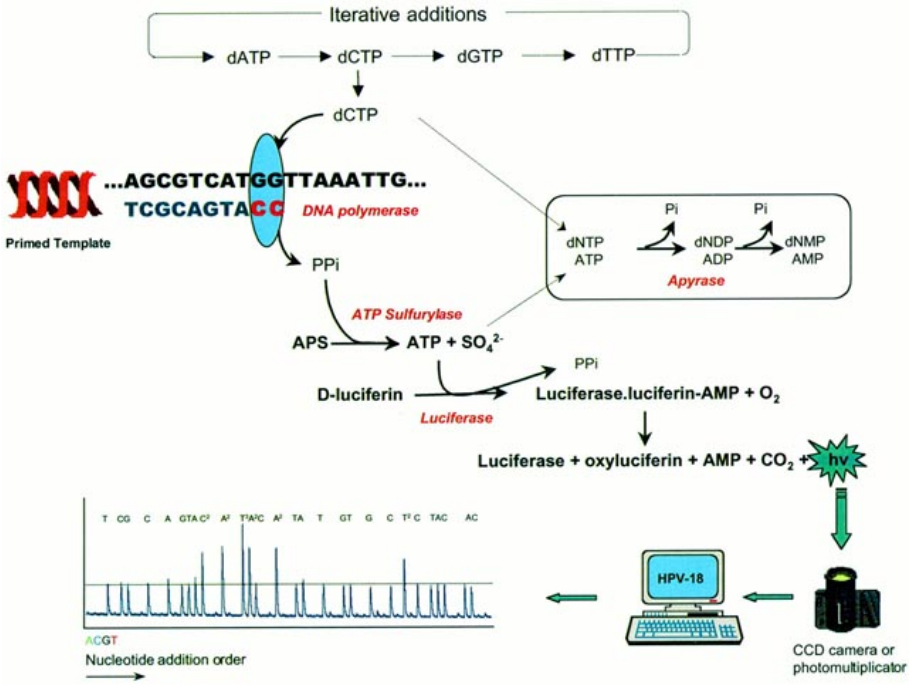

Figura 3.4: Esquematização do método de pirosequenciamento de DNA (GHARIZADEH et al., 2001)

3.1.4 Sequenciamento por Síntese

O método de Sequenciamento por Síntese utiliza os princípios de sequenciamento desenvolvidos por Sanger e colaboradores e pelo sistema de sequenciamento por capilaridade, ou seja, utiliza nucleotídeos modificados e que liberam fluorescência

após a incorporação à fita de DNA molde.

As plataformas mais utilizadas nesse método foram desenvolvidas pela Illumina Inc. Brevemente, o DNA é extraído, fragmentado, e cada fragmento é ligado a adaptadores que atuarão na ligação a uma lâmina (*flowcell*). Em seguida, ocorre a amplificação por ponte dos fragmentos, formando os clusters, para então iniciar o sequenciamento. Nessa etapa, a cada ciclo, e caso ocorra complementaridade, o nucleotídeo adicionado à fita de DNA molde e libera fluorescência (diferente para cada tipo de nucleotídeo), que será captada e transformada, gerando o arquivo contendo a sequência exata de cada um dos fragmentos processados representados por um cluster (Figura 3.5).

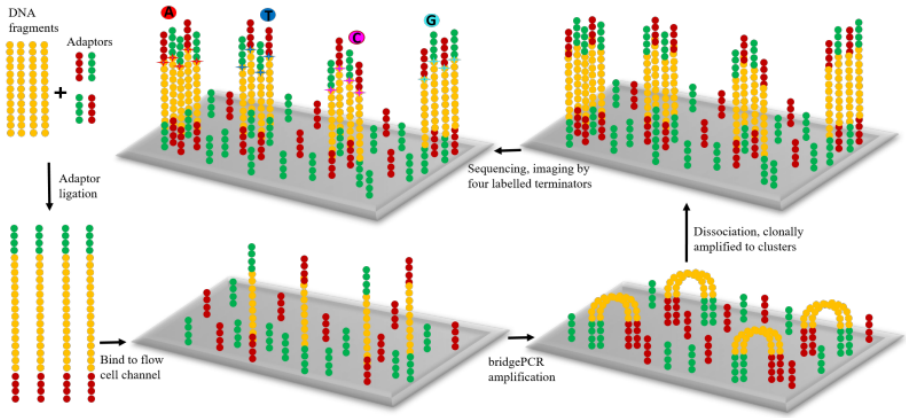

Figura 3.5: O princípio geral do método de Sequenciamento por Síntese (WONG et al., 2019)

3.1.5 Ion Torrent

A plataforma de Sequenciamento Ion Torrent foi desenvolvida pela *Life Technologies*. A identificação das bases se dá unicamente por diferencial de pH, e não por ddNTPs ou reações luminosas, como era o caso das tecnologias vistas anteriormente. Nessa tecnologia, a amostra de interesse é colocada em um pequeno chip, que

contém em sua superfície nanoporos providos de medidores de pH, em nanoescala. Após ser colocado no chip, o DNA sofre uma reação irreversível de ligação a esses nanoporos (MERRIMAN et al., 2012).

Com o DNA ligado, é iniciado o processo de sequenciamento. Do mesmo modo que ocorre nos métodos descritos anteriormente, no Ion Torrent as bases serão adicionadas uma de cada vez, e um ciclo será repetido milhares de vezes. Caso a base correta seja adicionada, a enzima DNA polimerase agirá e um íon H+ será liberado, provocando uma diminuição do pH no nanoporo. Essa alteração no pH será detectado e, dessa forma, a sequência presente em cada nanoporo será determinada (Figura 3.6).

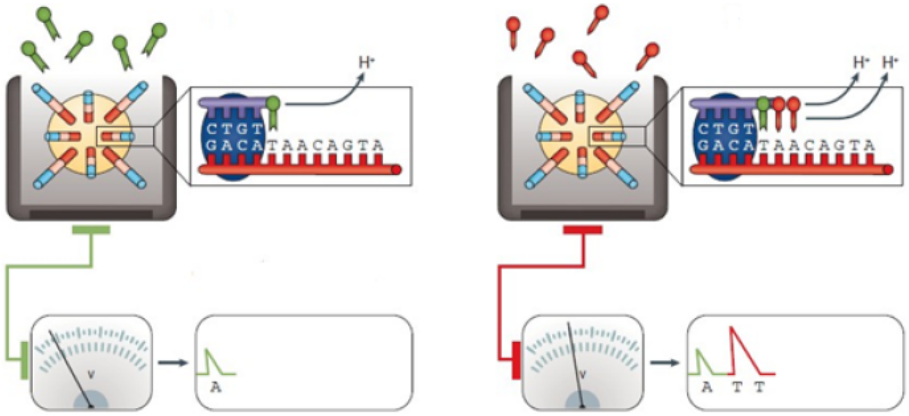

Figura 3.6: O princípio geral do método de Sequenciamento Ion Torrent (KCHOUK; GIBRAT; ELLOUMI, 2017)

3.1.6 PacBio

PacBio é um sistema baseado em uma nova tecnologia de sequenciamento de molécula única em tempo real (SMRT). Essa tecnologia possibilita a observação em tempo real da síntese de DNA através da enzima DNA polimerase, garantindo a esse método uma precisão, independentemente do conteúdo de Citosina e Guanina (CG).

Esse método apresenta uma faixa de leitura superior a qualquer outra tecnologia de sequenciamento, e todo o processo é feito de forma rápida e simples (RHOADS; AU, 2015).

Para o procedimento de preparo das amostras, a biblioteca do DNA molde é composta por fragmentos de DNA dupla fita conectados a adaptadores em suas terminações. Esses fragmentos com adaptadores são chamados de SMRTbells, capazes de transformar os fragmentos de DNA de fita dupla em moldes circulares, nos quais a enzima polimerase vai continuar a funcionar até que se inative ou até que ocorra o fim do período de observação. É realizada uma corrida com múltiplas passagens em torno desse molde circular, permitindo a condensação numa sequência consenso de maior precisão (Figura 3.7).

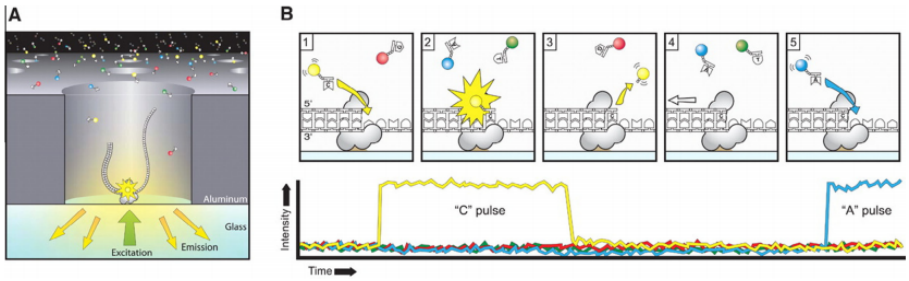

Figura 3.7: O princípio geral do método de Sequenciamento PacBio (RHOADS; AU, 2015)

3.1.7 Oxford Nanopore

A plataforma de sequenciamento Oxford Nanopore utiliza o sistema de detecção através de nanoporos proteicos imobilizados em membrana. A detecção dos nucleotídeos é feita através da análise da variação do potencial elétrico na membrana, que é alterado de acordo com a passagem de cada nucleotídeo da cadeia de DNA. Isto é possível devido ao fato de que cada nucleotídeo resultará em distinta

variação do potencial elétrico (KCHOUK; GIBRAT; ELLOUMI, 2017).

Em alguns casos, o sequenciador não consegue detectar a sequência do grupo, o que acarreta o aparecimento de longos gaps nos fragmentos. O *throughput* ainda é baixo, apesar de os fragmentos serem muito grandes. O tamanho do arquivo final é bastante extenso e com pouca disponibilidade de programas e protocolos para análise dos dados (Figura 3.8).

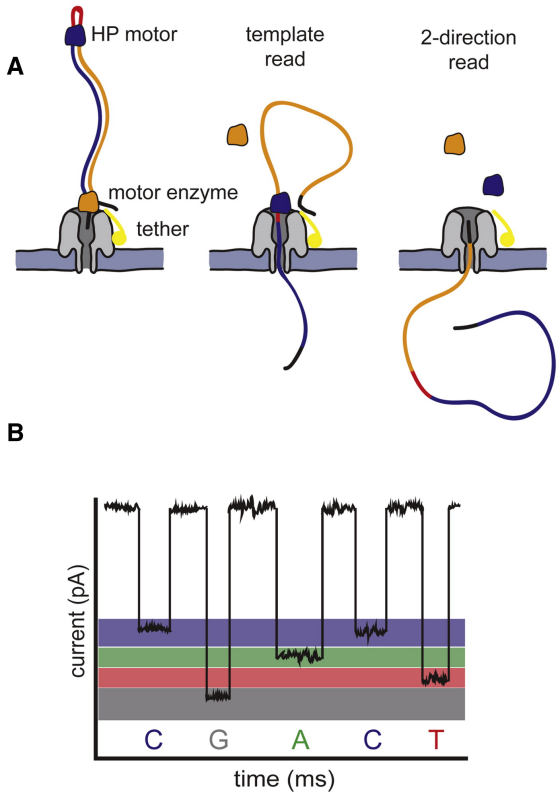

Figura 3.8: O princípio geral do método de Sequenciamento Oxford Nanopore (KCHOUK; GIBRAT; ELLOUMI, 2017)

3.2 Tipos de Arquivos Gerados no Sequenciamento

Durante o processo de sequenciamento do DNA, são gerados arquivos que podem conter sequências curtas (*short reads*), cujo comprimento varia entre 50 a 150 bases, ou longas (*long reads*), cujo comprimento ultrapassa 1000 bases, dependendo da técnica utilizada. Os arquivos gerados possuem formatos diferentes e são utilizados de acordo com o programa de montagem. Os principais formatos de arquivos são *fastq*, *fasta*.

- **Arquivo *fastq***

O *fastq* é um formato baseado em texto para armazenar uma sequência de nucleotídeos e seus índices de qualidade correspondentes (COCK et al., 2009). Esse formato foi desenvolvido pelo *Wellcome Trust Sanger Institute*, com o objetivo de melhorar a qualidade do arquivo Fasta. Atualmente, tornou-se padrão para o armazenamento de informações de sequenciamento de DNA, sendo este o formato liberado pelas diferentes plataformas. A seguir, um exemplo do arquivo *fastq*.

O arquivo *fastq* contém informações referentes a cada um dos fragmentos de DNA sequenciados, as quais são especificadas em quatro linhas, conforme descrição seguinte.

```
@HWI-ST1054:100:d0abmacxx:7:1208:13194:181739 1:N:0:CGATGT
AAAGTGCCATAGAGTTGTACAAGCAGGAGAGATACATGTGGATANNNNC
+
B?BDBDEFHGHHHJGIGIJJIIJJIIGIEGHHJJJGGIIE*00?FHGJJHDFEFFDE@
```

Linha 1 = "@"+ identificador de sequência
Linha 2 = sequência

Linha 3 = "+" é opcionalmente seguido pelo mesmo identificador de sequência
Linha 4 = qualidade em Ascii, que codifica para o escore de qualidade Phred

- **Arquivo *fasta***

O formato *fasta* é baseado em texto para representar uma sequência de nucleotídeos, utilizando-se uma sequência de letras que os representam: Adenina = A, Citosina = C, Guanina = G e Timina = T. Os arquivos no formato *fasta* são mais fáceis de manipular e de analisar as sequências contidas neles, usando-se ferramentas simples de processamento de texto e linguagens de *script* (PEARSON, 2004). As linguagens de programação mais utilizadas para esse processo são Python, Ruby e Perl. O arquivo *fasta* contém informações referentes a cada um dos fragmentos de DNA sequenciados, as quais são especificadas em duas linhas, conforme descrição seguinte.

```
>HWI-ST1054:100:d0abmacxx:7:1208:13194:181739 1:N:0:CGATGT
AAAGTGCCATAGAGTTGTACAAGCAGGAGAGATACATGTGGATANNNNC
```

Linha 1 = ">" + identificador de sequência
Linha 2 = sequência

3.3 Alinhamento

O alinhamento de sequências é um processo para organizar e comparar sequências de DNA com um genoma/sequência de referência. Os alinhamentos podem ser

feitos entre duas sequências, denominadas alinhamento simples, ou entre três ou mais sequências, assim denominadas alinhamento múltiplo. Ademais, os alinhamentos podem ser realizados em duas abordagens diferentes: alinhamento global ou alinhamento local (COSTA; SILVA; TEIXEIRA, 2020).

3.3.1 Alinhamento Global

Alinhamento global é um processo no qual as sequências de DNA-alvo são alinhadas à sequência de referência por cada resíduo (nucleotídeo ou aminoácido). Em outras palavras, leva em consideração todas as sequências e toda a sua extensão, realizando um alinhamento de ponta a ponta. Quanto maior a dissimilaridade entre alvo e referência, maior número de gaps será observado. O principal algoritmo para essa abordagem é o *Needleman-Wunsch*, baseado em programação dinâmica, desenvolvida por Saul Needleman e Christian Wunsch em 1970 (NEEDLEMAN; WUNSCH, 1970).

O algoritmo analisa todos os possíveis segmentos existentes dentro das duas sequências por meio de uma matriz de similaridade. Cada linha da matriz é associada a um caractere da sequência A e cada coluna a um caractere da sequência B, de forma que a quantidade de elementos da matriz equivale ao produto dos tamanhos das sequências comparadas.

Para encontrar o alinhamento com o maior escore, uma matriz F é alocada. Há uma coluna para cada caractere da sequência A e uma linha para cada caractere da sequência B. À medida que o algoritmo avança, a matriz $F_{(i,j)}$ é preenchida com o escore ótimo do alinhamento entre os i primeiros caracteres da sequência A e os j primeiros da sequência B. Os valores encontrados em $S_{(Xi,Yj)}$ medem a similaridade entre os caracteres das sequências utilizadas, e d é a pontuação do espaçamento

(Equação 3.1).

$$F_{(i,j)} = max \left\{ \begin{array}{l} F_{(i-1,j-1)} + S_{(Xi,Yj)} \\ F_{(i,j-1)} - d \\ F_{(i-1,j)} - d \\ d = 0 \end{array} \right\} \tag{3.1}$$

Na Figura 3.9, é visto o processo de alinhamento global. As sequências A e B são comparadas na totalidade do seu comprimento, sendo as diferenças de comprimento da sequência compensadas com *"gaps"*.

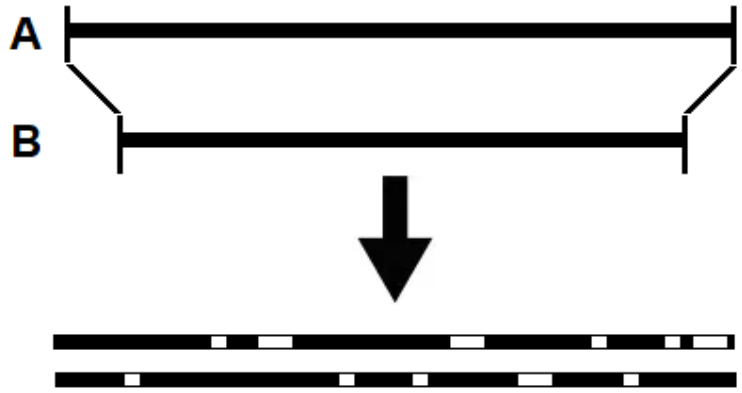

Figura 3.9: Processo de Alinhamento Global entre sequências. O alinhamento é feito em todo o comprimento da sequência (LEMOS; CASANOVA, 2000)

3.3.2 Alinhamento Local

Diferentemente do alinhamento global, os alinhamentos locais consideram subseções das sequências para corresponder à referência, isto é, buscam somente alinhamento de regiões de alta similaridade, não importando as sequências adjacentes a essas regiões. Isso é útil para encontrar regiões e motivos com semelhança entre as sequências-alvo e de referência. No alinhamento local é utilizado o algoritmo de *Smith-Waterman*. Esse é um algoritmo baseado em programação dinâmica, mas

com opções diferentes para iniciar e terminar em qualquer posição (SMITH; WATERMAN, 1981).

O algoritmo é dividido em duas etapas: na primeira etapa ele recebe como sequências de entrada S_0 e S_1, com tamanhos $|S_0| = m$ e $|S_1| = n$. A matriz $H_{(m+1,n+1)}$, onde $H_{(i,j)}$ contém a pontuação entre os prefixos $S_0[1..i]$ e $S_1[1..j]$. No início, a primeira linha e coluna são preenchidas com zeros. Os demais elementos de H são obtidos através da Equação 3.2.

$$h_{(i,j)} = max \left\{ \begin{array}{l} H_{(i-1,j-1)} + (S_0[i] + S_1[j]) \\ H_{(i,j-1)} + g \\ H_{(i-1,j)} + g \\ 0 \end{array} \right\} \tag{3.2}$$

Na etapa seguinte, é obtido o alinhamento local ideal, usando as saídas da primeira etapa. O cálculo começa na célula que possui o valor mais alto em H, seguindo o caminho que produziu a pontuação ideal até o valor zero ser atingido.

Na Figura 3.10 é visto o processo de alinhamento local utilizando as sequências A e B. O processo consiste na identificação de regiões isoladas de elevada similaridade entre as duas sequências, independentemente do seu contexto.

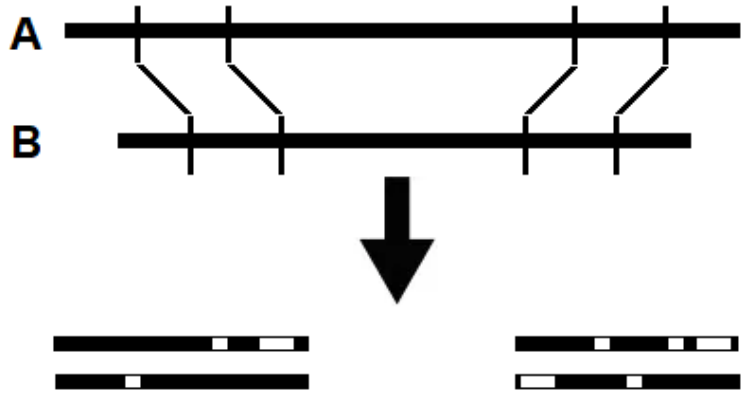

Figura 3.10: Processo de Alinhamento Local entre sequências. O alinhamento é feito em partes do comprimento da sequência (LEMOS; CASANOVA, 2000)

3.4 Montagem de DNA

O primeiro passo para a montagem de um genoma é o alinhamento das sequências (caso o montador use alinhamento), descrito nos tópicos anteriores, e cujo objetivo é alcançar uma montagem que represente o cromossomo original do organismo em estudo (MILLER; KOREN; SUTTON, 2010).

A montagem do genoma é feito através da união de um grande número de sequências de DNA, resultando na representação do cromossomo original do organismo em estudo em um cenário ideal. Em um projeto de sequenciamento *shotgun*, todo o DNA do organismo analisado é inicialmente particionado em fragmentos menores, os quais serão "lidos" por máquinas de sequenciamento automático, capazes de ler até 1000 nucleotídeos ou bases de uma só vez, dependendo da tecnologia utilizada (Figura 3.11).

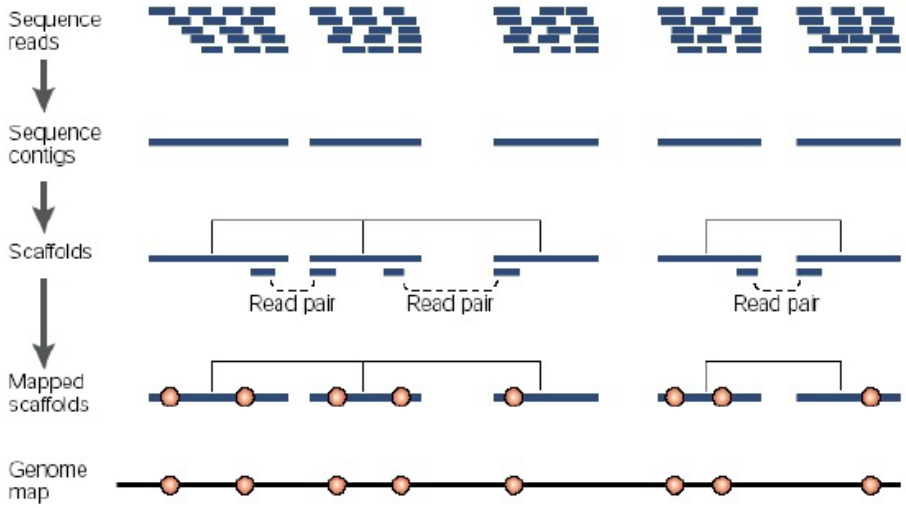

Figura 3.11: Processo de montagem de genoma (TASMA et al., 2015)

Um algoritmo de montagem de genoma é então utilizado para reunir todas as partes, podendo colocá-las na ordem original, detectando todos os locais onde existem coincidências entre fragmentos distintos de DNA. As partes coincidentes podem ser fundidas, unindo dois fragmentos de DNA. O processo é repetido até que a sequência esteja completa, ou seja, o mais próximo possível do cromossomo original. Existem dois tipos de processo de montagem: por referência e a *de novo*.

3.4.1 Montagem por Referência

No processo, a sequência que será montada é comparada a um genoma já existente e com montagem suficiente para sua utilização como referência, isto é, pode-se usá-lo como referência na tomada de decisão ao encontrar ambiguidades e erros de sequenciamento e também como um mapeamento. Nesse processo a utilização de recursos computacionais é baixa (Figura 3.12).

Figura 3.12: Esquema representando a montagem de genoma utilizando o processo baseado no alinhamento das sequências-alvo a um genoma de referência (PARKS; LISTON; CRONN, 2010)

3.4.2 Montagem *de novo*

Montagem *de novo* significa "desde o princípio", isto é, executar o processo de montagem de uma sequência de DNA quando não se tem um genoma conhecido ou que possa ser usado como referência durante o processo de montagem ou quando não se quer usar a referência. Nesse caso, a utilização de recurso computacional é muito alto, sendo necessário o uso de grande quantidade de processamento e memória (Figura 3.13).

Figura 3.13: Processo de montagem *de novo* (PARKS; LISTON; CRONN, 2010)

4 FRAMEWORK MELC GENOMICS 2.0

Neste capítulo serão revisados alguns dos *frameworks* mais utilizados para a montagem de genomas em bioinformática. Suas características foram comparadas com aquelas desenvolvidas para o *framework* MELC Genomics versão 2.0, apresentado como proposta desta tese, que tem o objetivo de apresentar interfaces intuitivas e amigáveis para os usuários, além da utilização de um montador paralelo de alto desepenho com uso do paradigma de programação OpenACC.

4.1 Frameworks

Os *frameworks* têm como principal objetivo a execução de um maior número de tarefas relacionadas ao tratamento de dados sequenciados, como análise de qualidade, alinhamento e montagem. Têm como características o uso de interface gráfica ou acesso por navegador web, com isso os usuários não necessitam usar a linha de comando para a execução de uma determinada tarefa, além de serem intuitivos e amigáveis. Nesta tese foram avaliados os *frameworks* Galaxy, BioExtract Server e MELC Genomics.

4.1.1 Galaxy

O Galaxy é um sistema de fluxo de trabalho científico e uma plataforma de integração de dados para dados biológicos, fornecendo uma interface gráfica com o usuário para especificar em quais dados operar, quais etapas executar e em que ordem as executar (GIARDINE et al., 2005).

Para o seu funcionamento o Galaxy se baseá em três pontos: o primeiro são os dados, que podem ser enviados ou utilizar os dados já existentes no banco de dados do Galaxy. O segundo ponto é a análise que é feita através das ferramentas disponíveis no *framework*. Por último a execução completa de todo o fluxo de trabalho necessário como por exemplo, a execução de uma montagem.

Todas as operações podem ser realizadas usando uma interface simples. Na Figura 4.1 podemos visualizar a interface inicial do Galaxy.

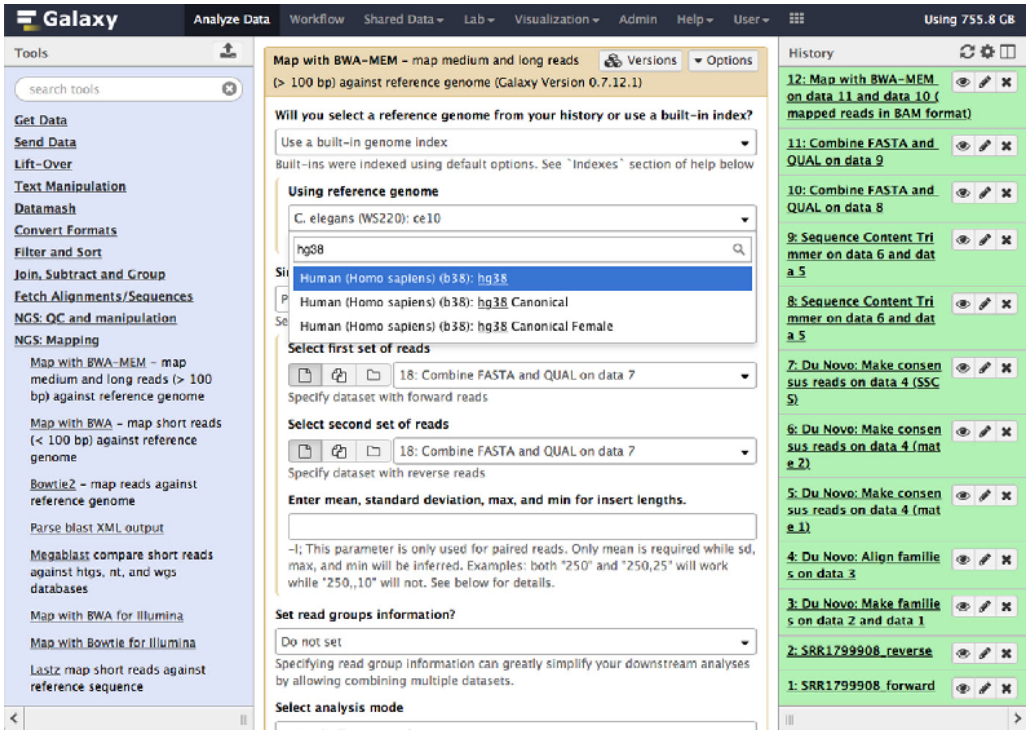

Figura 4.1: Interface de análise do Galaxy, que consiste em menu de ferramentas (painel esquerdo), interface de ferramentas (painel central), histórico (painel direito) (GIARDINE et al., 2005)

Com o Galaxy os usuários podem realizar consultas independentes sobre dados genômicos de diferentes fontes, combiná-los ou refiná-los, realizar cálculos, ex-

trair e visualizar sequências ou alinhamentos correspondentes e montagem de sequências.

Ele suporta *uploads* de dados do computador do usuário e diretamente de muitos recursos on-line, como o USCS Genome Browser, BioMart e InterMine. Originalmente, o Galaxy foi desenvolvido para análise genômica, mas atualmente também é usado para expressão gênica, proteômica, transcriptômica, e outras áreas.

Com o Galaxy é possível o usuário realizar uma analisa de qualidade dos dados submetidos antes de executar o processo de montagem do genoma. Para a verificação da qualidade o Galaxy usa o programa QUAST (GUREVICH et al., 2013).

Como dito anteriormente com o Galaxy é possível faz a montagem de genomas e transcriptomas. Entre os programas que compõem o *framework* utilizados para a montagem de genomas está o montador Velvet (ZERBINO; BIRNEY, 2008). No Galaxy os executáveis velveth e velvetg são executados de forma separada, com isso o usuário pode definir qual executar.

Existem duas formas de usar o Galaxy. A primeira opção é fazer uma instalação do Galaxy em um servidor local para uso em uma rede privada, desta forma é possível configurar os recursos computacionais que serão utilizados de acordo com os tipos de dados. A outra opção é usá-lo de forma remota, acessando diretamente o site do Galaxy.

4.1.2 BioExtract Server

O BioExtract Server é um sistema aberto, baseado na Web, em que os usuários podem construir seus próprios *pipelines* (LUSHBOUGH et al., 2010). Os usuários podem consultar dados de sequência on-line, analisá-los usando uma variedade de ferramentas de informática, criar e compartilhar fluxos de trabalho personalizados para análises repetidas e salvar os dados e fluxos de trabalho resultantes em relatórios padronizados.

O acesso ao BioExtract Server é feito através de um navegador web, conforme apresentado na Figura 4.2.

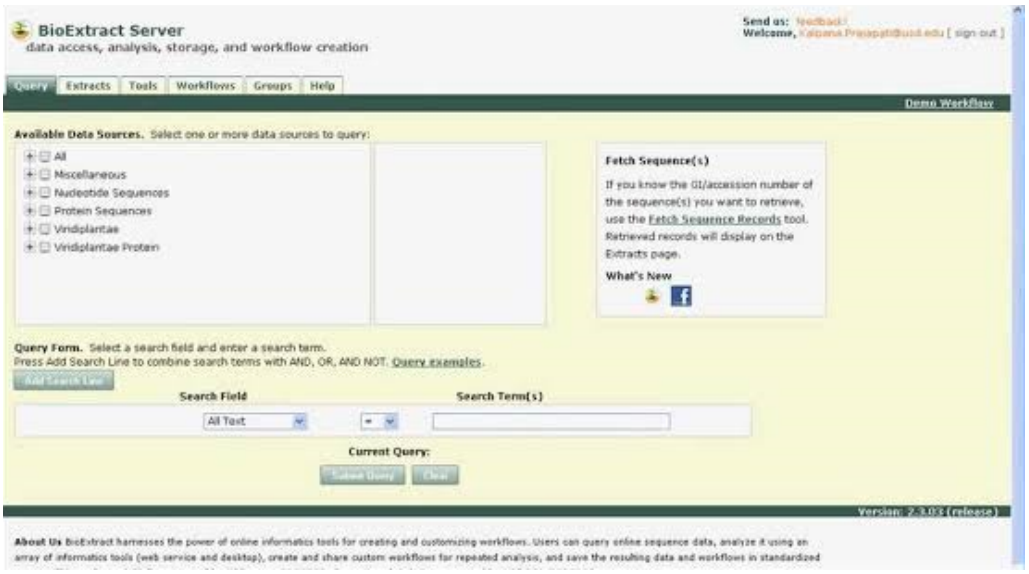

Figura 4.2: Interface principal de navegação BioExtract (LUSHBOUGH et al., 2010)

Além disso, esse servidor inclui interfaces de dados integrados, como os bancos de dados de nucleotídeos e proteínas do National Center for Biotechnology Information (NCBI), o banco de dados de nucleotídeos não redundantes do Laboratório Europeu de Biologia Molecular (EMBL-Bank), o Universal Protein Resource (Uni-

Prot) e o UniProt Reference. Banco de dados de Clusters (UniRef).

Depois que uma tarefa de um fluxo de trabalho for executada, as informações dessas tarefas específicas podem ser acessadas para verificação do gráfico de fluxo de trabalho. Além disso, um relatório de fluxo de trabalho de uma determinada tarefa pode ser gerado com informações detalhadas. Esses relatórios de podem ser baixados como tipo de arquivo PDF (Figura 4.4).

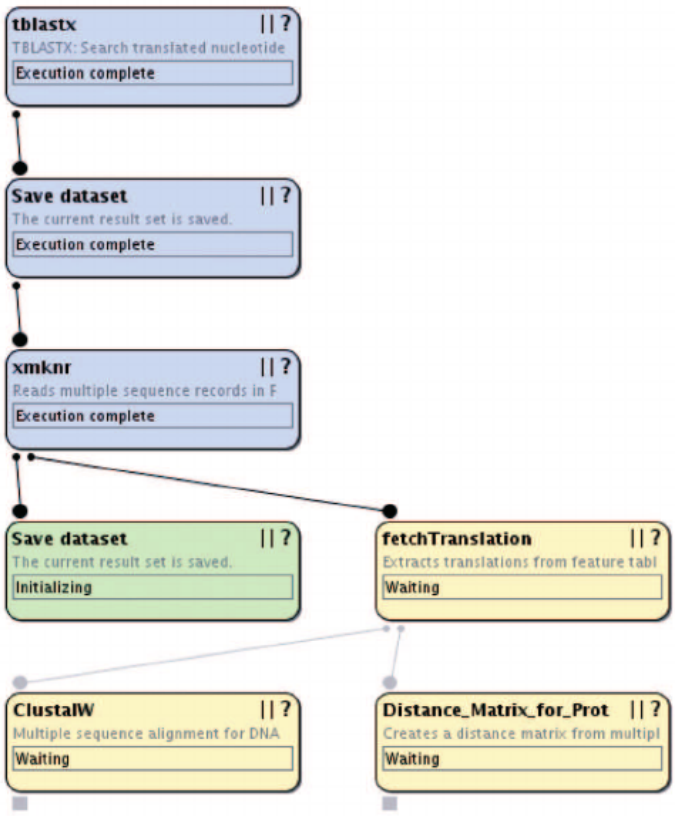

Figura 4.3: Execução dos fluxos de trabalho com o BioExtract Server. Os estados do nó mudam à medida que cada etapa é executada. Aqui é mostrado o fluxo de trabalho para a tarefa de criar um alinhamento de sequência múltipla (LUSHBOUGH et al., 2010)

Os fluxos de trabalho do BioExtract Server podem ser compartilhados. Os

usuários podem exportar um fluxo de trabalho como um arquivo XML e compartilhá-lo com outros colaboradores. Quando um arquivo XML de fluxo de trabalho é importado no BioExtract Server por um usuário, uma nova instância desse fluxo de trabalho é criada. Um segundo método de compartilhamento de fluxo de trabalho é através do myExperiment. O myExperiment é um ambiente de pesquisa virtual projetado para fornecer um mecanismo de colaboração e compartilhamento fluxos de trabalho.

Fluxos de trabalho do BioExtract Server importados para myExperiment pode ser anotado adicionalmente através seu sistema e executados diretamente por meio de sua plataforma. Um outro processo de compartilhamento é a extração de dados e análises, os fluxos de trabalho podem ser compartilhados com grupos colaborativos através do próprio BioExtract Server.

4.1.3 MELC Genomics 1.0

Em 2018 a primeira versão do MELC Genomics (COSTA, 2018) foi desenvolvida integrando alguns dos programas mais usados para um *pipeline* de montagem completo. Com uma interface web simples, intuitiva e amigável, o MELC Genomics podia ser executado em qualquer navegador, além de ser simples e rápido (Figura 4.4).

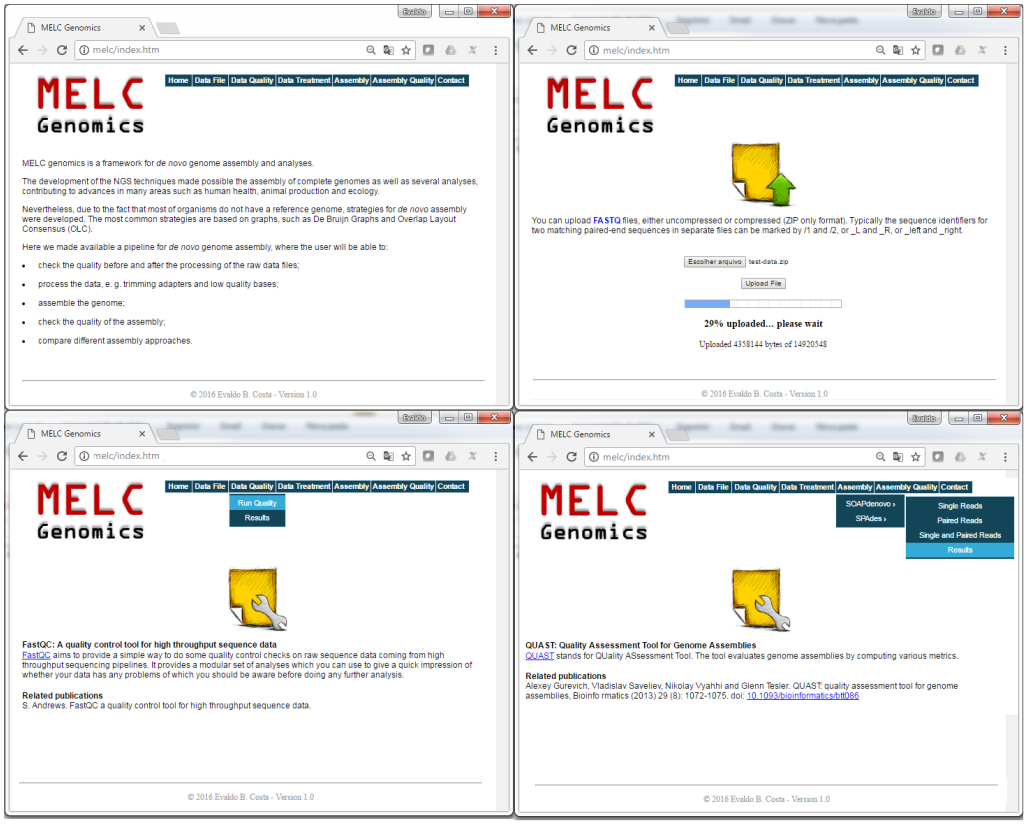

Figura 4.4: Interface de navegação do MELC Genomics 1.0. A barra de navegação fornece links para os principais componentes (COSTA, 2018)

O MELC Genomics 1.0 permitia que os usuários realizassem a verificação da qualidade antes e após o processamento dos arquivos de dados sequenciados brutos; processamento de dados, como o corte de adaptadores e bases de baixa qualidade; montagem do genoma de novo; verificação da qualidade da montagem e comparação entre diferentes abordagens de montagem em uma interface amigável ao usuário (COSTA, 2018).

Como estava estruturado em módulos independentes, o *pipeline* podia ser executado em um fluxo de trabalho ou separadamente, de acordo com as necessidades do usuário (Figura 4.5).

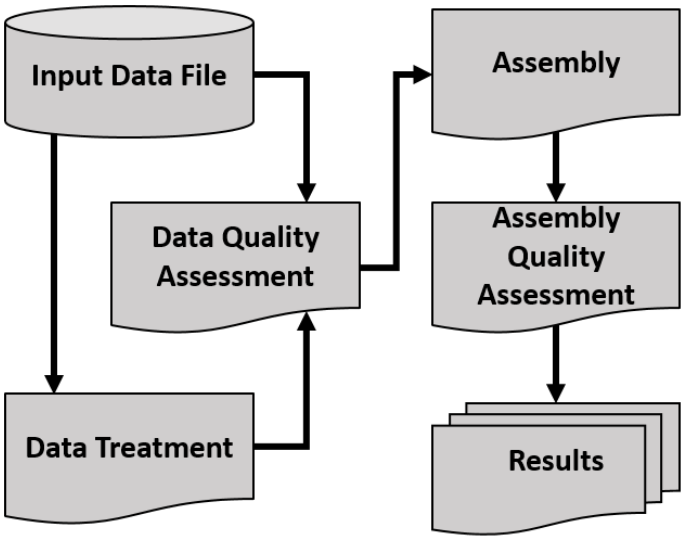

Figura 4.5: Fluxograma de funcionamento do MELC Genomics 1.0 (COSTA, 2018)

4.1.4 Análise dos frameworks

Como descrito anteriormente, os *frameworks* Galaxy, BioExtract Server e Melc Gemonics possuem como principal característica tornar a análise e o processamento das informações simples e amigáveis para os usuários.

Não necessitam de qualquer tipo de interferência para a execução dos processos, sejam eles de análise de dados ou montagem de genomas. A Tabela 4.1 apresenta um comparativo entre os *frameworks* apresentados.

O uso desses *frameworks* torna os processos de bioinformática como alinhamento, verificação da qualidade de dados gerados e montagem mais simples e rápidos, pois é possível em um único sistema executar vários processos ao mesmo tempo.

Entre os *frameworks* avaliados se destacam o Galaxy e o Melc Genomics,

Tabela 4.1: Comparação de sistemas de frameworks utilizados em bioinformática

Características	BioExtract	Galaxy	MELC Genomics 1.0
Interface gráfica	Sim	Sim	Sim
Tipo de acesso	Remoto	Local / Remoto	Local / Remoto
Código aberto	Não	Sim	Sim
Instalação local	Não	Sim	Sim
Sistema operacional	-	UNIX/Linux or Mac OSX	Linux

ambos têm códigos abertos e podem ser instalados tanto localmente em uma rede privada como podem ser utilizados de forma remota.

Em ambos os casos, depois que estiverem instalados, todo o acesso aos *frameworks* é feito através de um navegador web.

4.2 MELC Genomics 2.0

A proposta desta tese é desenvolver uma versão 2.0 que tem como objetivos apresentar uma interface de usuário renovada, com autenticação dos usuários, com opções adicionais que incluem funcionalidade como verificação da qualidade dos dados antes e depois da montagem, acompanhamento da utilização dos recursos computacional em tempo real, além da montagem utilizando aceleradores do tipo *manycore* ou GPU, com o objetivo de diminuir o tempo de montagem. O MELC Genomics 2.0 será executado através de uma interface web simples, com o objetivo de ser intuitiva e amigável, podendo ser acessada com o uso de qualquer navegador.

Para o processo de desenvolvimento do MELC Genomics 2.0 utilizamos linguagem programação em PHP para web, onde foram escritos mais de 900 linhas de programação distribuídas entre os códigos específicos para cada módulo.

Também foram criados mais de 50 *scripts* que coletam informações de acesso ao sistema, consumo de recurso computacional, criação de usuário entre outras. Todas essas informações são transformadas em arquivos que permitem o funcionamento do MELC Genomics.

4.2.1 Módulos

O MELC Genomics 2.0 foi desenvolvido em módulos. Cada módulo do sistema funciona de forma independente, isto é, pode ser executado de forma separada. O *Framework* possui os seguintes módulos (Figura 4.6):

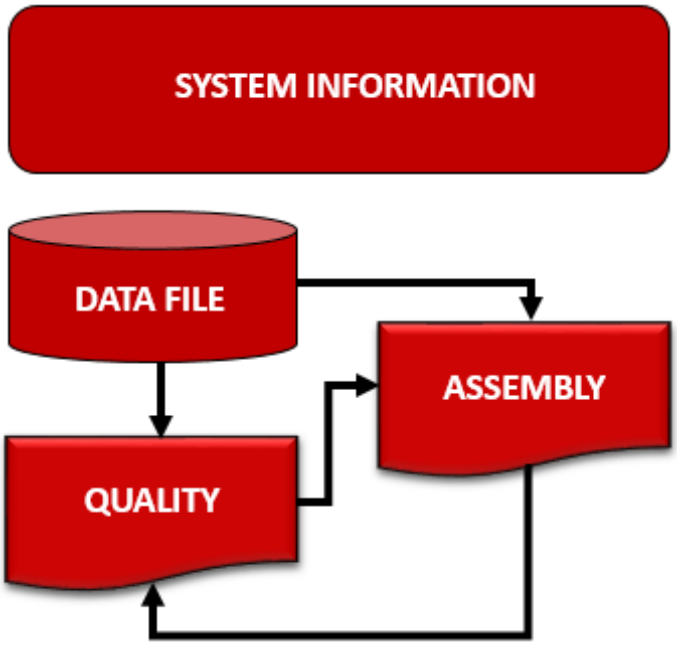

Figura 4.6: Estrutura de módulos de funcionamento do MELC Genomics 2.0

Módulo de SYSTEM INFORMATION Através desse módulo o usuário tem acesso às informações de configuração do sistema, como nome do servidor,

versão de kernel, quantidade de memória física, processos em execução, informações sobre o dispositivo de GPU e capacidade de armazenamento do disco em tempo real (Figura 4.7).

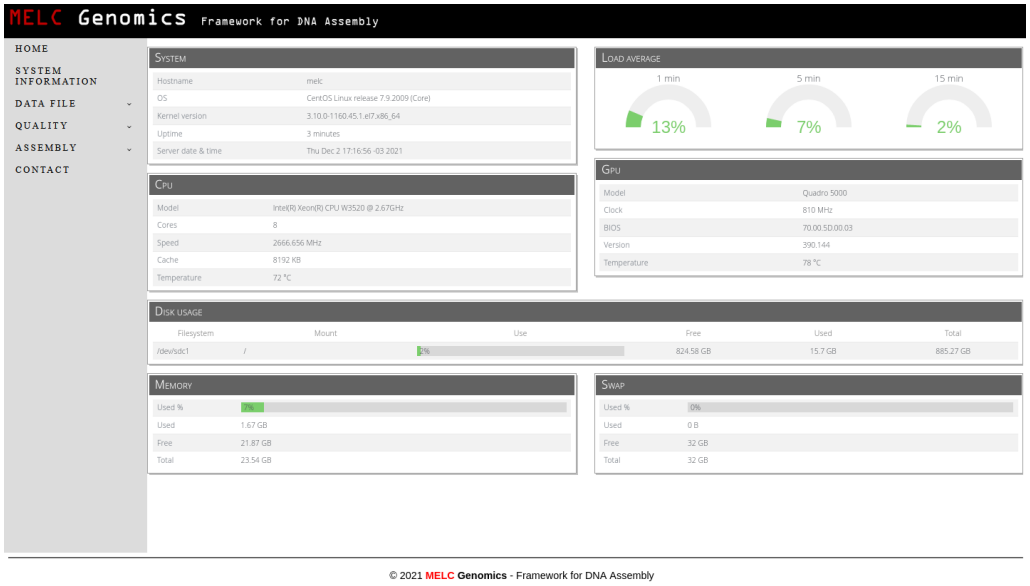

Figura 4.7: Neste módulo o usário tem acesso Informações de utilização de recursos computacionais em tempo real

Sempre que houver uma atualização no sistema, seja física ou lógica, é possível monitorar e identificar a configuração atual.

Quando ocorre a execução de uma montagem, é possível acompanhar a utilização dos recursos computacionais que estão sendo usados no sistema em tempo real, com isto facilmente se identifica se o sistema é capaz de executar ou não uma determinada montagem.

É parte integrante deste módulo o programa de monitoramento chamado *eZ Server* (http://www.ezservermonitor.com) Monitor. O programa *eZ Server* não faz o monitoramento de uso de GPU. Para isto foi necessário o desenvolvimento de um

de código que faça este monitoramento. Após o desenvolvimento deste código, ele foi integrado aos outros módulos já existentes do *eZ Server*.

Foi necessário também fazer algumas alterações nos códigos já existentes para que as informações fossem coletadas de maneira mais eficiente e num tempo menor para as atualizações dos dados a serem exibidos.

Módulo de DATA FILE

Esse módulo é utilizado para enviar os arquivos de sequências de DNA para a montagem dos genomas no sistema. Todos os arquivos devem ser enviados em formato zip. Para isso, deve-se utilizar a área **UPLOAD** desse módulo (Figura 4.8).

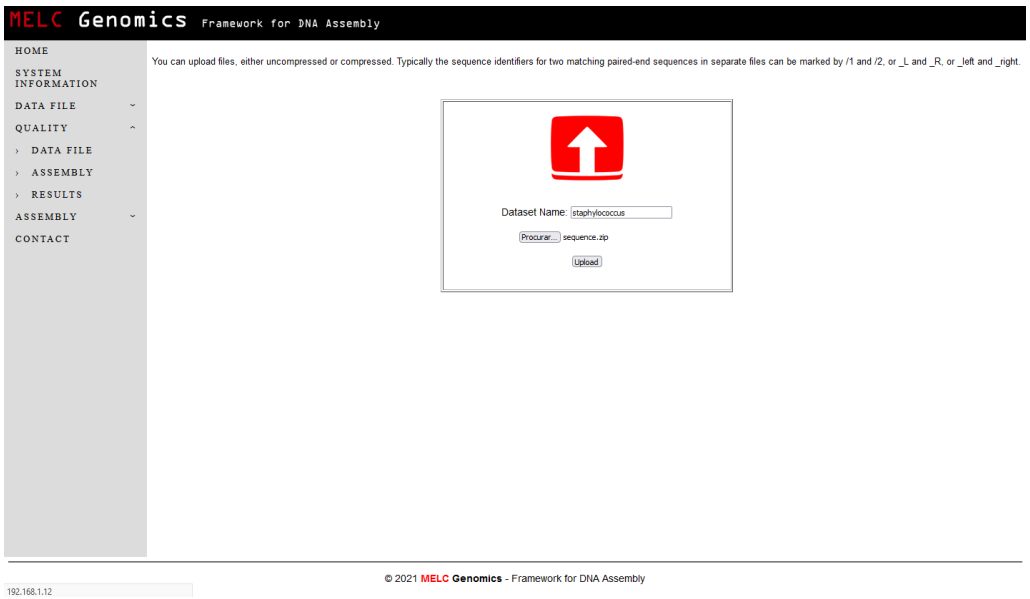

Figura 4.8: Enviando arquivos de sequência para montagem

No campo **Dataset Name**, colocar o nome do conjunto de arquivos sequen-

ciados. Ao apertar o botão **Procurar**, será aberta uma janela para localizar o arquivo com as sequências e depois enviar o arquivo.

Após o envio dos arquivos, eles são descompactados pelo sistema e armazenados em uma área chamada **LIST AND DELETE** desse módulo, com o nome com que foi especificado no campo **Dataset Name**, quando os arquivos foram enviados.

Nessa área é feita a administração dos arquivos, com todas as informações de cada um, como nome, tamanho e data de modificação e estes arquivos podem ser baixados ou removidos do sistema. Para o gerenciamento dos arquivos foi integrado a este módulo o programa QuiXplorer (http://quixplorer.sourceforge.net), porém, também foi necessário executar algumas alterações neste programa que os arquivos sejam acessados. (Figura 4.9).

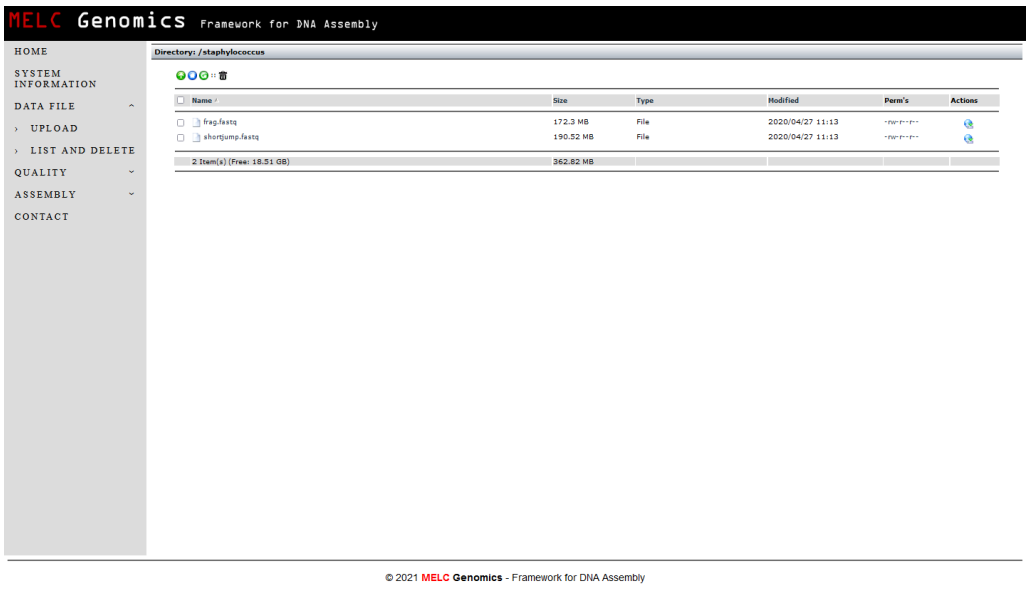

Figura 4.9: Lista de arquivos de sequência para montagem

Módulo QUALITY

Nesse módulo é possível fazer uma análise prévia para verificar a qualidade dos dados enviados para o sistema antes de fazer a sua montagem. Essa análise é feita usando-se a área de **DATA FILE** desse módulo, na qual é feita a escolha de qual arquivo será analisado (Figura 4.10).

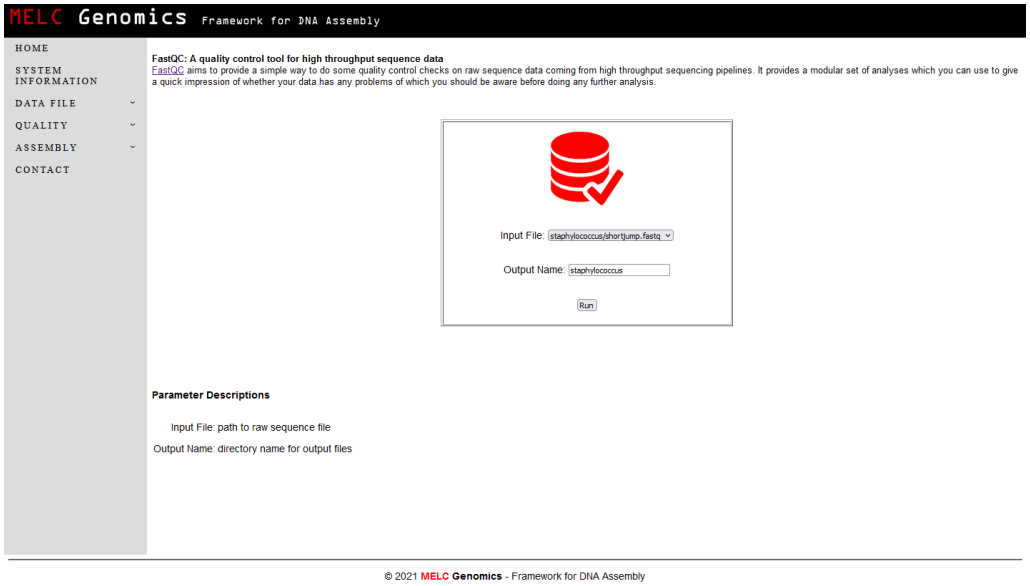

Figura 4.10: Verificação da qualidade dos arquivos de sequência para montagem

No campo **Input File** são listados todos os *dataset* e os seus arquivos para selecionar qual deles será verificada a qualidade. No campo **Output Name**, recomenda-se colocar o nome do diretório e salvar os arquivos de saída. Para a execução desta análise é feito o uso do programa FastQC (ANDREWS, 2010).

Um outro tipo de análise que pode ser feita é a verificação da qualidade do resultado da montagem. Para isso, deve-se utilizar a área de **ASSEMBLY** desse módulo. Assim como na área de **DATA FILE**, os dois campos **Input File** e **Output Name** têm as mesmas características. No campo **Threads** é especificada a quantidade de núcleos que serão utilizados para a execução de verificação da quali-

dade. Para a execução desta análise é feito o uso do programa QUAST (GUREVICH et al., 2013) (Figura 4.11).

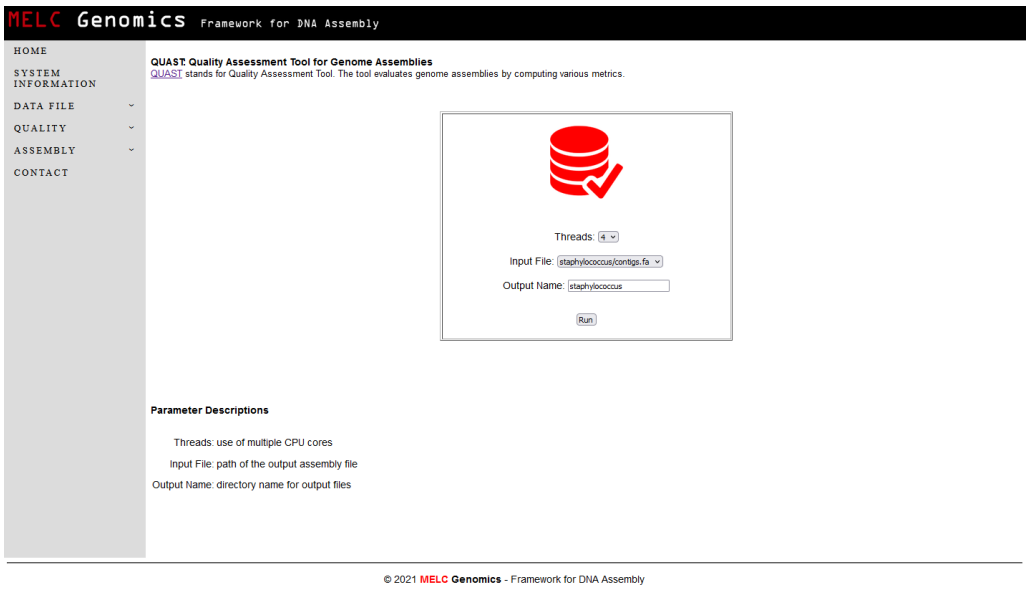

Figura 4.11: Verificação da qualidade dos arquivos após o processo de montagem

Todos os resultados gerados em ambas as análises são armazenados na área de **RESULTS** desse módulo. Nessa área é possível verificar os arquivos gerados, nos quais se tem todas as informações de cada um, como nome, tamanho e data de modificação. E estes arquivos Também podem ser baixados ou removidos do sistema (Figura 4.12).

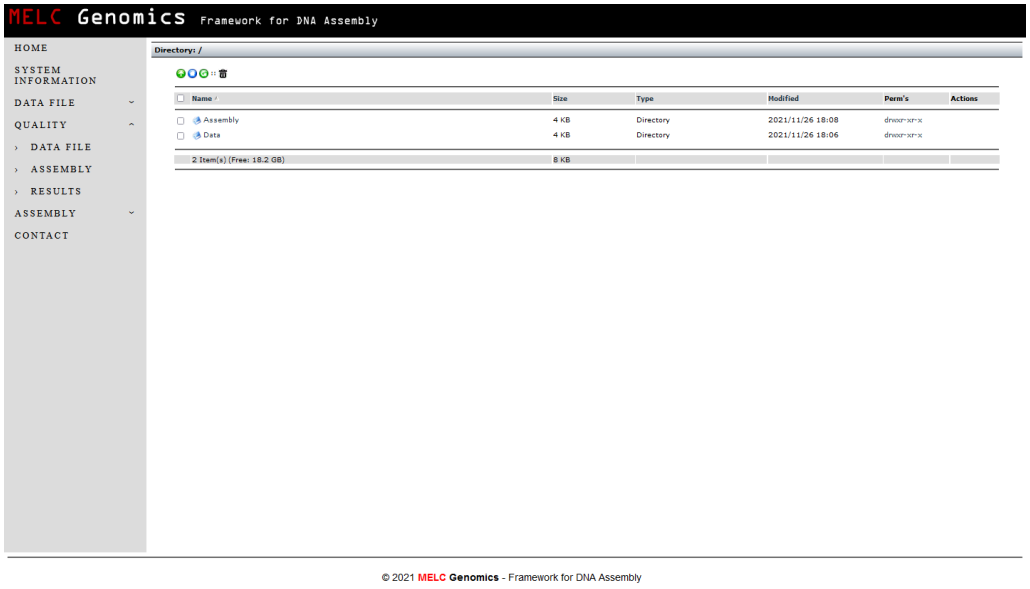

Figura 4.12: Lista dos resultados da verificação da qualidade dos arquivos

Módulo de ASSEMBLY

Esse módulo é utilizado para realizar o processo de montagem dos genomas. Para a sua execução será utilizada a versão de montador Velvet desenvolvido nesta tese para o uso em aceleradores conforme descrito no Capítulo 5.3 (Figura 4.13).

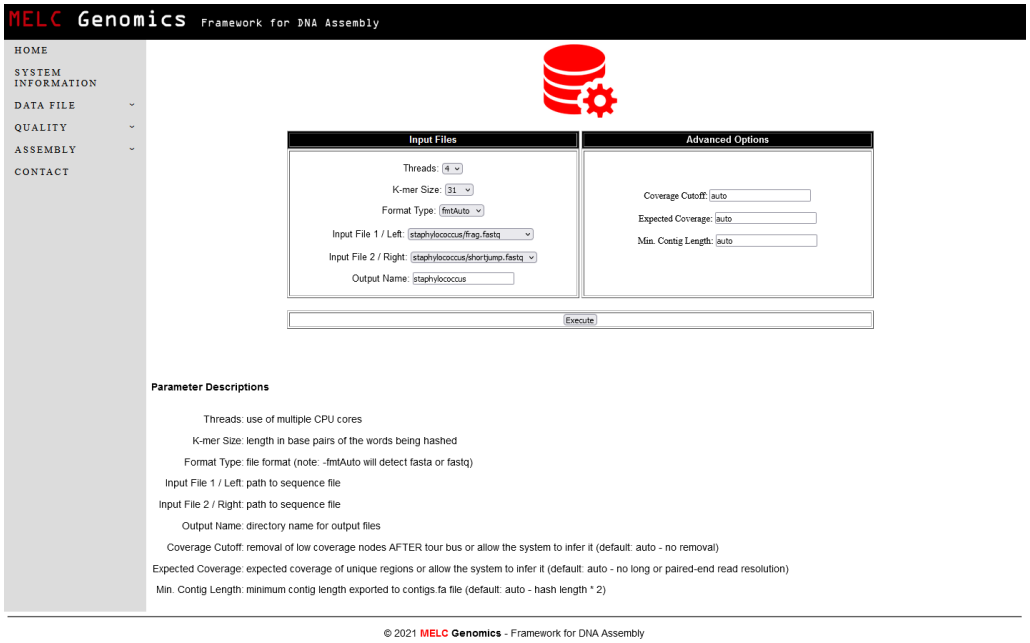

Figura 4.13: Processo de montagem de genoma

No campo **Threads** é especificada a quantidade de núcleos que serão utilizados para a execução da montagem. O campo **K-mer Size** é utilizado para especificar o tamanha do K que será usado. O compo **Format Type**, especifica qual o formato do arquivo. Os arquivos utilizados para a montagem são especificados nos dois campos **Input File 1 / Left** e **Input File 1 / Right**, esses aparecem em uma lista para à escolha dos usuário. No campo **Output Name**, colocar o nome do diretório e salvar os arquivos de saída. O campo **Coverage Cutoff** é utilizado para remover nós da baixa cobertura, o campo **Expected Coverage** pode-se determinar a área de cobertura e o campo **Min. Contig Length** especifica o tamanho mínimo do *contig*. Eles campos tem por padrão o valor "auto".

Todos os resultados gerados pelas montagens dos genomas são armazenados na área de **RESULTS** desse módulo. Para cada execução de montagem de um genoma é gerada uma estrutura com os arquivos de saída de acordo com o nome

dado ao processo.

Nessa área é possível verificar os arquivos gerados, nos quais se tem todas as informações de cada arquivo, como nome, tamanho e data de modificação. E estes arquivos também podem ser baixados ou removidos do sistema (Figura 4.14).

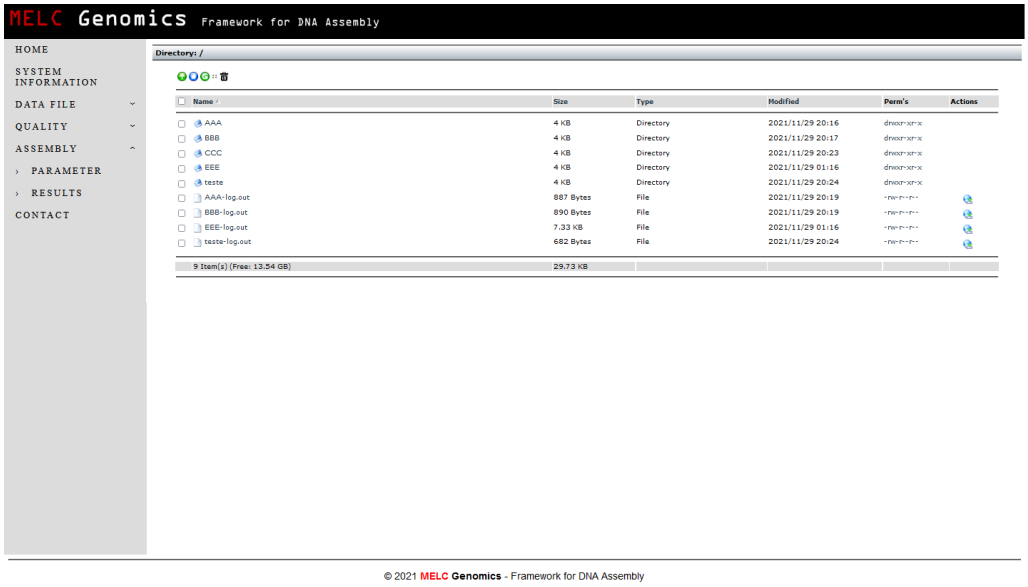

Figura 4.14: Lista dos resultados do processo de montagem

4.2.2 Instalação do MELC Genomics 2.0

A instalação do MELC Genomics 2.0 é fácil e simples de ser feita. O *framework* está disponível para baixar no GitHub através do link: https://github.com/evaldocosta/melc.

MELC Genomics 2.0 foi desenvolvido para o sistema operacional Linux 64 bits. Antes de fazer a instalação do *framework*, é necessária a instalação dos programas seguintes para garantir o seu correto funcionamento:

- Servidor APACHE
- PHP
- R
- JAVA
- PERL
- Matplotlib
- PYTHON

Após fazer a instalação dos programas necessários para uso do *framework*, descompacte os pacotes que foram baixados do GitHub e sega os processos de configuração conforme descrito a seguir.

- **Baixando os arquivos**

  Após baixar o MELC Genomics 2.0, instale os arquivos no diretório onde está instalado o servidor web, sendo que por padrão o servidor Web é instalado no diretório /var/www/html.

- **Configuração do PHP**

  Para configuração do PHP, edite o arquivo /etc/php.ini alterando os valores das linhas seguintes:

file_uploads = On (Whether to allow HTTP file uploads) upload_max_filesize = 2M (Maximum allowed size for uploaded files)

- **Configuração do servidor Apache**

  Para configurar o servidor Apache, edite o arquivo /etc/httpd/conf/httpd.conf para adicionar o endereço IP ou nome do servidor adicionando a linha:

ServerName <IP ou nome do servidor>:80

  Dessa forma o servidor Apache sempre iniciará automaticamente quando o computador for ligado.

```
# chkconfig httpd on
```

Após finalizar a configuração do servidor Apache, inicie o serviço.

```
# service httpd restart
```

- **Configuração de usuário**

Para ter acesso ao sistema é feito a autenticação de usuário e, para adicionar ou remover um usuário basta editar o arquivo .users.txt. Para a adição de usuário e de senha, deve-se usar `":"` como separador, sendo um por linha conforme descrito abaixo.

```
# more .users.txt
admin:admin
```

- **Configuração de contato e recuperação de senha**

Para cadastrar um e-mail de contato ou para recuperação de senha, use o arquivo .contato.txt. e adicione somente o e-mail no arquivo conforme descrito abaixo.

```
# more .contato.txt
user@info.com
```

4.2.3 Executando o MELC Genomics 2.0

Assim como na versão do MELC Genomics 1.0, na versão atual o acesso também é feito por um navegador web acessando o link http://<IP ou nome do servidor>/melc. Na página inicial do *framework* são solicitados usuário e senha para acesso (Figura 4.15). Usuário padrão admin e senha padrão admin.

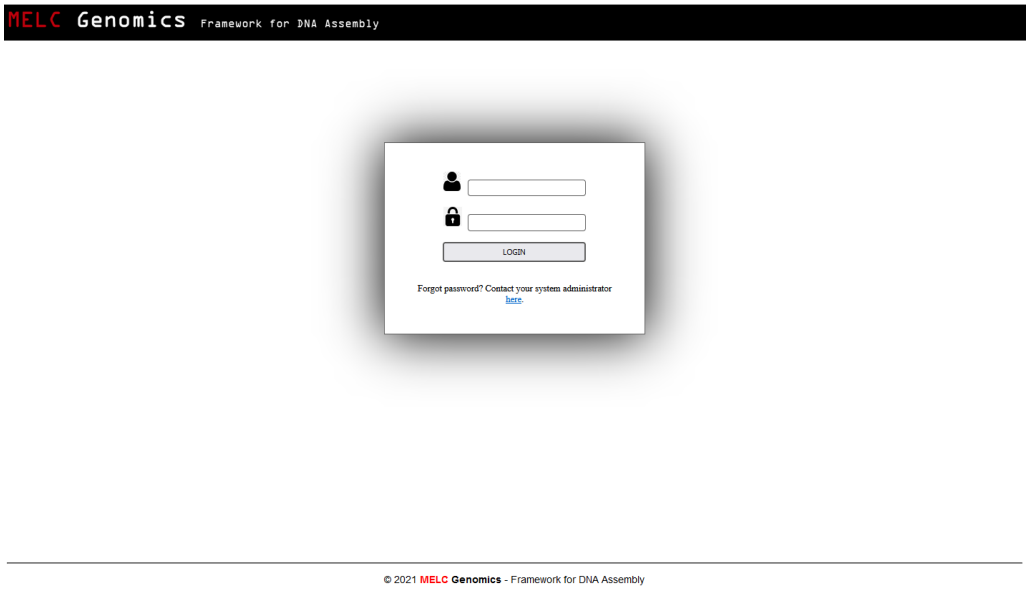

Figura 4.15: Acesso inicial ao MELC Genomics 2.0

Depois de efetuar o acesso usando usuário e senha, o acesso ao *framework* é liberado, como pode ser visto na Figura 4.16.

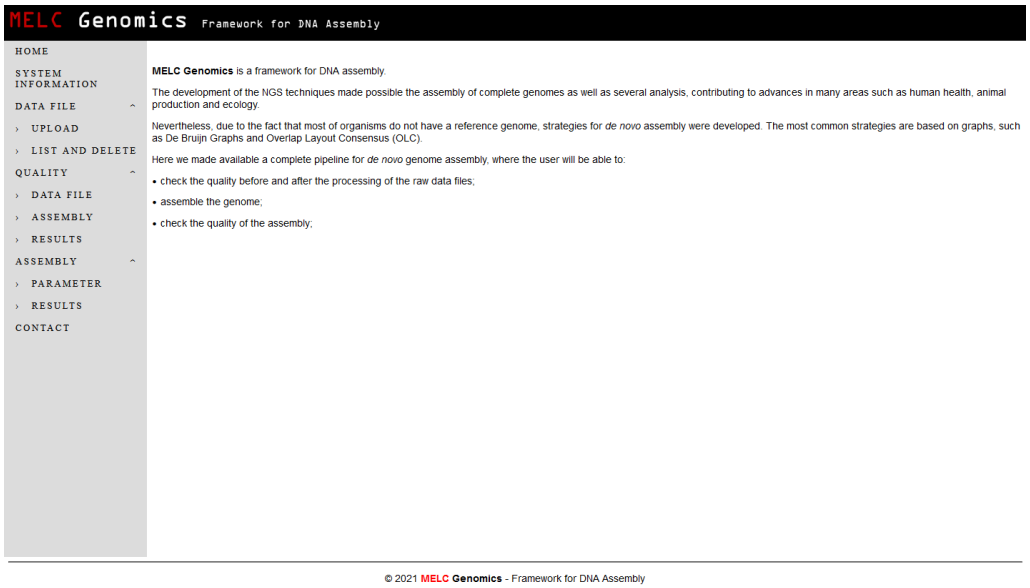

Figura 4.16: Página de acesso ao módulos do MELC Genomics 2.0

O *framework* faz uso do montado Velvet para o processamento dos dados de entrada do usuário para a geração do arquivo final com a montagem do genoma. No capítulo seguinte vamos descrever o funcionamento em geral de um programa de montagem de DNA, com detalhes específicos para o montador Velvet, na versão original com OpenMP e na versão proposta neste trabalho, que faz uso do OpenACC.

5 O MONTADOR DE DNA

Este capítulo descreve as características e o algoritmo de montagem da versão atual do montador Velvet, utilizando o paradigma OpenMP. O modelo de programação em OpenACC, as caraceríticas da placa aceleradora NVIDIA e as modificações propostas nesta tese e a versão modificada em OpenACC, são também discutidas.

O objetivo do desenvolvimento de uma nova versão do montador Velvet é a utilização de novos recursos computacionais com base no uso de GPU. Para o desenvolvimento da nova versão do montador Velvet apresentado nesta tese foi utilizada a GPU NVIDIA Tesla K80 com o paradigma de programação OpenACC.

5.1 Programa de Montagem de DNA

Com a grande quantidade de dados gerados pelo sequenciamento de DNA, tornou-se necessário o desenvolvimento de programas para executar a montagem dessas sequências. Atualmente um grande número de montadores são utilizados em diferentes *frameworks* para tornar o processo de montagem mais simples e rápido para os usuários.

Para a escolha do montador de sequenciamento de DNA utilizado no *framework* apresentado nesta tese, foi feita uma análise comparativa avaliando diversos montadores de genoma *de novo*, utilizando critérios tais como a utilização de recursos computacionais e o tempo de montagem.

Esses montadores possuem abordagens distintas tanto no uso das estraté-

gias quanto na linguagem de programação e no paradigma de programação. As estratégias usadas são:

- *Overlap-Layout-Consensus* (OLC)
- Grafo *de Bruijn*

O uso da abordagem OLC consiste em produzir alinhamentos entre as leituras e identificar sobreposições, agrupando-as em contigs, e então produzir uma sequência consenso. Já na estratégia de uso do grafo *de Bruijn*, as leituras são fragmentadas em sequências menores de tamanho fixo k (*k-mers*) denominadas *seeds* (ZERBINO; BIRNEY, 2008). Um quadro comparativo é apresentado na Tabela 5.1.

Tabela 5.1: Comparação de softwares montadores de sequências genômicas amplamente utilizados em bioinformática

Montador	Linguagem de programação	Paradigma de programação	Algoritmo	Licença
ABySS	C++	MPI	Grafo De Bruijn	Código aberto
ALLPATHS-LG	C++	OpenMP	Grafo De Bruijn	Código aberto
Edna	C++	Pthreads	OLC	Código aberto
SOAPdenovo	C++	Pthreads	Grafo De Bruijn	Código aberto
Velvet	C	OpenMP	Grafo De Bruijn	Código aberto
CABOG	C	OpenMP	OLC	Código aberto
SPAdes	C++	Pthreads	Grafo De Bruijn	Código aberto

Pthreads e OpenMP representam dois paradigmas de multiprocessamento diferentes, sendo Pthreads uma *Application Programming Interface* (API) de baixo nível para trabalhar com o uso de *threads* com um controle extremamente refinado, porém com uso limitado na linguagem de programação.

Por outro lado, o OpenMP e MPI são APIs de alto nível, portáteis e que

não limitam o uso na linguagem de programação. Esses paradigmas de programação possuem maior escalabilidade quando comparados ao Pthreads. Além disso o paradigma MPI é amplamente utilizado para a comunicação de processos através da rede entre computadores.

A avaliação dos montadores de sequência *de novo*: ABySS, Velvet, Edena, SGA, Ray, SSAKE e Perga mostrou que, apesar de todos serem capazes de processar genomas procarióticos ou eucarióticos inteiros, apenas os montadores Velvet e ABySS demonstraram boa eficiência nos quesitos tempo de montagem e utilização de recursos computacionais (KHAN et al., 2018).

Em relação à avaliação quanto à correta montagem das sequências genômicas, realizou-se o mapeamento dos contigs gerados aos respectivos genomas de referência, usando o programa QUAST. Em seguida, foi calculada a fração do genoma à qual os contigs gerados foram mapeados, resultando no percentual de bases de contigs alinhados no genoma de referência (Figura 5.1). Apesar de resultados semelhantes entre os montadores ABySS e Velvet, este último possui também alta escalabilidade para lidar com uma grande quantidade de dados em relação aos demais montadores.

ASSEMBLER	PROKARYOTIC SINGLE-END	PROKARYOTIC PAIRED-END
ABySS	69.8	66.3
Velvet	59.6	57.1
Edena	43.8	51.4
SGA	—	50.4
Ray	48.7	58.8
SSAKE	44.3	13.2
Perga	57.6	51.9

ASSEMBLER	EUKARYOTIC SINGLE-END	EUKARYOTIC PAIRED-END
ABySS	85.4	82.4
Velvet	82.6	85.6
Edena	62.2	90.4
Perga	82.0	83.2
SGA	—	52.4
SSAKE	49.2	74.0
Perga	—	—

Figura 5.1: Montadores avaliados quanto ao desempenho e confiabilidade na montagem de genomas. Observa-se o percentual de bases dos contigs montados e mapeados ao respectivo genoma de referência (KHAN et al., 2018)

Outro importante estudo realizado foi o GAGE (Genome Assembly Gold-standard Evaluations), em que foi realizada a avaliação de alguns montadores de genoma de novo, como ABySS, ALLPATHS-LG, Bambus2, CABOG, MSR-CA, SGA, SOAPdenovo e Velvet. Foram utilizados diferentes conjuntos de dados. Os resultados descrevem o desempenho dos diferentes montadores avaliados no processo de montagem (SALZBERG et al., 2011).

Novamente o *software* Velvet está entre os montadores com os melhores resultados obtidos. Entretanto, foi verificado que esse montador tem um alto consumo de recursos computacionais, principalmente o uso de memória.

Baseado nos resultados apresentados nesses estudos, optou-se por utilizar neste trabalho o montador de Velvet, pois, além de ter apresentado bons resultados na montagem de genomas, trata-se de um programa livre, e seu código permite fazer a implementação utilizando modelo de programação para acelerador.

Uma das principais características desses montadores, que os diferencia dos demais, é o uso do método de grafos *de Bruijn* para o processo de comparação e montagem das sequências. O conceito de grafo *de Bruijn* veio de teoria dos grafos e é utilizado em várias áreas. Em Bioinformática, é amplamente utilizado pelos montadores de fragmentos pequenos (*short reads*). A abordagem Euleriana para o uso no processo de montagem foi descrito por Pevzner em 2001 (PEVZNER; TANG; WATERMAN, 2001).

5.1.1 Grafo *De Bruijn*

Uma definição generalizada de grafo *de Bruijn* se baseia em um conjunto de símbolos de M e N dimensões. Para aplicações de Bioinformática, esses símbolos

estão relacionados com os nucleotídeos. O conjunto de símbolos consiste em quatro caracteres ('A', 'C', 'T', 'G'), e a dimensão é equivalente ao tamanho *k-mer* (pequenas partes da sequência) escolhido.

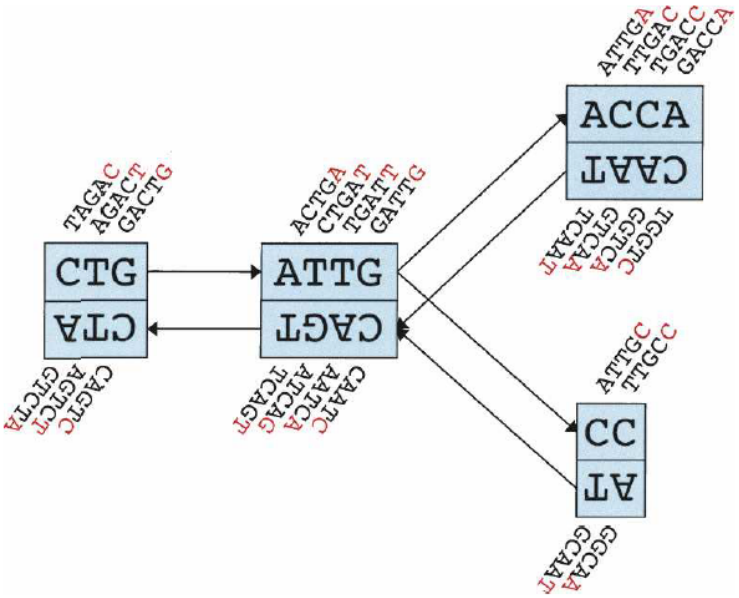

Figura 5.2: Representação esquemática do grafo *de Bruijn* (ZERBINO; BIRNEY, 2008)

A Figura 5.2 mostra como o processo é executado. Os algoritmos de montagem de genomas baseados no grafo *de Bruijn* trabalham em duas etapas. No primeiro passo, os arquivos de *reads* são divididos em tamanhos menores, denominados *seeds*, que são comuns e se repetem frequentemente, com tamanho predefinido k (*k-mers*). Então, o grafo *de Bruijn* é construído a partir da sobreposição k-1 das sequências *seeds*, formando todos os grafos. No passo seguinte, o genoma é obtido a partir dos grafos gerados.

5.2 Montador Velvet

Os montadores de próxima geração (*NGS*) são capazes de realizar a montagem de genomas com base em *reads* de uma maneira muito econômica. A manipulação dos grafos *de Bruijn* é um método realista usado para o alinhamento e que requer desenvolvimentos adicionais para a resolução de situações, como erros de sequenciamento e regiões de repetições do genoma. O programa Velvet utiliza essa metodologia para a montagem de genomas *de novo* e para alinhamentos de *reads* (ZERBINO; BIRNEY, 2008), sem perda de informações, além de ter um procedimento para a correção de erros e de remoção de repetições para criar o arquivo de sequência final.

O montador Velvet pode ser usado para construir rapidamente sequências longas e contínuas a partir de conjuntos de dados de sequências curtas. Essa função é útil, principalmente ao se estudar dados de um novo organismo para o qual um genoma de referência ainda não foi montado ou ao tentar determinar a origem de *reads* não mapeadas.

O Velvet constrói um grafo *de Bruijn* a partir dos *reads* e remove os erros do grafo. Em seguida, ele tenta resolver repetições, com base nas informações disponíveis, sejam sequências longas ou de final emparelhadas, gerando uma montagem das sequências, juntamente com várias estatísticas.

O processo de montagem realizado pelo Velvet está dividido em quatro etapas:

- Etapa 1: Fazer o *hash* das leituras.
  - O Velvet divide cada leitura em *k-mers* de comprimento k.
  - Um *k-mer* é uma subseqüência de comprimento k da leitura.

  - Uma leitura com 36 pares de bases de comprimento teria 6 *31-mers* diferentes.
  - Os *k-mers* e seus complementos reversos são adicionados a uma tabela *hash* para categorizá-los.
  - Cada *k-mer* é armazenado uma vez, mas o número de vezes que aparece também é registrado.

- Etapa 2: construção do grafo de Bruijn.
  - O Velvet adiciona os *k-mers* um a um ao grafo.
  - Os *k-mers* adjacentes se sobrepõem por k-1 nucleotídeos.
  - Um *k-mer* que não tem nenhuma sobreposição k-1 com qualquer *k-mer* que já esteja no grafo inicia um novo nó.
  - Cada nó armazena o número médio de vezes que seus *k-mers* aparecem na tabela *hash*.
  - Sequências diferentes podem ser lidas no grafo, seguindo um caminho diferente através dele.

- Etapa 3: Simplificação do gráfico.
  - Mesclagem de cadeia: quando houver dois nós conectados no grafo sem divergência, mescle os dois nós.
  - Recorte da ponta: as pontas são cadeias curtas (normalmente) de nós que estão desconectados em uma extremidade. Eles serão cortados se seu comprimento for $< 2 \times k$ ou sua profundidade média de *k-mer* for muito menor do que o caminho contínuo.
  - Remoção de bolhas: as bolhas são caminhos redundantes que começam e terminam nos mesmos nós. Elas são criadas por erros de sequenciamento, variantes biológicas ou sequências repetidas ligeiramente variáveis.

  - O Velvet compara os caminhos usando programação dinâmica. Se eles forem muito semelhantes, os caminhos serão mesclados.
  - Remoção de erros: Conexões erradas são removidas usando uma "cobertura de corte". Nós curtos genuínos que não podem ser simplificados devem ter uma cobertura alta. É feita uma tentativa de resolver as repetições usando a "cobertura esperada" dos nós do grafo.
  - Informações de leitura final emparelhadas: o Velvet usa algoritmos chamados "Pebble" e "Rock Band" (ZERBINO; BIRNEY, 2008) para ordenar os nós um em relação ao outro, a fim de organizá-los em *contigs* mais longos.

- Etapa 4: Leitura dos *contigs*
  - Segue as cadeias de nós através do grafo e "faz a leitura" das bases para criar os *contigs*.
  - Onde há uma divergência/convergência ambígua, interrompe-se o *contig* atual e inicia um novo.

A primeira etapa do algoritmo é executada pelo velveth e as três últimas etapas são executadas pelo programa velvetg, ambos da suíte de programa do Velvet.

O tamanho dos *k-mers* utilizados para construir o grafo é um parâmetro muito importante e tem um grande efeito no resultado final da montagem. Geralmente, *k-mers* pequenos criam um grafo com maior conectividade, mais ambigüidade (mais divergências) e "caminhos" menos claros através do grafo. Já *k-mers* maiores produzem grafos com menor conectividade, mas com maior especificidade. Os caminhos no grafo são mais claros, mas são menos conectados e propensos a quebrarem.

O cobertura de corte "c" usada durante a etapa de correção de erros do Velvet também tem um efeito significativo na saída do processo de montagem. Se "c" for

muito baixo, a montagem conterá nós do grafo que são o produto de erros de sequenciamento e conexões incorretas. Se "c" for muito alto, ele pode criar montagens incorretas nos *contigs* e destruir muitos dados úteis.

Cada conjunto de dados tem seus próprios valores ótimos para o tamanho do *k-mer* e para a cobertura de corte usada na etapa de remoção de erros. Escolhê-los de forma adequada é um dos desafios enfrentados pelos novos usuários do software Velvet.

Para executar o montador Velvet, a configuração mínima exigida é de 12 GB de memória física, e o sistema operacional Linux de 64 bits. O Velvet consiste em dois programas que são usados para gerar os resultados: velveth e velvetg. Velveth lê os arquivos de sequências que foram passados como entradas de dados e cria dois arquivos de saída chamados Roadmaps e Sequences, que serão utilizados pelo velvetg. O velvetg usa os arquivos de saída gerados pelo velveth e constrói o grafo *de Bruijn*. Também faz simplificação e correção de erros sobre o grafo. Como último passo, ele extrai os contigs da sequência. O valor usado para o parâmetro k, o tamanho do *k-mer*, foi de 31 (Figura 5.3) para as montagens estudadas nesta tese.

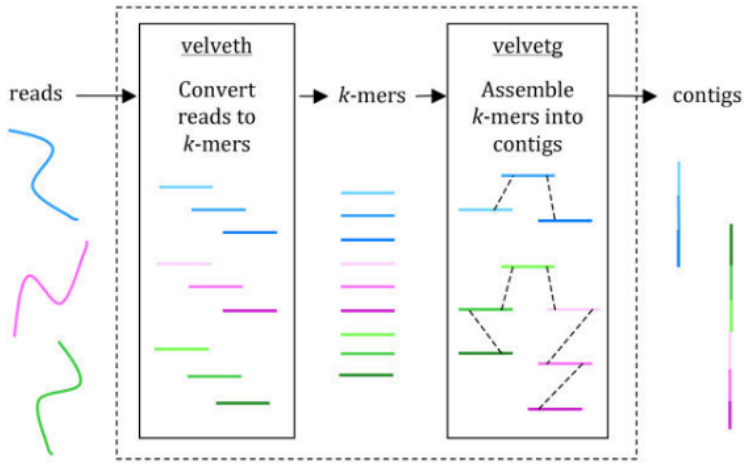

Figura 5.3: Processo de montagem usando o velveth e o velvetg (EDWARDS; HOLT, 2013)

5.3 Modelo de programação OpenACC

O OpenACC é um modelo de programação aberta para computação paralela, desenvolvido com o objetivo de simplificar a programação paralela, oferecendo alto desempenho e portabilidade entre diversos tipos de arquiteturas: *multicore*, *manycore* e GPUs.

O OpenACC é compatível com os modelos de programação OpenMP e MPI, ambas as abordagens podem ser combinadas com o OpenACC. Em geral, as diretivas do OpenACC são muito semelhantes às do OpenMP. Em relação ao CUDA, OpenACC é totalmente compatível, tornando a necessidade de alteração do código a menor possível (COSTA; SILVA, 2019).

O modelo de programação OpenACC tem algumas características que o tornam fácil e simples de utilizar, como:

- é independente de fabricante;
- oculta a complexidade do *hardware* dos programadores;
- requer poucas modificações no código-fonte;
- é mais fácil de programar e depurar que o CUDA;
- possui algumas facilidades que o CUDA não oferece;
- o mesmo código pode ser usado em multicore, manycore e GPUs;
- é similar ao OpenMP (familiaridade);
- é de fácil transição para o OpenMP 4.5 (futuro).

Antes de iniciar o processo de programação utilizando o OpenACC é recomendado que se faça uma análise do código a ser paralelizado seguindo o ciclo de desenvolvimento e análise de código conforme descrito:

1. Analisar o código para determinar quais as regiões mais prováveis que podem ser paralelizadas ou otimizadas.
2. Paralelizar o código iniciando com as regiões com o maior tempo de execução.
3. Otimizar o código para melhorar o tempo de execução observado a partir das alterações executadas.

Na Figura 5.4 é apresentado o ciclo de desenvolvimento e análise do código.

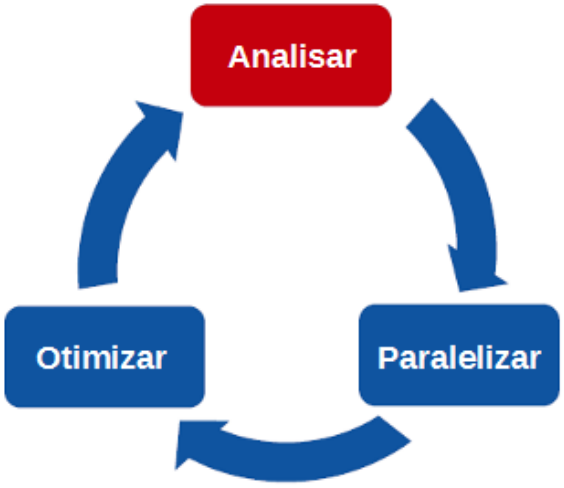

Figura 5.4: Ciclo de desenvolvimento e análise do código (COSTA; SILVA, 2019)

O OpenACC pode ser utilizado em códigos programados em linguagens de programação C/C++ ou Fortran, deve-se atentar a sintaxe correta para tipo de linguagem de programação correta conforme a Figura 5.5.

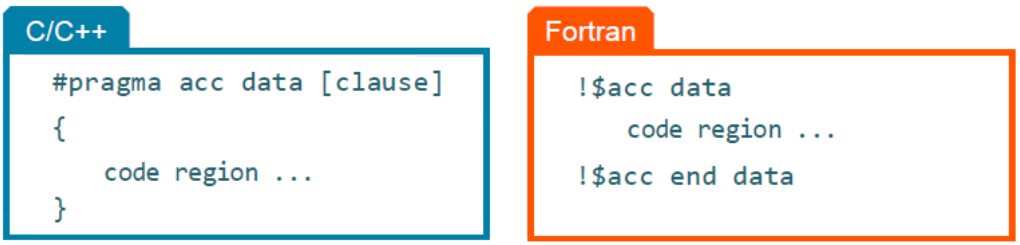

Figura 5.5: Sintaxe do OpenACC em C/C++ e Fortran (COSTA; SILVA, 2019)

5.3.1 Diretivas e cláusulas

O modelo de programação usado no OpenACC é baseado em diretivas e cláusulas. As diretivas são comandos de instrução passadas pelo programador ao compilador. Cláusulas são os parâmetros adicionais atribuídos às instruções usadas nas diretivas.

As diretivas do OpenACC são muito parecidas com as do OpenMP, sendo escritas na forma de pragmas. Existem vantagens em usar diretivas, uma delas é o fato de o código precisar de pequenas modificações, que podem ser feitas de forma incremental, um pragma de cada vez. O uso desse método torna-se especialmente útil para fins de depuração, já que fazer uma única alteração permite identificar rapidamente um erro.

- **Diretiva kernels**

  Utilizar a diretiva **kernels** significa que é informado ao compilador que existem regiões do código que podem ser paralelizadas e que o compilador será o responsável por identificar quais são essas regiões e qual estratégia será utilizada. A estratégia definida pelo compilador pode ser extrair o máximo de paralelismo do código ou executar somente o mínimo de paralelismo. Com uso da diretiva **kernels**, o compilador analisará o código e apenas paralelizará

quando tiver certeza de que é seguro fazê-lo. Em alguns casos, o compilador pode não ter informações suficientes para determinar se é seguro paralelizar um laço, neste caso essa paralelização não será feita.

- **Diretiva parallel**
  A paralelização usando o construtor **parallel** identifica uma região de código que será paralelizada. Quando executada em conjunto com a diretiva **loop**, o compilador gerará uma versão paralela do laço para o acelerador. Desta forma o compilador executa o paralelismo do código de forma direta. Para fazer o paralelismo de vários laços, é necessário que cada laço seja acompanhado de uma diretiva **parallel loop**. Cada região **parallel loop** pode ter diferentes laços, e cada laço pode ser paralelizado e otimizado de forma independente entre eles.

5.3.2 Memória Unificada

As memórias da CPU e GPU são normalmente separadas, ou seja, as informações armazenadas em seus sistemas de memórias não são compartilhadas. A movimentação de dados entre os dois sistemas de memórias é realizada sempre que existe a necessidade de processamento de alguma informação.

Uma forma de solucionar esse problema é o uso de memória unificada. A memória unificada é um espaço de endereçamento único acessível tanto pela CPU como pela GPU. Com o uso dessa tecnologia de *hardware/software*, as aplicações alocam dados que podem ser lidos e escritos tanto pelas CPUs como pelas GPUS (Figura 5.6).

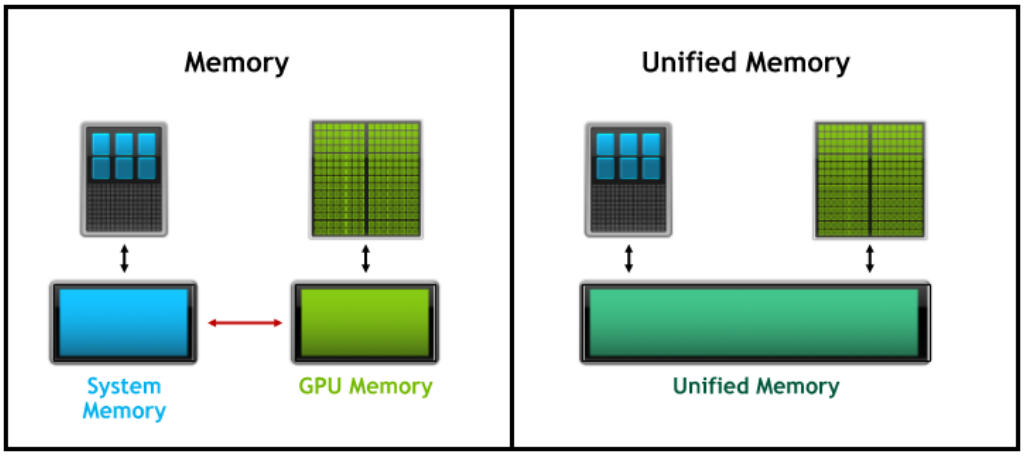

Figura 5.6: Modelo de memória tradicional e memória unificada (HARRIS, 2017)

Para o uso da memória unificada no OpenACC com o compilador PGI, basta utilizar usar a opção *-ta=tesla:managed* na compilação do código do programa. Portanto, as cláusulas e diretivas de dados OpenACC não são necessárias para dados "gerenciados". Elas são essencialmente ignoradas e, de fato, podem ser omitidas.

O uso de memória gerenciada se aplica apenas aos dados alocados dinamicamente. Dados estáticos (variáveis externas e estáticas em C, módulos em Fortran, blocos comuns e variáveis salvas) e dados locais das funções são ainda manipulados pelo ambiente de execução do OpenACC.

Embora o uso da memória unificada ofereça uma grande simplificação da programação, o desempenho final é variável em função de cada aplicação e deve ser avaliado com cuidado antes de sua utilização em produção (HARRIS, 2017).

5.3.3 Movimentação de dados

Um grande fator de impacto de desempenho no processamento paralelo é a movimentação de dados, principalmente quando se faz o processamento dos dados em lugares diferentes. Quando se usa processamento em aceleradores, nem sempre é possível carregar todos os dados para o acelerador. Isso ocorre em geral porque a memória da CPU é maior que a dos aceleradores, embora os aceleradores tenham maior largura de banda (Figura 5.7).

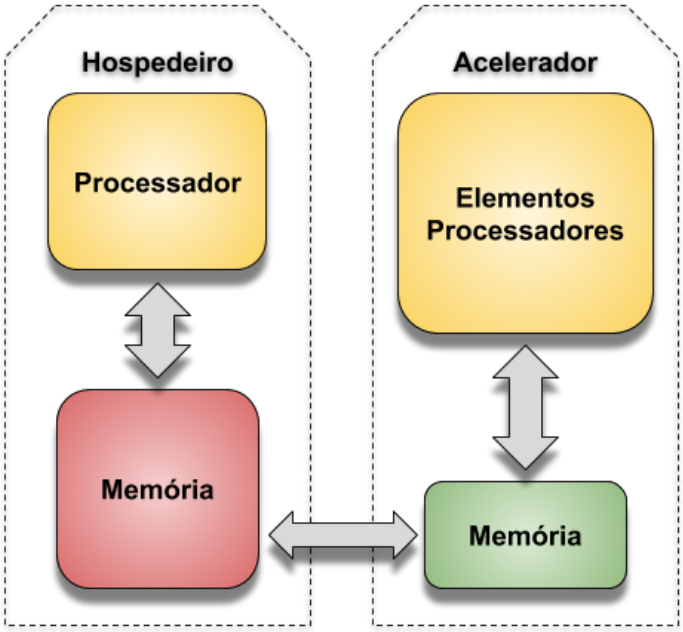

Figura 5.7: Modelo básico de movimentação de dados entre hospedeiro e o acelerador (CHEN, 2017)

A movimentação de dados entre o hospedeiro e o acelerador é feita através de um barramento de E/S, que é lento em comparação com a largura de banda de memória. Por sua vez, o acelerador não pode executar o processamento dos dados

até que eles estejam na sua memória local.

Para realizar a movimentação de dados entre o hospedeiro e o acelerador durante a execução do programa, é necessário o uso das cláusulas de dados. As cláusulas de movimentação de dados podem ser usadas nas diretivas kernels ou parallel e têm como principais caraterísticas:

- são independentes de fabricante;
- definem a região do código na qual os dados permanecem no acelerador;
- definem quais dados são compartilhados entre todos os kernels de uma região paralela;
- realizam transferências de dados explícitas.

5.4 GPU NVIDIA Tesla K80

A GPU NVIDIA Tesla K80 é baseado em tecnologia semelhante à usada nas placas NVIDIA GeForce, porem com características diferentes de memória e taxa de transferência. A Tesla K80 pode ser usada para simulação de vários tipos de modelos, sejam para modelos de cálculos ou modelos de visualização.

A Tesla K80 reduz significativamente os custos do centro processamento de dados, oferecendo desempenho extraordinário, resultando em um número menor de servidores com maior capacidade de processamento. É desenvolvida para aumentar a taxa de transferência de aplicativos reais em 5 a 10 vezes.

A Figura 5.8 mostra o diagrama de blocos da NVIDIA Tesla K80, que possui

duas GPUs Tesla GK210 idênticas, conectadas através de um switch PLX integrado. No diagrama é visto como cada GPU GK210 possui sua própria memória dedicada e como elas se comunicam em uma velocidade de x16 usando um switch PCI-e através de barramento PCIe.

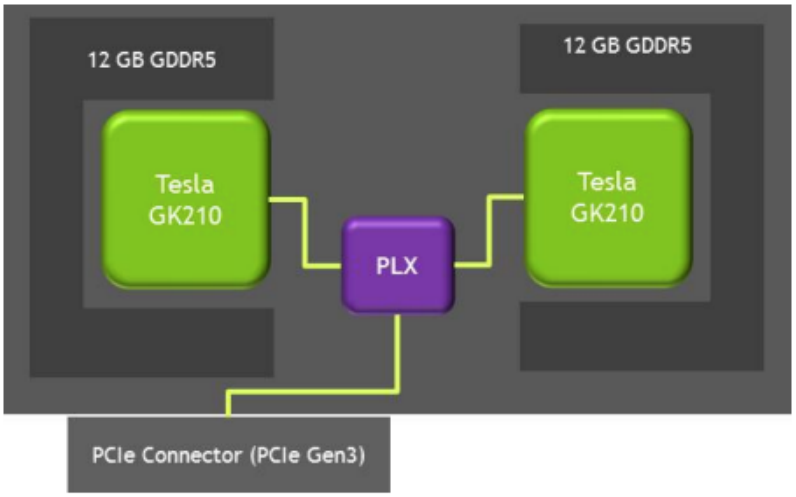

Figura 5.8: NVIDIA Tesla K80 - Diagrama em bloco (NVIDIA, 2015)

Na Tabela 5.2 são apresentadas as características técnicas do acelerador NVIDIA Tesla K80.

Tabela 5.2: Carcterísticas do acelerador NVIDIA Tesla K80

GPU	GK210 (Kepler)
Processo de fabricação	28 nm z
Processadores CUDA	4.992
Core clock	873 MHz
Memória RAM	24 GB GDDR5
Interface de memória	384 bits (2x)
Clock da memória	5 GHz
TDP	375 W

5.5 Compilador PGI

Para compilação dos códigos para a versão do montador em GPU foi utilizado o compilador da Portland Group (a Portland Group foi adquirida pela NVIDIA), que disponibiliza uma versão para uso sem a necessidade de licença e que pode ser baixada através do link:
https://developer.nvidia.com/nvidia-hpc-sdk-downloads.

Esse compilador utiliza de forma mais eficiente a GPU NVIDIA e consegue acesso a recursos específicos da GPU, obtendo, dessa forma, melhores resultados.

Uma outra alternativa é o uso do compilador GNU GCC. A partir da versão 6, o compilador tem suporte ao OpenACC. Como o GNU GCC é um compilador opensource, ou seja, não requer licença para uso.

5.6 Versão do Velvet em OpenACC

5.6.1 Biblioteca de compactação

O Velvet utiliza uma biblioteca de compactação de dados chamada zlib. A última atualização de versão do Velvet 1.2.10 acompanha a versão 1.2.3 biblioteca zlib. Para se obter um melhor desempenho para a nova versão desenvolvida, a biblioteca zlib foi atualizada para a última versão disponível, 1.2.11.

5.6.2 Arquivos paralelos

O montador Velvet é composto por arquivos fonte que são utilizados para geração dos executáveis velveth e velvetg. Esses arquivos fonte utilizam a linguagem "C" e o paradigma de programação OpenMP para a geração da versão do montador Velvet em paralelo para CPU.

- allocArray.c
- graph.c
- graphReConstruction.c
- kmerOccurenceTable.c
- passageMarker.c
- preGraphConstruction.c
- scaffold.c
- splay.c
- splayTable.c

Antes de fazer as alterações necessárias para que o Velvet possa ser executado usando o OpenACC, foi feita uma análise para identificar em quais regiões existe o maior uso de recursos computacionais durante a execução do velveth (Figura 5.9) e velvetg (Figura 5.10). Independentemente de quais genomas foram executados, os trechos dos executáveis onde houve o maior consumo de recursos computacional são os mesmos.

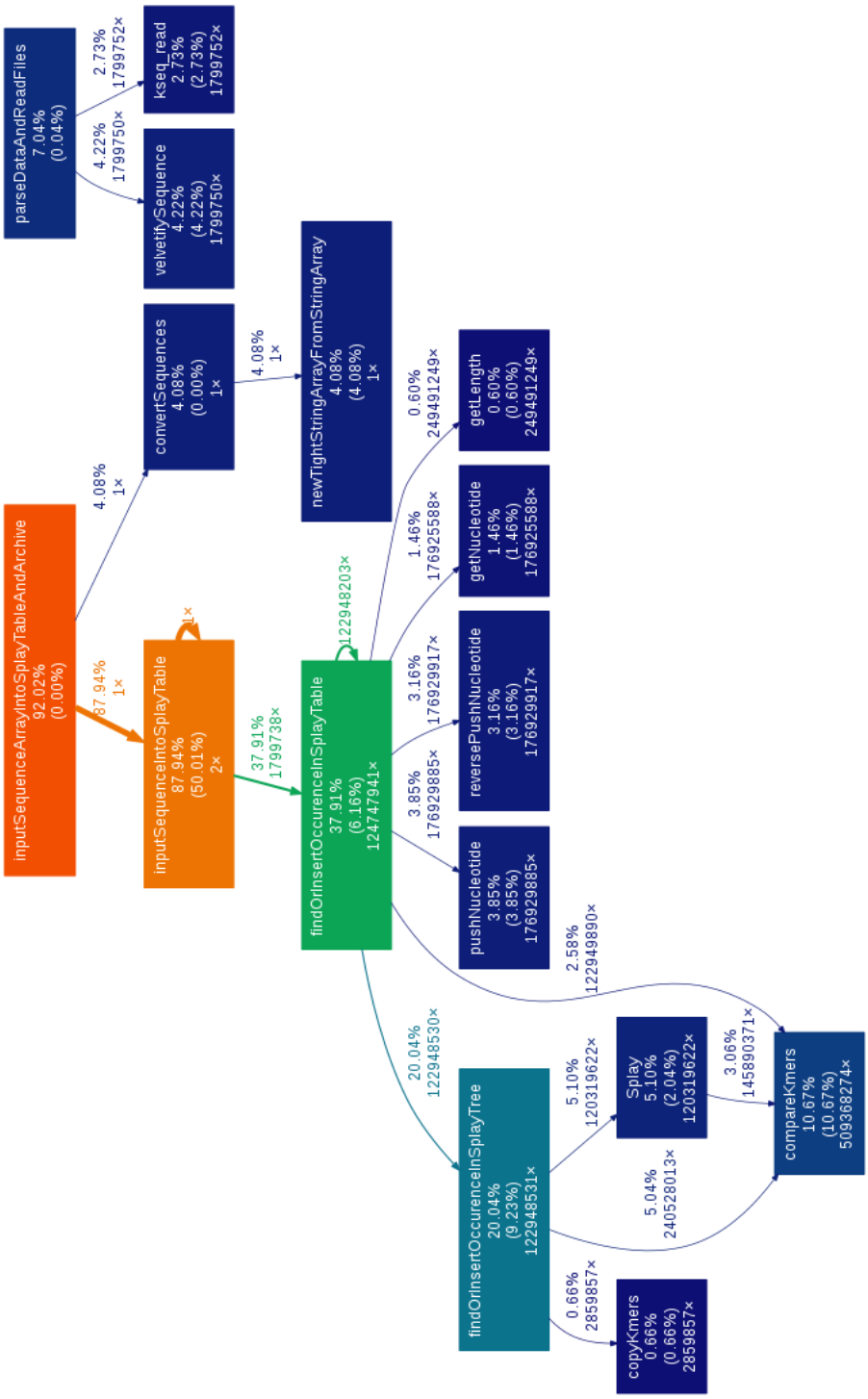

Figura 5.9: Análise de execução do velveth para identificação de regiões com maior uso de recurso computacional

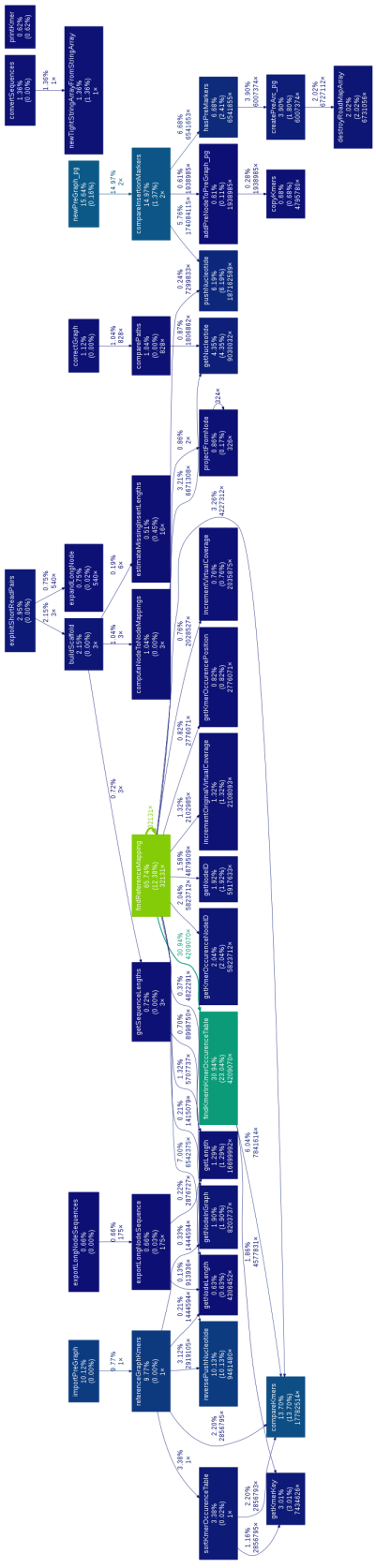

Figura 5.10: Análise de execução do velvetg para identificação de regiões com maior uso de recurso computacional

Fazendo uma análise mais específica dos trechos em que existe o maior uso de recursos, verifica-se que, durante a execução do velveth, a rotina denominada *inputSequenceIntroSplayTable* consome aproximadamente 87% do tempo e dos recursos computacionais, e na execução do velvetg, a rotina *findReferenceMapping* consome aproximadamente 65% do tempo de execução. Essas rotinas, por sua vez, fazem a chamada de outras rotinas. Na Figura 5.11 é apresentado um resumo de utilização dos recursos computacionais utilizados durante a execução dos programas velveth e velvetg.

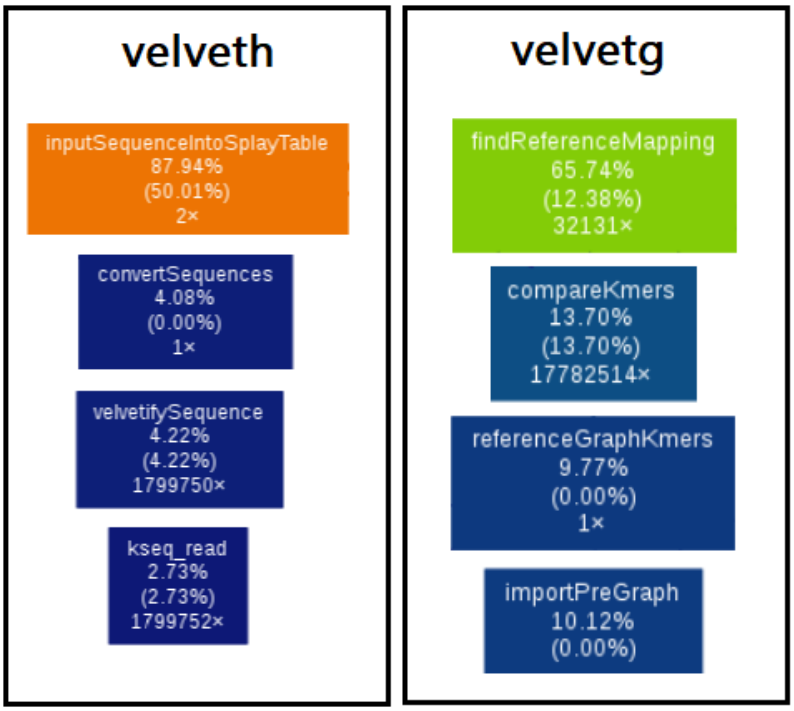

Figura 5.11: Resumo de utilização dos executaveis velveth e velvetg

Os arquivos que trabalham com o paradigma de programação paralela em OpenMP, que permite a alteração para o uso em OpenACC, foram mofificados com uso da diretiva **parallel**, sendo assim possível a obtenção de melhores resultados de desempenho através de parametrizações específicas com uso de cláusulas para obter

a maior utilização da GPU.

Existem arquivos de código no Velvet original que não utilizam nenhum paradigma de paralelismo. Esses arquivos têm como caraterística principal uma grande quantidade de laços, sendo esta uma das condições ideais para o uso do OpenACC. Com a nova codificação desses arquivos, o Velvet passou a ter um melhor desempenho e a utilizar a GPU para execução do seu processamento. Os arquivos alterados podem ser acessados através do link:
https://github.com/evaldocosta/velvetacc/src

A seguir a relação dos arquivos que foram alterados.

- binarySequences.c
- correctedGraph.c
- dfib.c
- fib.c
- graphStats.c
- readSet.c
- recycleBin.c
- run2.c
- tightString.c

Foram alterados todos os laços dos arquivos que suportam o uso do OpenACC de #**pragma omp for** para o #**pragma acc parallel loop**. Em alguns casos foi necessário o uso de cláusulas específicas para acesso à GPU, com isso foi possível obter melhor desempenho. A seguir é apresentada a relação dos pragmas do OpenACC que foram utilizados nos arquivos.

- #**pragma acc parallel loop**
- #**pragma acc parallel loop vector_length**

Entre as cláusulas utilizadas para fornecer um melhor desempenho está a que movimenta os dados entre o hospedeiro e a GPU e as cláusulas que acessam

diretamente a GPU, sendo possível especificar qual GPU a ser utilizada para o processamento.

- **#pragma acc data copy**
- **acc_set_device_num(device_num, acc_device_nvidia)**
- **int num_devices = acc_get_num_devices(acc_device_nvidia)**

Após fazer as alterações necessárias dos arquivos necessários para execução em OpenACC, foi feita uma nova análise para identificar o uso de recursos computacionais durante a execução do velveth e velvetg.

Na Figura 5.12 é possível verificar que o uso de GPU é baixo quando executado o velveth. Isto ocorre porque neste processo é feita a leitura dos arquivos de sequência, que resulta em uma baixa demanda de processamento de GPU.

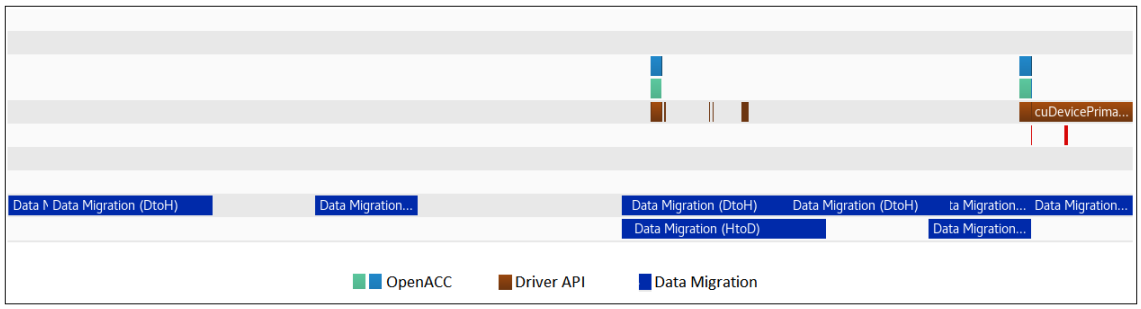

Figura 5.12: Análise de execução do velveth para identificação de regiões com uso de recurso computacional

Na Figura 5.13 é verificado o comportamento contrário quando é executado o velvetg. O maior uso de GPU ocorre durante toda a execução deste processo. Este fato se dá, pois nessa fase é realizada construção do grafo *de Bruijn*.

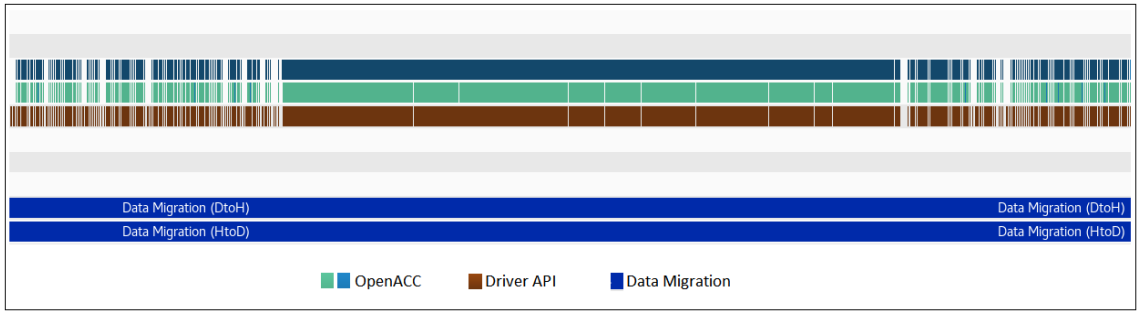

Figura 5.13: Análise de execução do velvetg para identificação de regiões com uso de recurso computacional

Também é possível verificar a movimentação de dados entre o servidor e a GPU em ambas as execuções. Esta movimentação de dados é maior durante a execução do velvetg, em função da construção do grafo *de Bruijn*.

6 AVALIAÇÃO DE DESEMPENHO

Neste capítulo realizamos uma comparação entre a versão do Velvet desenvolvido em OpenMP e a nova versão do montador desenvolvido em OpenACC para uso em GPU.

Todos os testes foram realizados em um computador com dois processadores multicore Intel Xeon E5-2609 v4 (1,7 GHz, 8 núcleos cada, 20 MB de cache), com 128 GB de memória e uma GPU NVIDIA Tesla K80. Os genomas usados nos testes foram armazenados em disco local do tipo SSD (Solid-State Drive). O sistema operacional utilizado foi o GNU/Linux CentOS de 64 bits versão 7.2.

6.1 Dados utilizados

Para a execução do montador Velvet foram utilizados três tipos de genomas que variam em tipo e tamanho, sendo dois genomas de bactérias e o cromossomo humano 21. Os genomas utilizados para a execução dos testes são amplamente citados na literatura científica, e, devido ao *status* de sua montagem, são determinados como genomas de referência.

Todos os conjuntos de dados brutos podem ser baixados pela internet dos servidores do European Nucleotide Archive (ENA) e National Center for Biotechnology Information (NCBI). Os testes foram executados nos seguintes genomas: *Staphylococcus aureus*: NCBI SRA (SRR022868, SRR022865), *Rhodobacter sphaeroides*: NCBI SRA (SRR081522, SRR034528) e *Homo sapiens* (cromossomo 21): ENA (DRR000546). A Tabela 6.1 lista os genomas utilizados, seus tamanhos e nome

dos arquivos.

Tabela 6.1: Informações dos genomas utilizados para execução dos testes

Espécies	Tamanho (Mbp)	Arquivos
Staphylococcus aureus	2,9	NCBI SRA (SRR022868, SRR022865)
Rhodobacter sphaeroides	4,6	NCBI SRA (SRR081522, SRR034528)
Homo sapiens (chr21)	46,7	ENA (DRR000546)

A qualidade da montagem dos genomas apresenta valores de referência semelhantes aos encontrados em outros estudos realizados e arquivado nos principais centros de genomas NCBI (*National Center for Biotechnology Information*) e ENA (*European Nucleotide Archive*). Na Tabela 6.2 são apresentados os valores dos principais indicadores após o processamento dos genomas.

Tabela 6.2: Resultados do processamento dos genomas

Espécies	Contigs		
	Quantidade	N50	GC%
Staphylococcus aureus	175	93	32,55
Rhodobacter sphaeroides	389	44	68,67
Homo sapiens (chr21)	117618	828	40,38

Para a execução dos testes em ambas as versões do montador, tanto em CPU quanto em GPU, foram executados os comandos com os parâmetros descritos na próxima seção.

6.2 Avaliação da versão do Velvet em CPU

Nesta seção será feita uma avaliação da utilização de recursos computacionais executando a versão do montador Velvet em CPU. Como o Velvet é executado com

Staphylococcus aureus:
```
# velveth .  31 -fastq -shortPaired frag.fastq -shortPaired2
shortjump.fastq
# velvetg .
```

Rhodobacter sphaeroides:
```
# velveth .  31 -fastq -shortPaired frag.fastq -shortPaired2
shortjump.fastq
# velvetg .
```

Homo sapiens (Chromosome 21):
```
# velveth .  31 -fastq -shortPaired DRR000546_1.fastq -shortPaired2
DRR000546_2.fastq
# velvetg .
```

dois processos, velveth e velveltg, foi feita uma análise do tempo de execução, uso de processador e de memória em cada um desses programas.

6.2.1 Tempo de execução

O tamanho dos genomas foi o principal fator que influenciou o tempo de processamento desses genomas, por isso o tempo para o processamento dos genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides* foi bem menor que o tempo gasto para o processamento do cromossomo 21 de *Homo sapiens*. Entretanto, a fase de leitura dos arquivos foi a que consumiu a maior fração do tempo de execução do Velvet, independentemente do tamanho do genoma. Na Tabela 6.3 são apresentados os tempos totais de processamento aferidos.

Tabela 6.3: Tempo total de processamento dos genomas na versão CPU

Espécies	Tempo em segundos (s)
Staphylococcus aureus	358
Rhodobacter sphaeroides	568
Homo sapiens (chr21)	7713

6.2.2 Utilização de memória

A utilização de memória variou de acordo com o genoma sendo montado. Essa variação ocorre, principalmente, pelo tamanho do genoma, ou seja, quanto maior o genoma a ser montado, maior a quantidade de memória utilizada.

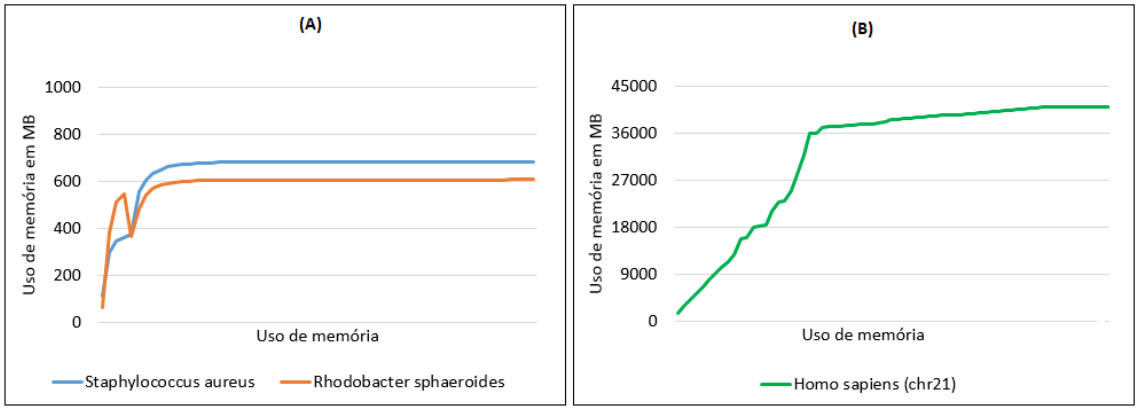

Figura 6.1: utilização de memória do velveth. (A) utilização de memória dos genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides* (B) utilização de memória do cromossomo 21 de *Homo sapiens*

Como pode ser visto na Figura 6.1, os genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides* tiveram uma utilização de memória praticamente constante durante a execução do velveth, com uma ocupação em torno de 600-700 MB. Isso se justifica por serem genomas menores. Para o cromossomo 21 de *Homo sapiens*, a utilização de memória aumentou gradativamente durante a primeira metade da exe-

cução do velveth, mantendo-se praticamente constante na segunda metade, porém em valores bem maiores, acima de 36 GB, do que os outros genomas.

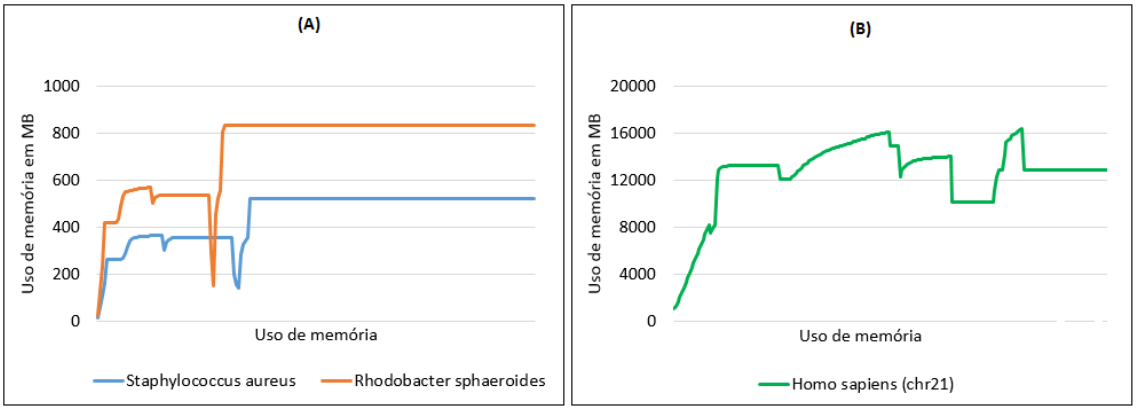

Figura 6.2: utilização de memória do velvetg. (A) utilização de memória dos genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides* (B) utilização de memória do cromossomo 21 de *Homo sapiens*

Durante a execução do velvetg, assim como ocorreu com o velveth, os genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides*, após um rápido aumento inicial, a utilização de memória se manteve constante, em torno de 500-800 MB, até o fim do programa. Já para a espécie *Homo sapiens* utilização foi em torno de 12 GB, com algumas variações, até o final da execução do programa (Figura 6.2).

6.2.3 Utilização de CPU

Na Figura 6.3 é vista a distribuição da utilização de CPU durante o processamento do genoma *Staphylococcus aureus*, tanto na fase de leitura dos arquivos (programa velveth), como na fase de montagem do genoma (programa velvetg).

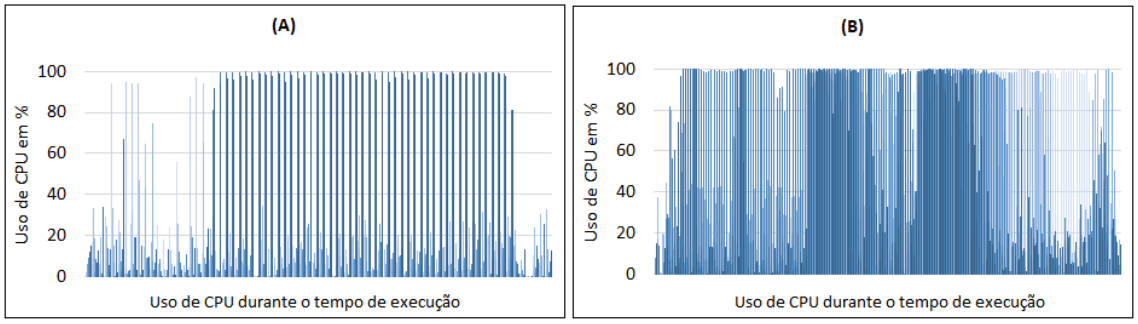

Figura 6.3: (A) utilização de CPU durante a leitura dos arquivos do genoma *Staphylococcus aureus*. (B) utilização de CPU durante a montagem do genoma *Staphylococcus aureus*

Durante a fase de leitura dos arquivos do genoma existe uma baixa utilização de CPU, basicamente com a ocupação de dois processadores. Já durante a fase de montagem do genoma, a utilização de CPU passa a ser maior, envolvendo todos os processadores disponíveis, com uma utilização média em torno de 35% (Figura 6.4).

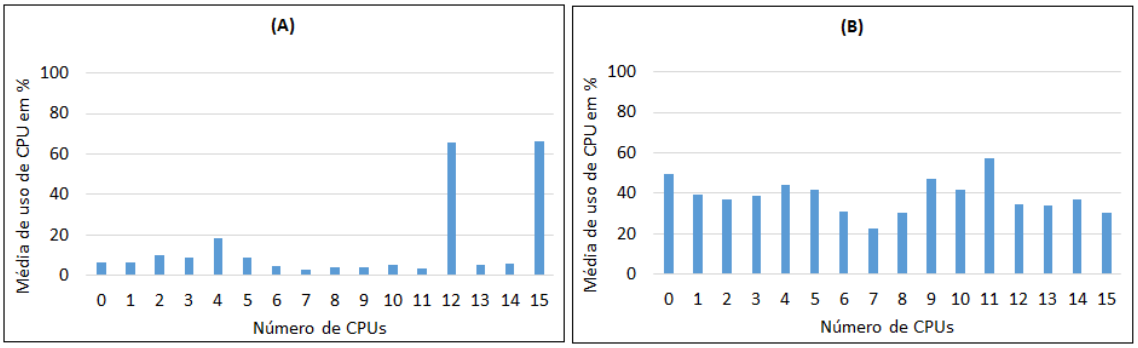

Figura 6.4: (A) média de utilização de CPU durante a leitura dos arquivos do genoma *Staphylococcus aureus*. (B) média de utilização de CPU durante a montagem do genoma *Staphylococcus aureus*

Como pode ser visto nas Figura 6.5 e Figura 6.6, durante as fases de leitura e de montagem do genoma *Rhodobacter sphaeroides*, há uma utilização um pouco maior de CPU, em média de utilização de 40 %, mas não muito mais significativa

em relação às fases de leitura e montagem do genoma *Staphylococcus aureus*.

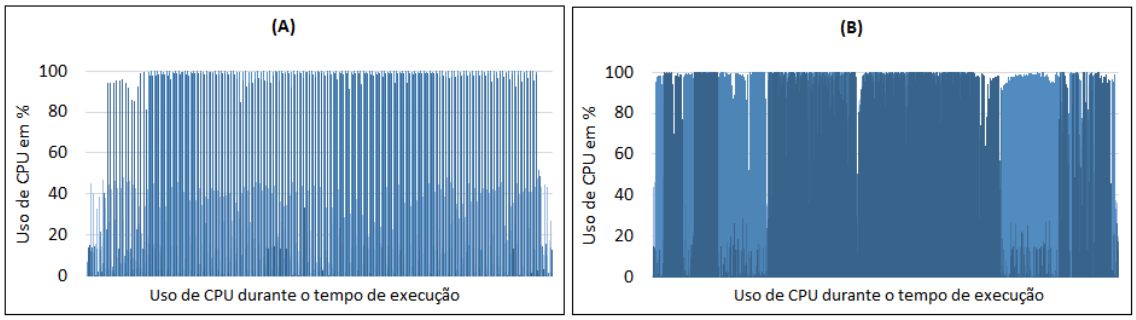

Figura 6.5: (A) utilização de CPU durante a leitura dos arquivos do genoma *Rhodobacter sphaeroides*. (B) utilização de CPU durante a montagem do genoma *Rhodobacter sphaeroides*

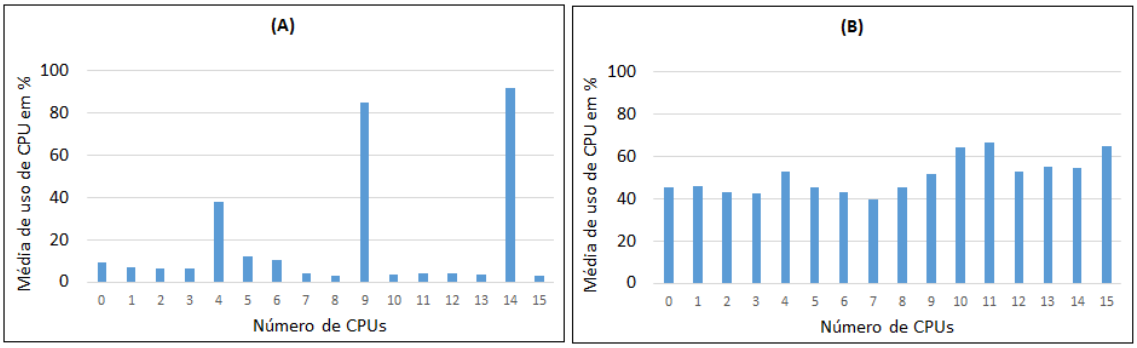

Figura 6.6: (A) média de utilização de CPU durante a leitura dos arquivos do genoma *Rhodobacter sphaeroides*. (B) média de utilização de CPU durante a montagem do genoma *Rhodobacter sphaeroides*

Embora a montagem do cromossomo 21 de *Homo sapiens* apresente o mesmo comportamento que os demais genomas, a utilização de CPU foi maior em termos percentuais, claramente devido ao tamanho do cromossomo. Observa-se que, à medida que o tamanho do cromossomo aumenta, também existe um aumento em percentual do uso de CPU, com ocupação média em torno de 60 %, conforme mostram as Figuras 6.7 e 6.8.

Pode-se observar na Figura 6.8 que novamente apenas dois processadores estão ativamente envolvidos na tarefa de leitura dos arquivos do genoma, mas com utilização maior que nos outros casos, conforme ilustrado nas Figuras 6.7, 6.5 e 6.3.

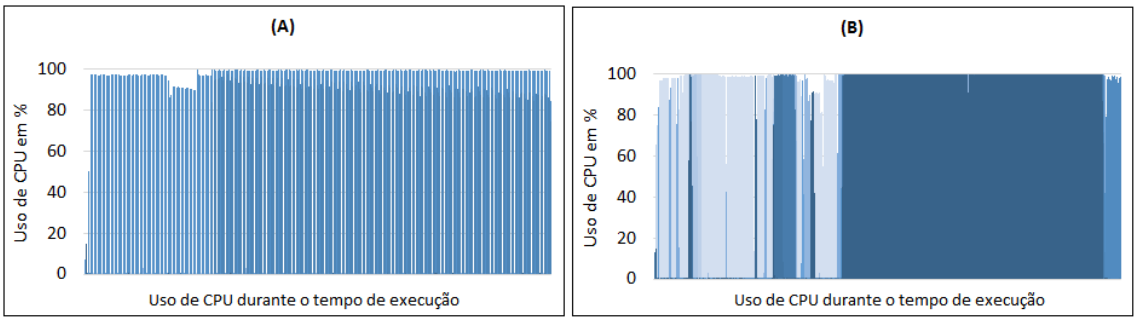

Figura 6.7: (A) utilização de CPU durante a leitura dos arquivos do genoma *Homo sapiens*. (B) utilização de CPU durante a montagem do genoma *Homo sapiens*

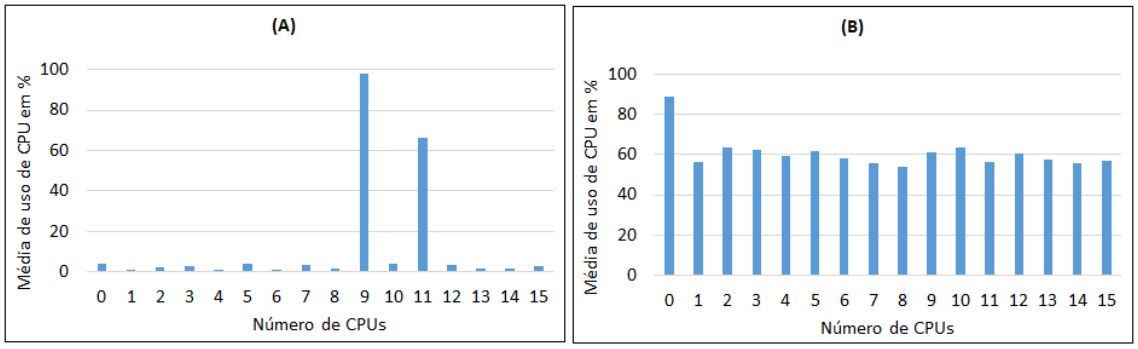

Figura 6.8: (A) média de utilização de CPU durante a leitura dos arquivos do genoma *Homo sapiens*. (B) média de utilização de CPU durante a montagem do genoma *Homo sapiens*

6.3 Avaliação da nova versão do Velvet em GPU

Nesta seção será feita uma avaliação da utilização de recursos computacionais executando a nova versão que foi elaborada para o montador Velvet com uso de diretivas OpenACC e aceleração com uso da GPU. Como o Velvet é executado com dois processos, velveth e velveltg, foi feita uma análise do tempo de execução, uso

de processador e de memória, tanto na CPU como na GPU, para cada um desses processos.

6.3.1 Tempo de execução

Durante a execução do velveth, os tempos obtidos entre a versão do montador desenvolvida para uso em CPU e a nova versão desenvolvida em GPU foram próximos, não havendo um ganho significativo de uma versão em relação à outra.

Porém, os tempos obtidos na execução na nova versão do velvetg tiveram uma grande diminuição em comparação com a versão para uso em CPU. Essa diminuição do tempo se deve à maior utilização dos recursos paralelos.

Isso resultou em uma redução significativa no tempo total de montagem, conforme visto na Tabela 6.4. Na seção seguinte faremos uma comparação com os resultados obtidos com as duas versões.

Tabela 6.4: Tempo total de execução para a montagem dos genomas utilizando a versão em GPU

Espécies	Tempo em segundos (s)
Staphylococcus aureus	65
Rhodobacter sphaeroides	113
Homo sapiens (chr21)	2279

6.3.2 Utilização de memória

O consumo de memória, como na versão do montador para CPU, também varia de acordo com o tamanho do genoma utilizado. As Figuras 6.9 e 6.10 apresentam a utilização de memória para a nova versão do montador para GPU.

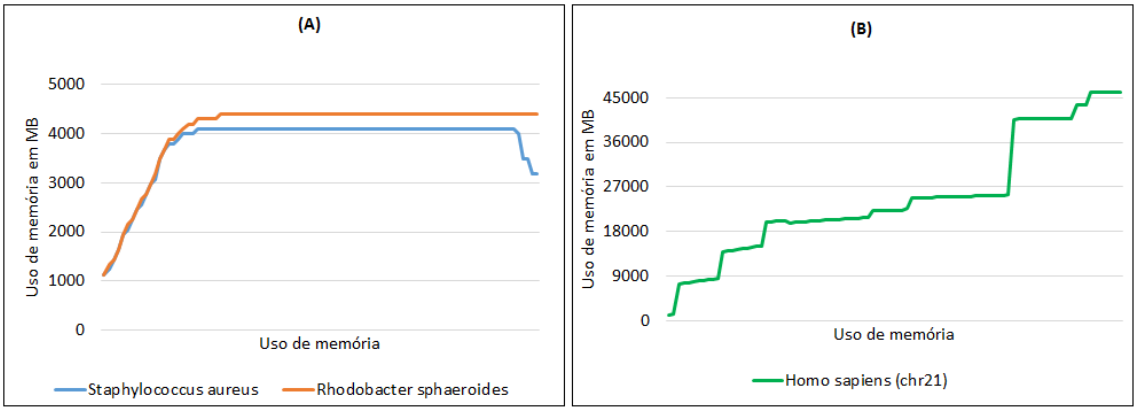

Figura 6.9: utilização de memória do velveth. (A) utilização de memória dos genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides*. (B) utilização de memória do cromossomo 21 de *Homo sapiens*

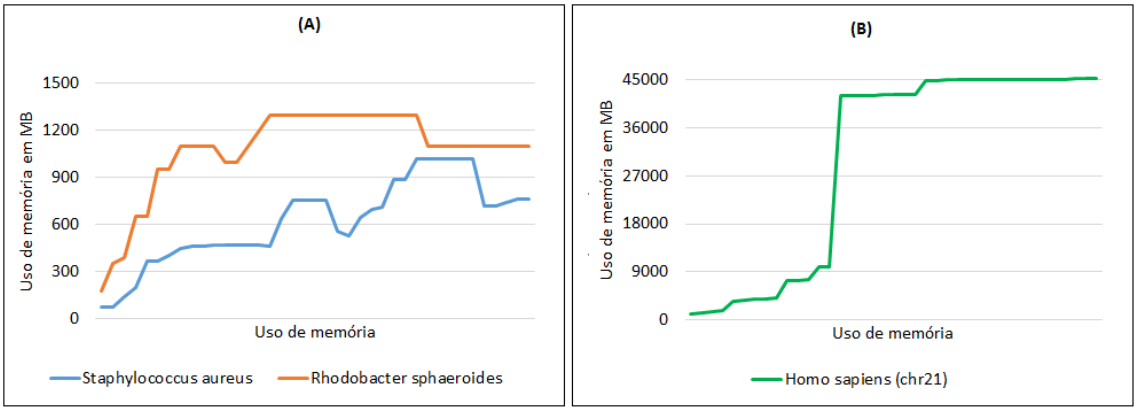

Figura 6.10: utilização de memória do velvetg. (A) utilização de memória dos genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides*. (B) utilização de memória do cromossomo 21 de *Homo sapiens*

6.3.3 Utilização de CPU

Durante a execução do velveth, houve um aumento no consumo de CPU, relacionado à movimentação dos dados entre a CPU e o acelerador. O processamento de todos os genomas apresentou esse comportamento. Na Figura 6.11 e Figura

6.12 é vista a distribuição da utilização de CPU durante a execução do genoma *Staphylococcus aureus*.

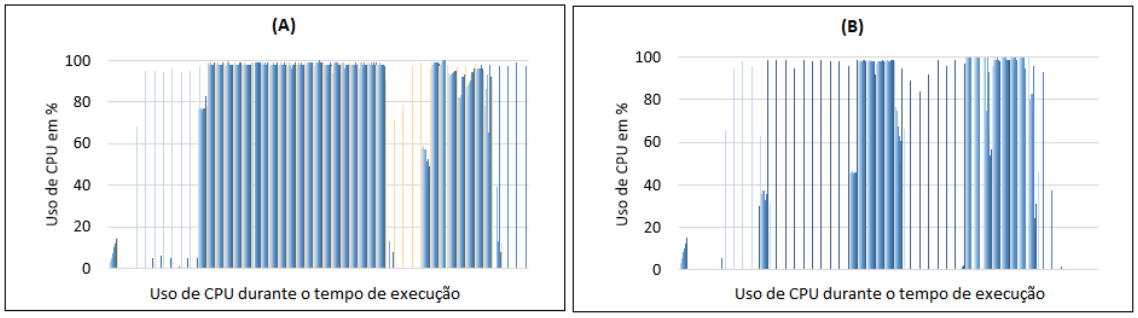

Figura 6.11: (A) utilização de CPU durante a leitura dos arquivos do genoma *Staphylococcus aureus*. (B) utilização de CPU durante a montagem do genoma *Staphylococcus aureus*

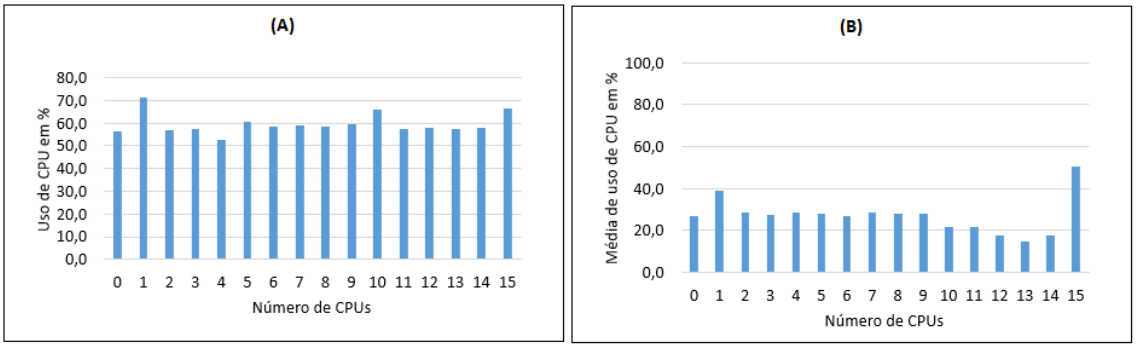

Figura 6.12: (A) média de utilização de CPU durante a leitura dos arquivos do genoma *Staphylococcus aureus*. (B) média de utilização de CPU durante a montagem do genoma *Staphylococcus aureus*

Como observado na Seção 6.2.3, o uso de CPU é baixo durante a execução do programa velveth, na versão com OpenMP, pois esse processo faz a leitura dos arquivos de sequências curtas. Quando diretivas do OpenACC são inseridas nos códigos fontes do programa velveth, elas aumentam o uso de CPU, mesmo que isso não seja necessário, pois há uma tentativa de utilização das GPUs. Neste caso, o uso

do OpenACC não tem um ganho significativo em relação OpenMP, que compense este aumento de uso de CPU.

Como pode ser visto nas Figuras 6.13 e Figura 6.14, durante as fases de leitura e montagem do genoma *Rhodobacter sphaeroides*, a utilização de CPU tem o mesmo comportamento encontrado no processamento do genoma *Staphylococcus aureus*. Pois, como já observamos, são genomas que possuem características similares.

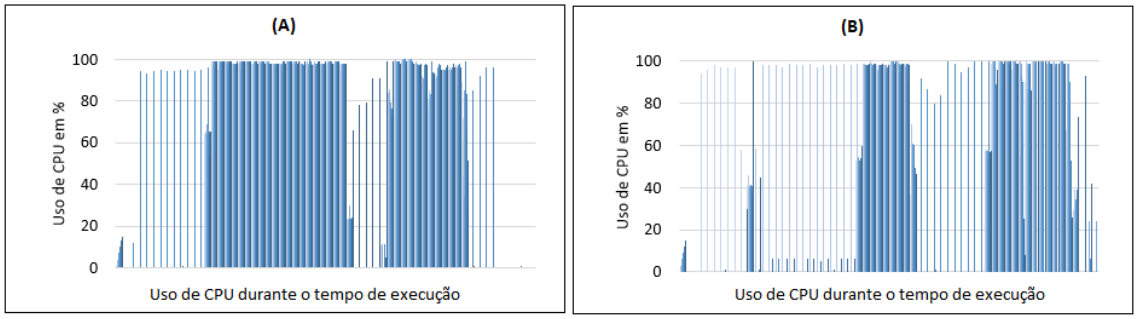

Figura 6.13: (A) utilização de CPU durante a leitura dos arquivos do genoma *Rhodobacter sphaeroides*. (B) utilização de CPU durante a montagem do genoma *Rhodobacter sphaeroides*

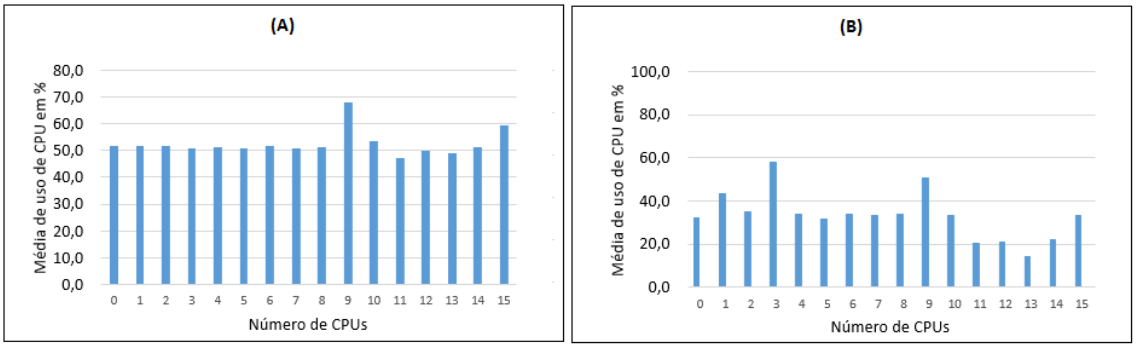

Figura 6.14: (A) média de utilização de CPU durante a leitura dos arquivos do genoma *Rhodobacter sphaeroides*. (B) média de utilização de CPU durante a montagem do genoma *Rhodobacter sphaeroides*

Embora o genoma *Homo sapiens (chr21)* seja maior que os demais genomas

avaliados, a relação de consumo de CPU teve comportamento similar aos demais casos, conforme visto nas Figuras 6.15 e 6.16.

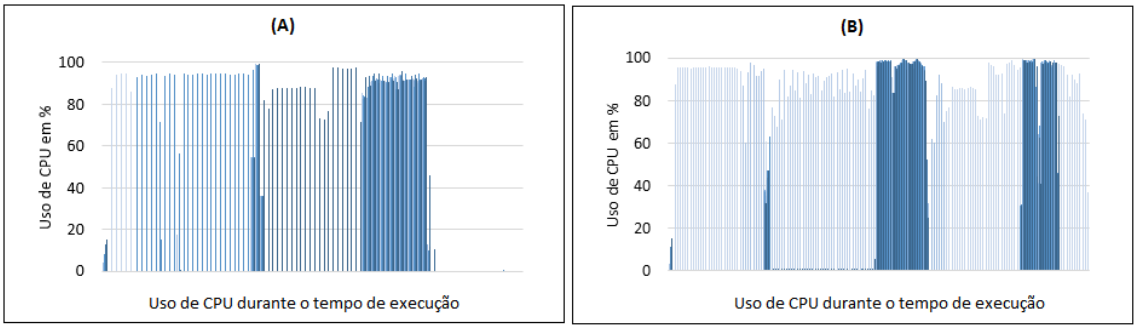

Figura 6.15: (A) utilização de CPU durante a leitura dos arquivos do genoma *Homo sapiens (chr21)*. (B) utilização de CPU durante a montagem do genoma *Homo sapiens (chr21)*

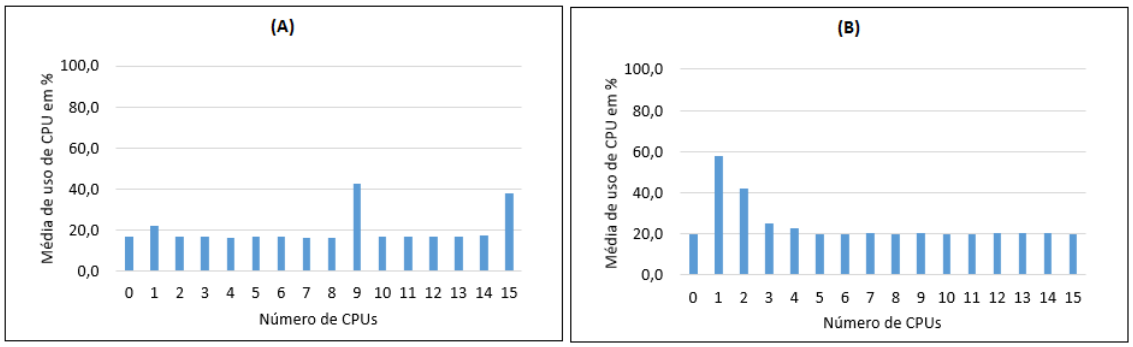

Figura 6.16: (A) média de utilização de CPU durante a leitura dos arquivos do genoma *Homo sapiens (chr21)*. (B) média de utilização de CPU durante a montagem do genoma *Homo sapiens (chr21)*

6.3.4 Utilização de GPU

A análise do uso de GPU é dividida em duas partes: uso dos núcleos de processamento de GPU e quantidade de memória utilizada.

- **Utilização dos núcleos de processamento**

Na Figura 6.17 são apresentados os resultados da média de utilização dos núcleos de processamento de GPU (*cuda cores*) durante a fase de montagem dos genomas.

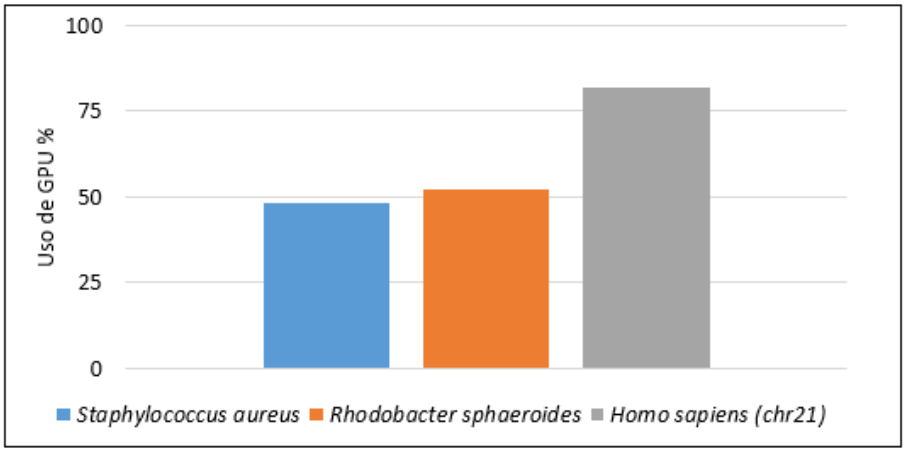

Figura 6.17: Média de utilização dos núcleos de processamento de GPU

Os genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides* obtiveram valores médios de uso menores que a média de uso do genoma *Homo sapiens (chr21)*, que foi cerca de 60% maior. Isso ocorreu porque os genomas de maior tamanho ocupam um número maior de núcleos da GPU (*CUDA core*) para o seu processamento, mas essa ocupação não é diretamente proporcional ao tamanho final de dados gerados, que é em torno de 10 vezes para os genomas aqui utilizados.

- **Quantidade de memória utilizada**

Assim como na média de utilização dos núcleos da GPU, o mesmo comportamento pode ser visto em relação à utilização média de memória da GPU, conforme

apresentado na Figura 6.18. Ou seja, genomas maiores fazem uso de mais recursos de memória da GPU, mas em nenhum caso foi consumido, em média, um total de memória maior do que os recursos disponíveis na GPU. Ainda, como no caso anterior, o consumo de memória também não é diretamente proporcional ao tamanho dos genomas.

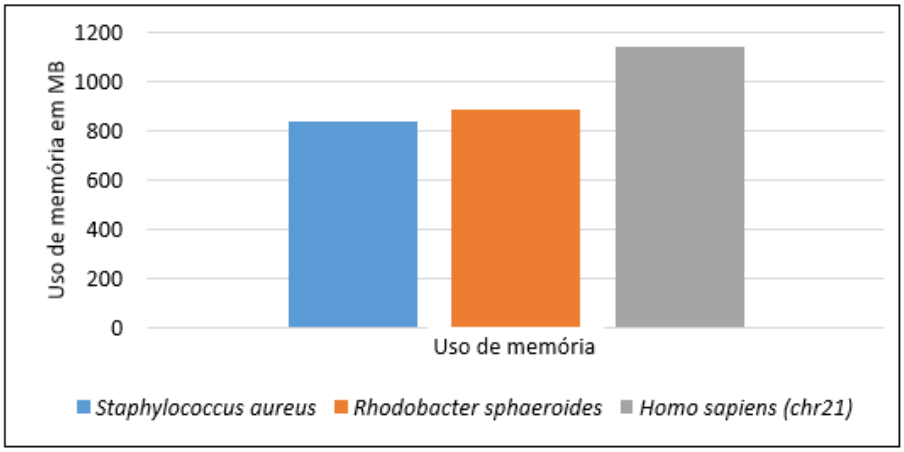

Figura 6.18: Média de utilização de memória da GPU

- **Movimentação de dados**

Outro ponto avaliado durante o processamento dos genomas foi a movimentação de dados entre o servidor e a GPU, que ocorre em ambos os sentidos, do servidor para a GPU e no sentido oposto, da GPU para o servidor.

Durante a execução do programa velveth não existe uma grande movimentação de dados entre o servidor e a GPU, porque este processo faz a leitura dos arquivos, quando a GPU é pouco usada.

Porém, a movimentação de dados na execução do programa velvetg é bem maior. Isso ocorre porque nesse processo é feita a construção do grafo *de Bruijn*, cuja

construção é delegada para a GPU. Na Figura 6.19 são apresentados os resultados médios da movimentação de dados entre o servidor e a GPU, durante a execução do programa velvetg.

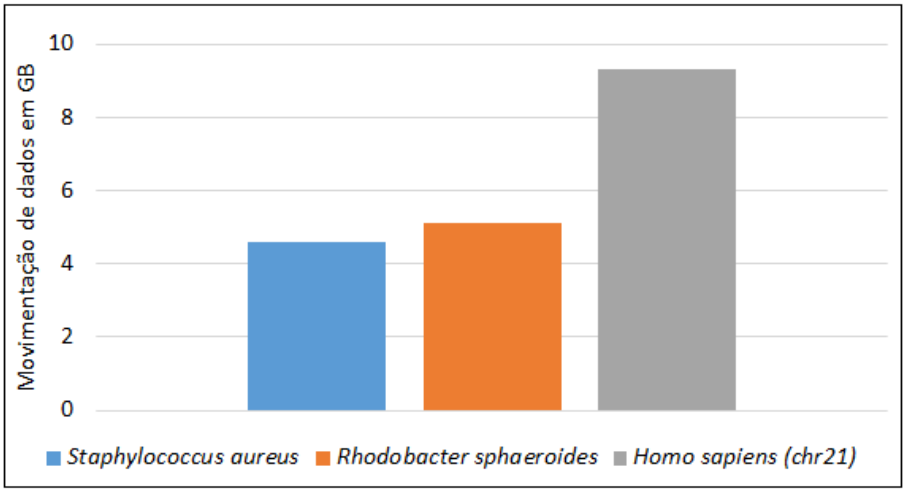

Figura 6.19: Média de movimentação de dados do servidor para a GPU durante a execução do programa velvetg

Verifica-se que, na Figura 6.20, em que é apresentada a média de movimentação de dados entre a GPU e o servidor, a quantidade de dados é maior. Isto ocorre pelo processamento das informações que são executadas na GPU e depois enviadas para o servidor.

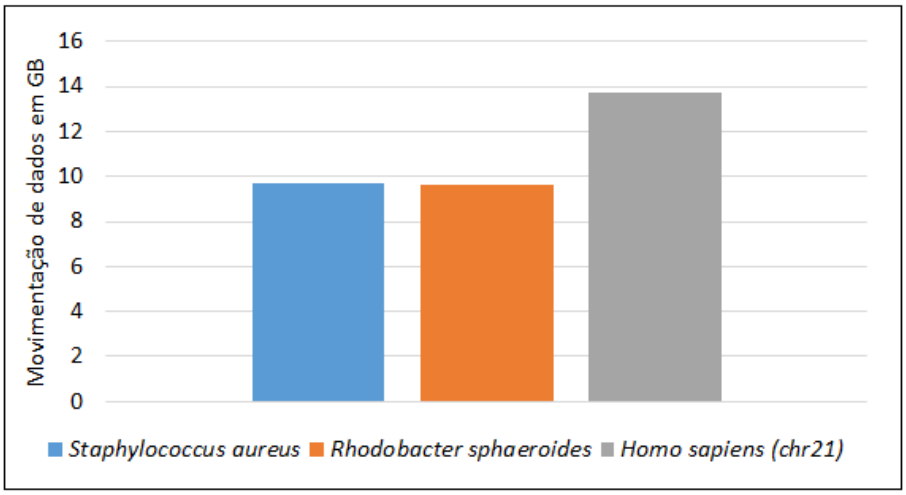

Figura 6.20: Média de movimentação de dados da GPU para o servidor

6.4 Comparação entre as versões OpenMP e OpenACC

6.4.1 Tempo de execução

O tempo de execução do velvetg com a nova versão do montador em OpenACC foram menores do que da versão com OpenMP. Usando os genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides*, o *speedup* obtido foi em média 5 vezes maior, e o genoma *Homo sapiens* alcançou um ganho médio de 3 vezes.

Tabela 6.5: Comparação de ganho dos tempos de execução da versão OpenACC em relação à versão OpenMP

Espécies	Tempo em segundos (s)		Speedup
	OpenMP	OpenACC	
Staphylococcus aureus	358	65	5,51
Rhodobacter sphaeroides	568	113	5,04
Homo sapiens (chr21)	7713	2279	3,38

Pode ser observado na Tabela 6.5 o ganho total obtido no processamento dos genomas.

6.4.2 Utilização de memória

Durante a execução do velveth usando a versão do montador com suporte a OpenMP, a utilização do recurso de memória foi menor, comparando com a versão em OpenACC. Esse comportamento foi o mesmo em todos os genomas utilizados nos testes.

Observa-se que na execução do processo velvetg o consumo de memória foi menor quando usado o genoma *Homo sapiens* na versão do montador em OpenACC, fato que pode estar associado ao tamanho do genoma.

A Figura 6.21 e Figura 6.22 mostram o consumo de memória durante a execução dos processos velveth e velvetg nas versões em OpenMP e OpenACC.

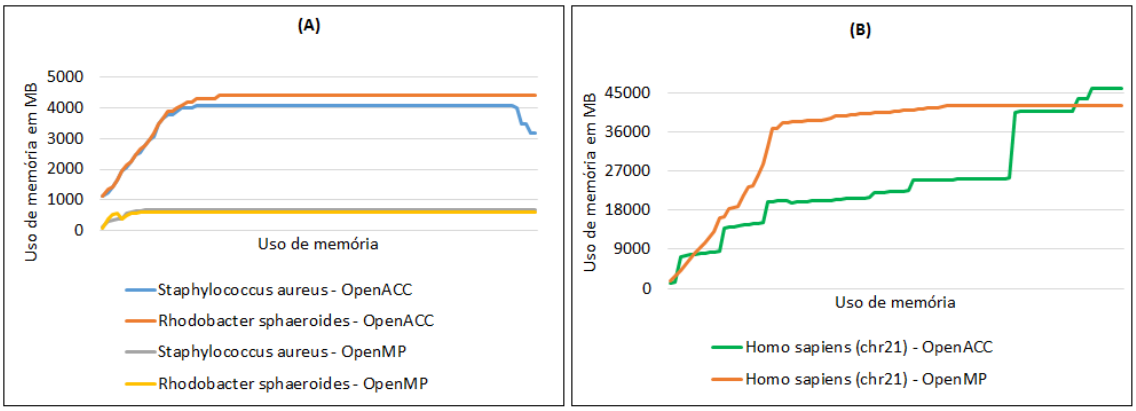

Figura 6.21: utilização de memória do velveth. (A) utilização de memória dos genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides*. (B) utilização de memória do genoma *Homo sapiens*

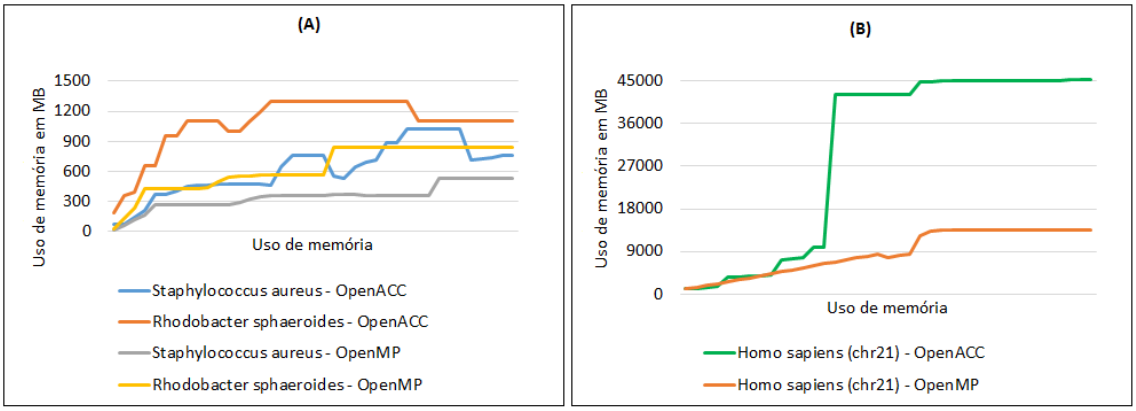

Figura 6.22: utilização de memória do velvetg. (A) utilização de memória dos genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides*. (B) utilização de memória do genoma *Homo sapiens*

6.4.3 Utilização de CPU

O uso de CPU na execução do processo velveth, na versão com OpenACC, teve um aumento em relação à versão em OpenMP, assim como também houve um aumento no uso de memória (Figura 6.23).

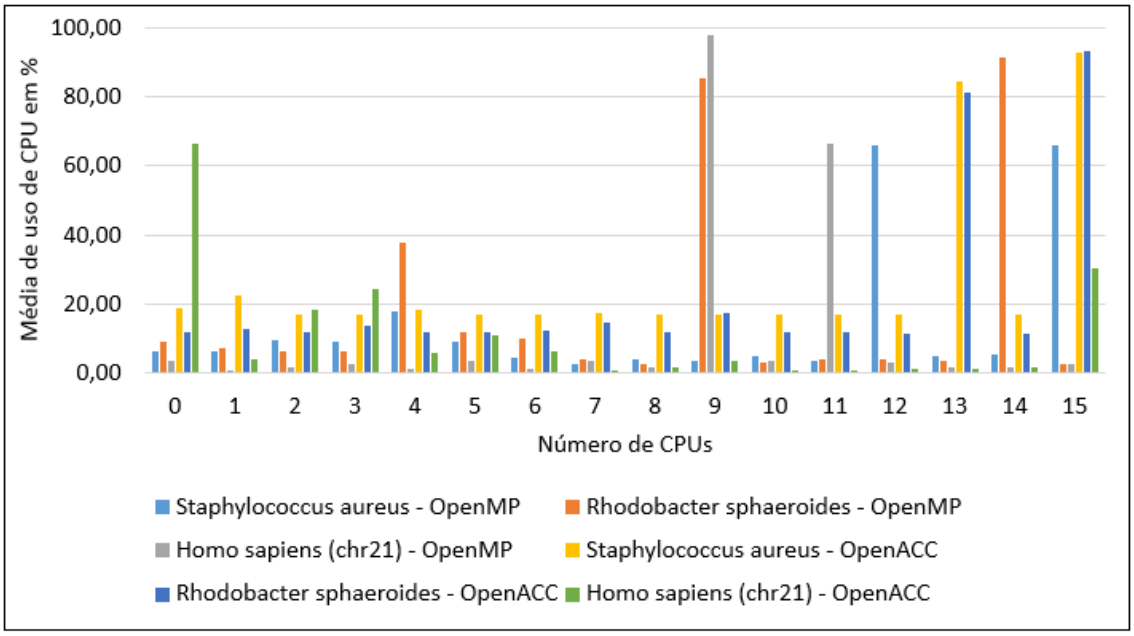

Figura 6.23: Média de utilização de CPU durante a leitura dos arquivos do genoma

Porém quando executado o processo velvetg usando OpenACC, o consumo de CPU foi menor, principalmente usando o cromossomo 21 de *Homo sapiens*, como visto na Figura 6.24.

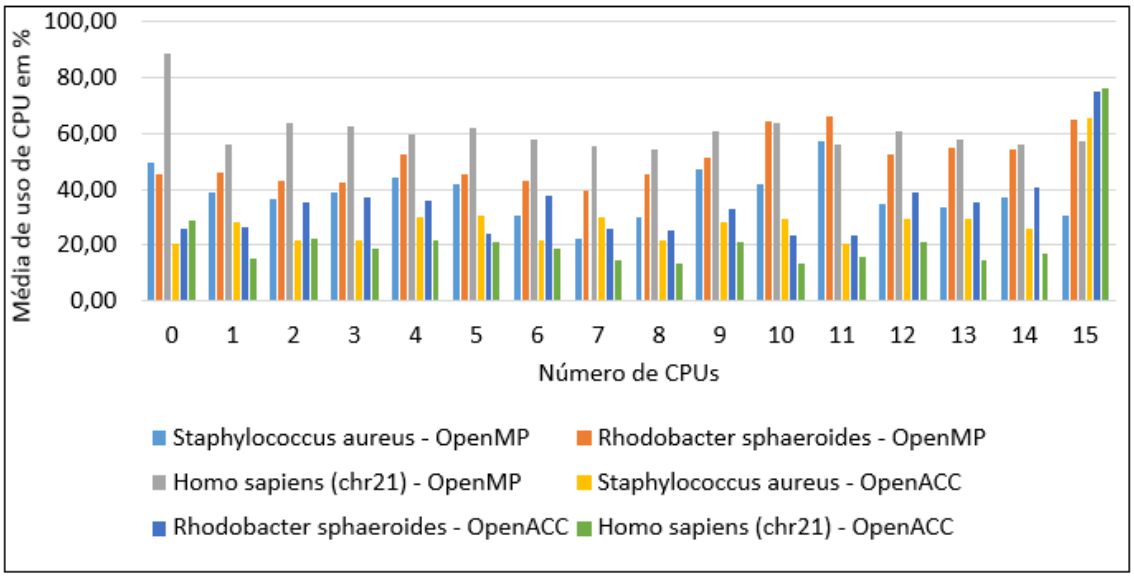

Figura 6.24: Média de utilização de CPU durante a montagem do genoma

7 CONCLUSÕES E TRABALHOS FUTUROS

Neste capítulo serão apresentados as conclusões, as contribuições e os trabalhos futuros.

7.1 Conclusões

Com o avanço nas novas técnicas utilizadas para o sequenciamento de DNA, houve um grande aumento na quantidade de dados gerados, surgindo assim a necessidade de desenvolver novos programas para tratar esses dados.

Por outro lado, com o desenvolvimento de dispositivos de alto poder de processamento, chamados de aceleradores (tais como *manycores* e GPUs), foi possível acumular um alto poder de processamento em dispositivos compactos e com baixo consumo de energia. Esses dispositivos têm entre suas principais características um alto grau de paralelismo, contendo vários núcleos de processadores simples e independentes, variando de algumas dezenas de núcleos a alguns milhares.

Existe um grande número de programas desenvolvidos para o uso em bioinformática, que em sua grande maioria são específicos para uma determinada tarefa, dentro do *pipeline* de montagem do DNA. Em alguns casos, se faz necessário o uso de vários programas para a execução de uma tarefa de montagem de DNA. O uso de vários programas é um processo trabalhoso que exige, em grande parte do tempo, conhecer em detalhes a utilização de cada um desses programas e como eles podem se integrar para gerar o resultado pretendido. Para resolver esse problema e fazer com que o máximo de tarefas fossem executadas de forma transparente para o usuá-

rio, surgiram os *frameworks*, que possibilitam agregar diversos programas em uma única interface de usuário.

Nesta tese foi desenvolvido o *framework* MELC Genomics 2.0, que além da facilidade de integração, tem como objetivo utilizar aceleradores para otimizar a execução de suas tarefas. Além de executar o processo de montagem com auxílio de GPU, existem módulos que fazem a verificação da qualidade dos dados e acompanham em tempo real a utilização de recursos computacionais.

Esta nova versão do MELC Genomics foi desenvolvida utilizando programação em PHP para web. Foram feitas mais de 900 linhas de programação distribuídos entre os códigos desenvolvido para cada módulo. Também foram criados mais de 50 *scripts* que coletam informações de acesso ao sistema, consumo de recurso computacional, criação de usuário entre outras. O MELC Genomics 2.0 foi desenvolvido em módulos. Cada módulo do sistema funciona de forma independente, isto é, pode ser executado de forma separada. O código fonte da nova versão está disponível publicamente no GitHub através do link: https://github.com/evaldocosta/melc.

Para realizar a tarefa de montagem, foi desenvolvida uma nova versão do montador Velvet usando o paradigma de programação OpenACC, que executa as partes mais críticas do montador na GPU. Após uma avaliação entre alguns montadores optou-se por utilizar o montador de Velvet, pois, além de ter apresentado bons resultados na montagem de genomas, trata-se de um programa livre, e seu código permite fazer a implementação utilizando modelo de programação para acelerador.

Uma comparação da versão anterior do Velvet, que usa o paradigma de programação OpenMP, com a nova versão em OpenACC, mostra que o ganho total durante todo o processo de montagem dos genomas chegou a ser de até 5 vezes mais rápido para os genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides*, e o

cromossomo 21 de *Homo sapiens* alcançou um ganho médio de 3 vezes. O código fonte da nova versão do Velvet em OpenACC está disponível no GitHub através do link: https://github.com/evaldocosta/velvetacc.

Os testes realizados para avaliação da nova versão em OpenACC, mostraram que o programa **velveth** não teve ganho significativo da versão em OpenACC sobre a versão em OpenMP, o que enseja a realização de maiores estudos sobre como otimizar a sua paralelização. Para a execução do programa **velvetg** houve grande diminuição do uso de CPU, mas também do tempo total de execução, mostrando a eficiência da transferência da computação da CPU para a GPU.

A movimentação de dados entre o servidor e a GPU durante a montagem do genoma foi maior na execução do programa **velvetg**, por conta da construção do grafo *de Bruijn* que este programa realiza. Na execução do **velveth** essa movimentação de dados é mínima, porque a tarefa de leitura dos arquivos é predominante neste programa. Em relação ao uso de memória e ocupação dos núcleos da GPU ambos programas apresentaram o mesmo comportamento, ou seja, quanto maior o genoma, mais recursos são utilizados.

Os resultados finais das avaliações realizadas mostram que o uso de diretivas do OpenACC na nova versão do montador Velvet foi eficiente na redução do tempo de execução da montagem dos genomas, diminuindo em até 5 vezes em relação a versão original em OpenMP, apresentando uma utilização eficaz dos recursos de GPU (memória, núcleos de processamento).

7.2 Trabalhos futuros

Atualmente a versão do MELC Genomics utiliza para o processamento dos dados uma única GPU. Em ambiente onde existam mais de uma GPU instalada, o sistema será executado na primeira GPU identificada pelo sistema operacional. Como trabalho futuro, pretendemos desenvolver uma nova versão que suporte o uso de várias GPUs para aumentar o desempenho do MELC Genomics.

O uso de Inteligencia Artificial (IA) está cada vez mais presente nas atividades do dia a dia e no desenvolvimento de aplicações para todas as áreas, assim como em diversos processos na bioinformática, através de modelos de Aprendizado de Máquina que são usados, por exemplo, na indústria farmacêutica para o desenvolvimento de novos medicamentos. Em um artigo recente Castro e colaboradores (CASTRO et al., 2021) fazem uma apresentação do uso de IA aplicada a bioinformática.

Com uma demanda crescente do uso de IA, o desenvolvimento de um novo módulo no MELC Genomics ou a alteração dos módulos existentes para trabalhar com IA no processamento dos dados, usando modelos de Aprendizado de Máquina existentes ou no desenvolvimento de novos modelos é uma proposta a ser considerada também em trabalhos futuros.

REFERÊNCIAS

ALTSCHUL, S. et al. Basic Local Aligment Search Tool. **Journal of molecular biology**, [S.l.], v.215, p.403–10, 11 1990.

A.M.D. **High-Bandwidth Memory (HBM)**: reinventing memory technology. [S.l.]: Inc, 2013.

ANDREWS, S. **FASTQC. A quality control tool for high throughput sequence data**. 2010.

BATLEY, J.; EDWARDS, D. Genome sequence data: management, storage, and visualization. **BioTechniques**, [S.l.], v.46, n.5, p.333–336, 2009. PMID: 19480628.

CASTRO, D. et al. **Inteligência Artificial aplicada à Bioinformática**. [S.l.: s.n.], 2021.

CHEN, S. **Introduction to OpenACC**. [S.l.]: Research Computing Services Information Services and Technology Boston University, 2017.

COCK, P. et al. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. **Nucleic Acids Research**, [S.l.], v.38, n.6, p.1767–1771, Dec. 2009.

CODREANU, V.; RODRíGUEZ, J.; SAASTAD, O. W. **Best Practice Guide - Knights Landing**. [S.l.]: Zenodo, 2017.

CORREA, J. C.; SILVA, G. P. Analysis and Performance Evaluation of Parallel BLAST. **I. J. Comput. Appl.**, [S.l.], v.20, n.2, p.112–122, 2012.

COSTA, E. B. MELC Genomics: a framework for de novo genome assembly. **Journal of Computational Biology**, [S.l.], v.25, n.2, p.194–199, 2018.

COSTA, E. B.; SILVA, G. P. Introdução à Programação com OpenACC. In: WSCAD 2019 - MINICURSOS (). **Anais...** [S.l.: s.n.], 2019.

COSTA, E. B.; SILVA, G. P.; TEIXEIRA, M. G. An Approach to Parallel Algorithms for Long DNA Sequences Alignment on Manycore Architecture. **Journal of Computational Biology**, [S.l.], v.27, n.8, p.1248–1252, 2020.

COSTA, E.; SILVA, G.; TEIXEIRA, M. DALIGNER Performance Evaluation on Intel Xeon Phi Architecture. In: INTERNATIONAL CONFERENCE ON BIOINFORMATICS AND COMPUTATIONAL BIOLOGY, 11. **Proceedings...** EasyChair, 2019. p.36–47. (EPiC Series in Computing, v.60).

EDWARDS, D.; HOLT, K. Beginner's guide to comparative bacterial genome analysis using next-generation sequence data. **Microbial informatics and experimentation**, [S.l.], v.3, p.2, 04 2013.

FLEISCHMANN, R. et al. Whole-Genome Random Sequencing and Assembly of Haemophilus Influenzae Rd. **Science (New York, N.Y.)**, [S.l.], v.269, p.496–512, 08 1995.

GHARIZADEH, B. et al. Typing of Human Papillomavirus by Pyrosequencing. **Laboratory investigation; a journal of technical methods and pathology**, [S.l.], v.81, p.673–9, 06 2001.

GIARDINE, B. et al. Galaxy: a platform for interactive large-scale genome analysis. **Genome Research**, [S.l.], v.15, n.10, p.1451–1455, 2005.

GRIFFITHS, A. J. F. **An introduction to genetic analysis**. 8th ed..ed. New York: Freeman, 2005.

GUREVICH, A. et al. QUAST: quality assessment tool for genome assemblies. **Bioinformatics**, [S.l.], v.29, n.8, p.1072–1075, 02 2013.

HARRIS, M. **Unified Memory for CUDA Beginners**. [S.l.]: NVIDIA Corporation, 2017.

HOGEWEG, P. The Roots of Bioinformatics in Theoretical Biology. **PLoS Computational Biology**, [S.l.], v.7, n.3, 2011.

KCHOUK, M.; GIBRAT, J.-F.; ELLOUMI, M. Generations of Sequencing Technologies: from first to next generation. **Biology and Medicine**, [S.l.], v.09, 01 2017.

KHAN, A. R. et al. A Comprehensive Study of De Novo Genome Assemblers: current challenges and future prospective. **Evolutionary Bioinformatics**, [S.l.], v.14, p.117693431875865, 02 2018.

LEMOS, M.; CASANOVA, M. **Algoritmos para Análises de Sequências**. [S.l.: s.n.], 2000. 01-10p.

LIU, Y. et al. SWAPHI-LS: smith-waterman algorithm on xeon phi coprocessors for long dna sequences. In: IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING (CLUSTER), 2014. **Anais...** [S.l.: s.n.], 2014. p.257–265.

LUSHBOUGH, C. et al. BioExtract Server—An Integrated Workflow-Enabling System to Access and Analyze Heterogeneous, Distributed Biomolecular Data. **IEEE/ACM Transactions on Computational Biology and Bioinformatics**, [S.l.], v.7, n.1, p.12–24, Jan 2010.

MERRIMAN, B. et al. Progress in Ion Torrent semiconductor chip based sequencing. Electrophoresis. **Electrophoresis**, [S.l.], v.33, p.3397–417, 12 2012.

MILLER, J. R.; KOREN, S.; SUTTON, G. Assembly algorithms for next-generation sequencing data. **Genomics**, [S.l.], v.95, n.6, p.315–327, 2010.

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**, [S.l.], v.48, n.3, p.443 – 453, 1970.

NVIDIA. **NVIDIA's Next Generation CUDA Compute Architecture**: kepler tm gk110/210. [S.l.]: NVIDIA Corporation, 2014.

NVIDIA. **Tesla K80 GPU Accelerator**. [S.l.]: NVIDIA Corporation, 2015.

O. SANDES, E. F. de et al. CUDAlign 3.0: parallel biological sequence comparison in large gpu clusters. In: CCGRID. **Anais...** IEEE Computer Society, 2014. p.160–169.

OLIVEIRA, J. Azambuja de et al. Sequenciamento de DNA Mitocondrial para Avaliação de diferenças genéticas em ovinos (Ovis aries). **Journal of the Selva Andina Animal Science**, [S.l.], v.1, p.11–20, 01 2014.

PARKS, M.; LISTON, A.; CRONN, R. Meeting the challenges of non-referenced genome assembly from short-read sequence data. **Acta Horticulturae**, [S.l.], v.859, p.1–10, 04 2010.

PEARSON, W. Finding Protein and Nucleotide Similarities with FASTA. **Current protocols in bioinformatics / editoral board, Andreas D. Baxevanis ... [et al.]**, [S.l.], v.Chapter 3, p.Unit3.9, 03 2004.

PEREZ, C. **The Deep Learning Guide**: for amd radeon instinct gpus. [S.l.]: CreateSpace Independent Publishing Platform, 2018.

PEVZNER, P.; TANG, H.; WATERMAN, M. An Eulerian path approach to DNA fragment assembly. **Proceedings of the National Academy of Sciences of the United States of America**, [S.l.], v.98, p.9748–53, 09 2001.

RAHMAN, R. Intel Xeon Phi Coprocessor Architecture and Tools: the guide for application developers. In: APRESS. **Anais...** [S.l.: s.n.], 2013. v.1.

RHOADS, A.; AU, K. PacBio Sequencing and Its Applications. **Genomics, proteomics and bioinformatics**, [S.l.], v.13, 11 2015.

RONAGHI, M. Pyrosequencing sheds light on DNA sequencing. **Genome Res**, [S.l.], v.11, n.1, p.3–11, Jan. 2001.

SALZBERG, S. et al. GAGE: a critical evaluation of genome assemblies and assembly algorithms. **Genome research**, [S.l.], v.22, p.557–67, 12 2011.

SANGER, F.; COULSON, A. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. **Journal of Molecular Biology**, [S.l.], v.94, n.3, p.441 – 448, 1975.

SEARLS, D. B. The Roots of Bioinformatics. **PLOS Computational Biology**, [S.l.], v.6, n.6, p.e1000809+, June 2010.

SILVA, G. P. **Programação Paralela com MPI Um Curso Introdutorio**. [S.l.]: Amazon, 2018.

SMITH, T. F.; WATERMAN, M. S. Identification of common molecular subsequences. In: JOURNAL OF MOLECULAR BIOLOGY. **Anais...** [S.l.: s.n.], 1981. v.147(1), p.195–197.

SPENGLER, S. Bioinformatics in the Information Age. **Science**, [S.l.], v.287, p.1221–1223, 02 2000.

STEPHENS, Z. D. et al. Big Data: astronomical or genomical? **PLOS Biology**, [S.l.], v.13, n.7, p.1–11, 07 2015.

TASMA, I. et al. Utilization of Genome Sequencing Technology to Accelerate Plant Breeding Program. **J. Litbang Pert.**, [S.l.], v.1, p.159–168, 10 2015.

WONG, K.-C. et al. DNA Sequencing Technologies: sequencing data protocols and bioinformatics tools. **ACM Computing Surveys**, [S.l.], v.52, p.1–30, 09 2019.

ZERBINO, D.; BIRNEY, E. Velvet: algorithms for de novo short read assembly using de bruijn graphs. **Genome research**, [S.l.], v.18, p.821–9, 06 2008.

ZHU, M. et al. Performance Evaluation and Optimization of HBM-Enabled GPU for Data-Intensive Applications. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, [S.l.], v.PP, p.1–10, 02 2018.