FELIPE VALENCIA DE ALMEIDA

Ferramenta para correção de sequências genômicas em sistemas de memória compartilhada e FPGA

São Paulo
2018

FELIPE VALENCIA DE ALMEIDA

Ferramenta para correção de sequências genômicas em sistemas de memória compartilhada e FPGA

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção de Título de Bacharel em Engenharia .

São Paulo
2018

FELIPE VALENCIA DE ALMEIDA

Ferramenta para correção de sequências genômicas em sistemas de memória compartilhada e FPGA

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção de Título de Bacharel em Engenharia .

Área de Concentração:
Engenharia de Computação

Orientador:
Profa. Dra. Liria Matsumoto Sato

Co-orientador:
Prof. Dr. Edson Toshimi Midorikawa

São Paulo
2018

Nome: ALMEIDA, Felipe Valencia de

Título: Ferramenta para correção de sequências genômicas em sistemas de memória compartilhada e FPGA

Monografia (Trabalho de Conclusão de Curso) apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do Título de Bacharel em Engenharia na área de Engenharia de Computação.

Trabalho entregue para o departamento em: 05/12/2018

Responsáveis

Profa. Dra. Liria Matsumoto Sato
(Orientadora)

Prof. Dr. Edson Toshimi Midorikawa
(Co-orientador)

Felipe Valencia de Almeida (Orientado)

A meus pais

AGRADECIMENTOS

À professora Liria, ao professor Edson e à professora Tatiana por me guiarem nesta jornada em uma área multidisciplinar.

Aos meus pais que sempre me deram o apoio necessário para seguir adiante.

À Erina, que me acompanhou por todos os momentos de alegria e de tristeza durante a graduação.

À tia Fatima e ao tio Daniel, pelos momentos que passamos juntos.

À Nathalia, que me mostrou como fazer, e não fazer, um pcr.

À Ana Maria pela ajuda com as normas técnicas necessárias.

A todos os meus professores, do colégio e da faculdade, que contribuíram para a minha formação.

Aos funcionários do PCS, que muito fizeram por mim nestes 5 anos.

"We all must have an ideal to govern our conduct and insure contentment, but it is immaterial whether it be one of creed, art, science or anything else, so long as it fulfills the function of a dematerializing force."

-- Nikola Tesla

RESUMO

A montagem de genoma ou montagem de DNA é um processo importante para o entendimento dos seres vivos. Através dele é possível identificar anomalias e doenças, fornecendo aos pesquisadores subsídios para seus tratamentos. Desde o início do século XXI, houve rápidas evoluções nesta área, visando otimizar este processo que possui grande custo computacional.

Este projeto de formatura teve como objetivo a elaboração de uma ferramenta de montagem, através da correção, de sequências genômicas da tecnologia PacBio, em um ambiente de memória compartilhada com FPGA.

Dado o cenário atual do estado da arte nessa área, existe uma necessidade de aprimoramento dos algoritmos já existentes, visando a uma melhoria tanto na qualidade dos dados pós correção quanto no tempo necessário para realizar essa operação. Estabelecida inicialmente uma estratégia de correção com programação paralela e a sua implementação utilizando o OpenMP, foi estabelecido um roteiro de implementação na placa FPGA, obtendo um resultado satisfatório com base nas métricas de desempenho estabelecidas nessa área.

Palavras-Chave – Programação paralela. FPGA. Sequenciamento de genoma. Bioinformática.

ABSTRACT

The assembly of the genome or assembly of DNA is an important process for the understanding of living beings. Through that, it is possible to identify anomalies and diseases, providing researchers subsidies for their treatments. Since the beginning of the 21st century, there have been rapid developments in this area, aiming to optimize this process that has great computational cost.

This graduation project targets at the development of a tool for assembly, through a hybrid correction, of genetic sequences of PacBio technology, in a shared memory environment with FPGA.

Regarding the current state of the art scenario in this area, there is a need to improve existing algorithms, aiming to enhance the quality of the data after correction and the time required to perform this operation, as well. Establishing initially a correction strategy with parallel programming using OpenMP, it was established an implementation roadmap with the FPGAs, obtaining a satisfactory result based on the performance metrics achieved in this area.

Keywords – Parallel programming. FPGA. Genome sequencing. Bioinformatics.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API - *Application Program Interface*

ASCII - *American Standard Code for Information Interchange*

BLAST - *Basic Local Alignment Search Tool*

DNA - *Deoxyribonucleic acid*

FPGA - *Field Programmable Gate Array*

HDL - *Hardware Description Language*

HPS - *Hard Processor System*

IEEE - *Institute of Electrical and Electronics Engineers*

NCBI - *National Center for Biotechnology Information*

PacBio - *Pacific Biosciences*

PCI - *Peripheral Component Interconnect*

RNA - *Ribonucleic acid*

UART - *Universal Asynchronous Receiver-Transmitter*

VHDL - *VHSIC Hardware Description Language*

VHSIC - *Very-High-Speed Integrated Circuit*

SUMÁRIO

1 INTRODUÇÃO

Com o passar dos anos, a programação paralela ou programação de alto desempenho tem assumido papel importante no cenário da computação mundial. O advento dos computadores com múltiplos núcleos (*multicores*) permitiu ao programador contribuir para o ganho progressivo de desempenho na computação, que antes era possível apenas com o aumento na densidade de transistores.

A programação de alto desempenho está presente em diversas áreas de conhecimento, como Astrologia, Física, Química, Climatologia e Biologia, sendo esta última o escopo deste projeto.

Com o término do projeto de sequenciamento do genoma humano em 2003, tem ocorrido um avanço nas técnicas de sequenciamento de genomas, diminuindo o custo e aumentando a qualidade das amostras (GOODWIN; MCPHERSON; MCCOMBIE, 2016). Contudo, devido à grande existência de mutações sequenciais do DNA ainda é difícil realizar um mapeamento preciso nos genomas dos seres vivos.

Por isso, uma nova geração de sequenciadores está surgindo, com o propósito de reduzir as incertezas existentes no processo de análise do ácido desoxirribonucleico (DNA), e com isso, novos métodos estão sendo desenvolvidos com esse propósito (MISALE et al., 2014). Um desses métodos é uma melhoria do método PacBio, da empresa *Pacific Biosciences*, e será descrito posteriormente.

1.1 Motivação

A área de pesquisa genética possui um longo crescimento nas últimas décadas. O DNA humano possui cerca de 3 bilhões de bases nitrogenadas (VENTER et al., 2001), levando cerca de 9 meses para ser montado em 2001 (VENTER et al., 2001), 2 meses em 2007 (WHEELER et al., 2008) e atualmente são necessários poucos dias com um *cluster* adequado. Um grande desafio dessa área de pesquisa é a diferença obtida no processo

de montagem em diferentes seres vivos de uma mesma espécie. James Watson realizou o processo de montagem do seu DNA em 2007, e foi encontrada uma diferença de 1,4% entre o seu DNA e o DNA de referência obtido no projeto do sequenciamento do genoma humano (WHEELER et al., 2008).

Existe então duas vertentes de pesquisa da área. A primeira delas voltada para a comparação de diferentes seres vivos de diferentes espécies com base nas variações genéticas. Com isso, é possível localizar ancestrais comuns, melhorando assim as classificações taxonômicas já existentes.

A segunda é a comparação entre diferentes seres vivos de uma mesma espécie, com enfoque no ser humano. A necessidade de entender o funcionamento do ser humano tem movido grandes esforços por parte da comunidade internacional desde o projeto do genoma humano. Um novo projeto denominado Projeto 1000 Genomas foi lançado em 2008 e finalizado em 2015 com o intuito de aumentar a base de referências dos genomas humanos, fortalecendo a pesquisa que relaciona as ligações entre o genótipo e o fenótipo humano (CONSORTIUM et al., 2010). Seu legado foi fornecer uma série de informações e dados abertos para a comunidade científica. Uma destas informações é que os seres humanos possuem uma variação aproximada de 20 milhões de bases nitrogenadas em seu DNA, ou seja, possuem aproximadamente 0,6% de diferença em seu material genético (CONSORTIUM et al., 2015).

Até mesmo gêmeos monozigóticos (gerados por um mesmo espermatozoide e óvulo) possuem diferenças em seu DNA, consequência de diferentes processos de cópia e mutação do DNA (BRUDER et al., 2008).

E são nessas diferenças que se encontra o segredo para o entendimento e cura das doenças. Para tal, é necessária a comparação entre diversos genomas de diferentes seres, visando isolar as regiões distintas do DNA e entender seus impactos no fenótipo do ser vivo.

Esforços já estão sendo realizados com esse propósito. O número de variantes de genoma humano de referências vem crescendo nos últimos anos. Porém, existe uma demanda de se reduzir o custo dos recursos para se realizar as montagens dos DNAs.

Com a evolução dos sequenciadores de DNA, o paradigma utilizado na montagem de sequências genômicas vem se modificando ao longo dos anos. Com isso, existe uma necessidade constante de se repensar as ferramentas computacionais utilizadas neste processo, visando aprimorar as técnicas já existentes.

1.2 Objetivo

O objetivo deste trabalho é propor estratégias para implementar uma ferramenta para montagem de sequências genômicas através da correção, com a finalidade de melhorar a qualidade do genoma obtido e obter um bom desempenho de processamento. Um algoritmo para alinhar e corrigir sequências genômicas, no processo de montagem de genomas, em um computador e uma placa *Field-Programmable Gate Array* (FPGA) é proposto. Sendo a FPGA aplicada como uma aceleradora de algoritmos, espera-se melhorar o tempo de execução necessário para realizar essa etapa do processo de montagem de sequências genômicas.

1.3 Justificativa

Dentre as etapas necessárias para promover a montagem de sequências genômicas, o alinhamento é a que demanda maior esforço computacional, exigindo uma quantidade de tempo da ordem de centenas de milhares de CPUs hora, a depender do organismo em questão (ANGEL et al., 2018). Observando-se o estado da arte, existe um *tradeoff* entre o tempo de execução necessário para se alinhar todas as sequências necessárias e a qualidade dos dados gerados no processo de montagem. O método utilizado no momento consiste em abdicar parte da qualidade dos dados, com o intuito de tornar viável o tempo de execução.

1.4 Materiais e Métodos

Para o desenvolvimento do projeto inicialmente foi realizado um estudo da área de atuação, baseado na leitura de uma bibliografia básica com o intuito de assimilar os conceitos preliminares necessários. Após tal leitura básica foi realizado um estudo do estado da arte, para se estabelecer um levantamento das ferramentas já existentes e observar as deficiências de cada uma destas.

Passada a etapa inicial de estudo, foi elaborada uma sequência de etapas necessárias a serem realizadas para a ferramenta cumprir os requisitos funcionais e não funcionais do projeto. Então as etapas foram implementadas inicialmente em software de maneira sequencial. Com a validação da implementação sequencial, as etapas foram implementadas utilizando-se a programação paralela.

A estratégia de montagem aplicando correção foi avaliada pela implementação em *software*, incluindo uma análise de desempenho. O processo de correção propriamente dito foi implementado na placa FPGA. Uma análise de desempenho com resultados teóricos previstos para uma implementação utilizando a CPU em conjunto com a FPGA foi realizada.

A seguir é apresentada uma visão em tópicos do cronograma do projeto:

1. Estudo dos conceitos biológicos;
2. Estudo do estado da arte (ferramentas de montagem);
3. Definição do conjunto de etapas para a correção;
4. Implementação das etapas em software de maneira sequencial;
5. Validação dos resultados obtidos em software sequencial;
6. Implementação das etapas em software com programação paralela;
7. Validação dos resultados obtidos em software paralelo;
8. Implementação da correção em FPGA;
9. Levantamento dos resultados teóricos da FPGA;
10. Comparação dos resultados obtidos e validação da ferramenta.

As tecnologias utilizadas neste projeto se dividem em software e hardware. Na parte do software, foi utilizada a linguagem C, devido ao seu alto desempenho, e o OpenMP (CHAPMAN; JOST; PAS, 2007), que é amplamente utilizado nos projetos de programação paralela para ambientes de memória compartilhada. Na parte do hardware foi utilizada uma placa FPGA e o VHDL como ferramenta modeladora de hardware na placa.

1.5 Organização do Trabalho

Este trabalho está organizado de forma a inicialmente explicar os aspectos conceituais necessários para seu entendimento, depois apresentar uma visão geral da ferramenta e por último seu desenvolvimento em si.

Dessa forma, o capítulo 2 apresenta um levantamento bibliográfico sobre os aspectos conceituais importantes neste trabalho, sendo divididos em aspectos biológicos e computacionais.

O capitulo 3 resume os trabalhos relacionados, ou seja, as ferramentas já existentes no estado da arte.

O capitulo 4 apresenta a ferramenta, elencando seus requisitos funcionais e não funcionais e seu conjunto de etapas.

O capitulo 6 apresenta o processo de implementação da ferramenta.

O capitulo 7 define o conjuntos de testes realizados com o intuito de validar a ferramenta desenvolvida.

O capitulo 8 apresenta algumas considerações finais feitas sobre o projeto.

2 ASPECTOS CONCEITUAIS

O capítulo apresenta alguns dos aspectos conceituais mais importantes para inserir o leitor no contexto em que o projeto se encontra.

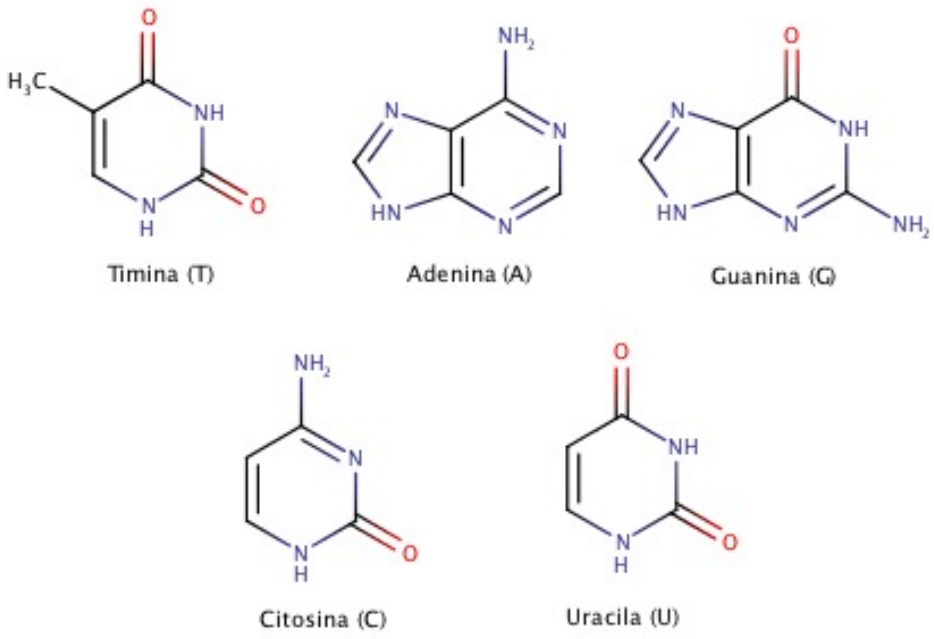
2.1 Aspectos Biológicos

Esta seção é dedicada à apresentação de conceitos fundamentais da genética para o entendimento deste projeto. Para tal, ela apresenta uma série de conceitos e terminologias utilizadas na área de uma maneira incremental.

2.1.1 Bases Nitrogenadas

Bases nitrogenadas são compostos orgânicos cíclicos que compõem o DNA e o ácido ribonucleico (RNA) dos seres vivos. Elas são divididas em 5 tipos, sendo eles: Adenina (A), Guanina (G), Citosina (C), Timina (T) e Uracila (U), sendo estas duas últimas encontradas apenas no DNA e no RNA respectivamente. A figura 1 apresenta a estrutura molecular das bases.

Figura 1: Estrutura Molecular das Bases Nitrogenadas



Fonte: Autor

No RNA do tipo mensageiro, uma trinca de bases nitrogenadas gera um códon e um códon é responsável por gerar um aminoácido do ser vivo. Esses aminoácidos são ligados gerando as proteínas, que atuam como um guia para o funcionamento do ser.

2.1.2 Cromossomos

Cromossomos são moléculas de DNA que compõem parte do material genético do organismo. As células do organismo podem ser divididas em haploides ou diploides, baseadas no número de cromossomos de cada tipo existente em seus núcleos. Os cromossomos são divididos em cromossomos sexuais, que são responsáveis por definir o sexo do ser vivo e os autossomos, que são responsáveis por definir as demais características do ser.

O número de cromossomos varia com a complexidade do organismo. Um organismo mais simples como a *Escherichia coli* (E. coli) possui apenas 1 cromossomo, enquanto o ser humano possui 46 cromossomos (44 autossomos + 2 cromossomos sexuais).

2.1.3 Genoma

O genoma pode ser definido como todo o material genético do organismo, ou seja, o conjunto de todo o DNA (ou RNA no caso dos vírus) que um organismo possui. Ele é

dividido em cromossomos, que por sua vez são compostos por bases nitrogenadas.

A importância do genoma está no fato dele conter toda informação do organismo, ou seja, ele funciona como um manual para o entendimento da vida. Através de sua análise, é possível identificar causas de doenças, e por consequência seu tratamento, mutações, prever comportamentos dentre outras coisas.

2.1.4 Sequenciadores

Um sequenciador, apresentado pela figura 2, é um aparelho que recebe como entrada uma amostra de material genético e tem como saída um ou mais arquivos texto em diversas extensões, sendo as mais comuns *fasta* ou *fastq*. Os arquivos texto contêm conjuntos de sequências de bases nitrogenadas e as qualidades de cada base. Nesse caso, a qualidade funciona como indicador da acurácia daquela base nitrogenada, ou seja, o grau de confiança em que a base foi sequenciada corretamente pelo sequenciador.

Figura 2: Sequenciador de DNA

Fonte: Illumina (s.d.)

Estes arquivos são fornecidos como entrada para uma ferramenta de montagem de genoma, responsável por agrupar estas sequências em *contigs* (conjunto de fragmentos de DNA sobrepostos, resultantes de um processo de montagem). O objetivo do algoritmo nesse caso, além de agrupar e distinguir grupos semelhantes com mutações, é gerar menor número possível de *contigs* no final de sua execução. Assim, é possível promover uma montagem precisa do material genético do ser vivo.

Em um cenário hipotético onde o processo de sequenciamento fosse perfeito, e não houvesse mutações no DNA, o número de *contigs* gerado na montagem deveria ser igual ao número de cromossomos do organismo. Na prática, o número gerado acaba sendo igual ou superior ao número de cromossomos.

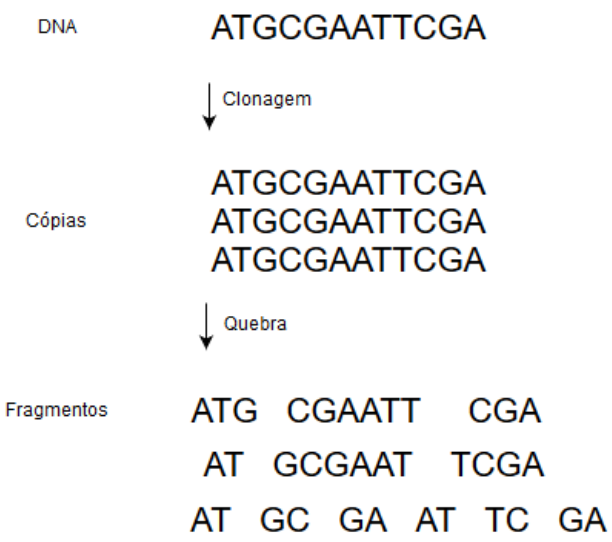
A evolução nos sequenciadores impulsiona a evolução das ferramentas de montagem, pois a forma com que o material genético é sequenciado e transformado nos arquivos texto sofre mudanças em cada geração de sequenciadores (ZHANG et al., 2011).

2.1.5 Sequenciamento *Shotgun*

O sequenciamento *shotgun* é o processo de sequenciamento de genoma que envolve a quebra do DNA em regiões aleatórias. Este processo possui maior emprego ao contrário da quebra sequencial do DNA devido ao fato que seu custo é inferior (BROWN, 2002).

Para realizá-lo, inicialmente o material genético deve ser copiado através de um processo bioquímico, gerando redundância do mesmo, então, o sequenciador realiza a quebra aleatória, gerando fragmentos denominados *reads*. As *reads* serão as sequências presentes no arquivo de saída do sequenciador. Esse processo é ilustrado pela figura 3.

Figura 3: Sequenciamento *shotgun*



Fonte: Autor

A redundância de material genético necessária para realizar o sequenciamento *shotgun* é chamada de cobertura (*coverage*) do genoma. A cobertura é representada por um multiplicador seguido de um número representando quantas vezes o material genético presente naquele arquivo é maior que o genoma do organismo. Por exemplo, um arquivo que tenha um multiplicador x20 possui 20 vezes mais material genético que o genoma do ser vivo.

Uma alta cobertura é um fator necessário para promover o correto procedimento de

montagem, pois quanto maior a cobertura maior a garantia que todas as bases nitrogenadas aparecem mais vezes nos fragmentos. Porém, caso a cobertura já seja elevada, um aumento pode apenas prejudicar o processo (SIMS et al., 2014).

2.1.6 Arquivos para Tratamento de Genomas

Há inúmeras maneiras de se trabalhar com genomas, devido a uma falta de padronização. Logo, diferentes formas de se armazenar as informações relativas ao genes foram criadas e vêm sendo utilizadas ao longo dos anos. Neste projeto serão utilizados apenas os arquivos com extensão .fasta e .fastq, cujo modelo será descrito a seguir.

2.1.6.1 Arquivo FASTA

O formato FASTA surgiu a partir de um algoritmo com o mesmo nome utilizado para localizar similaridade entre proteínas e sequências de DNA (NCBI, 2013), o que resultou em uma extensão com seu nome (*.fasta*). Ele consiste em um conjunto de sequências genômicas precedidas por um cabeçalho. Cada cabeçalho é iniciado pelo caractere ">"como identificador. Abaixo é apresentado um exemplo de arquivo fasta.

```
1 >seq1
2 AATTCGAGAG
3 >seq2
4 AAAAAAAAAA
5 >seq3
6 TCGATGGACG
```

Uma deficiência do arquivo fasta frente ao cenário atual é a necessidade de se representar cada base nitrogenada juntamente com a sua qualidade. Utilizando o fasta são necessários dois arquivos para conseguir esta representação. Para tal, um arquivo necessita conter o cabeçalho das sequências juntamente com as sequências e o outro arquivo conter o cabeçalho das qualidades juntamente com as qualidades.

Essa deficiência serviu de motivação para o desenvolvimento de outro formato de arquivo utilizado aqui, que é o FASTQ (*fasta quality*)

2.1.6.2 Arquivo FASTQ

O formato FASTQ surgiu com o intuito de permitir, em um mesmo arquivo, a representação tanto da base nitrogenada quanto de sua respectiva qualidade. Para isso ele se utiliza de dois identificadores, sendo o "@"para o cabeçalho das sequências e o "+"para o cabeçalho das qualidades. Abaixo é apresentado um exemplo de arquivo fastq.

```
@seq1
AATTCGAGAG
+
""##!!#"%(
@seq2
AAAAAAAAAA
+
""#'-$-',%
@seq3
TCGATGGACG
+
"!#"%(-',%
```

Observa-se então que o formato FASTQ conseguiu suprir uma carência do formato FASTA. Porém, novamente devido a uma falta de padronização, surge um problema na representação das qualidades (COCK et al., 2009).

Diferentes formas de codificação trabalham com diferentes faixas de ASCII (American Standard Code for Information Interchange) para representar suas qualidades, sendo necessário inicialmente identificar o tipo de codificação utilizada no arquivo antes de realizar qualquer operação com suas qualidades. A tabela 1 apresenta as diferentes formas de codificação utilizadas em arquivos fastq.

Tabela 1: Formatos de codificação FASTQ

Codificação	**Amplitude**	**ASCII Inicial**	**ASCII Final**
Sanger	0 - 40	! (33)	I (73)
Solexa	-5 - 40	; (59)	h (104)
Illumina 1.3+	0 - 40	@ (64)	h (104)
Illumina 1.5+	3 - 41	C (67)	i (105)
Illumina 1.8+	0 - 41	! (33)	J (74)

Fonte: Autor

A amplitude de valores definidas para as qualidades em cada tipo de codificação é definida como a pontuação da qualidade (*Q-score*). A relação entre o *Q-score* e a probabilidade do sequenciamento de determinada base ter sofrido algum erro é dada pela seguinte equação:

$$Q = -10 * log_{10}(P(erro)) \tag{2.1}$$

Realizando algumas substituições na equação, a tabela 2 apresenta algumas taxas de erro com base no valor do (*Q-score*).

Tabela 2: Relação entre *Q-score* e probabilidade de erro no sequenciamento

Q-score	**P(erro)**
10	0.1
20	0.01
30	0.001
40	0.0001

Fonte: Autor

Não existe uma regra prática para se definir a partir de que ponto do *Q-score* pode-se inferir que a base não foi sequenciada corretamente. Esse parâmetro deve ser flexível nas ferramentas de correção, pois irá influenciar diretamente nos resultados obtidos.

2.1.7 Alinhamento

Desde as primeiras tentativas de montagem de genoma, um dos principais desafios encontrados foi obter a similaridade entre duas sequências de bases nitrogenadas em relação a suas bases em comum. Esse tipo de alinhamento entre duas ou mais sequências teve como uma de suas primeiras implementações práticas e com ampla aplicação o algoritmo de Needleman-Wunsch (NEEDLEMAN; WUNSCH, 1970). Esse algoritmo consiste em se utilizar programação dinâmica, montando-se uma tabela baseada nas diferentes possibilidades de alinhamento entre as bases. Através de uma pontuação destas possibilidades de alinhamento (bases iguais, bases distintas ou base e lacuna) é possível se obter o alinhamento com maior similaridade, que é o com maior pontuação.

Logo após, surgiu o *Basic Local Alignment Search Tool* (BLAST), atualmente mantido pelo *National Center for Biotechnology Information* (NCBI) (BLAST, 2018). A

ferramenta possui diferentes implementações, para diferentes propósitos (cálculo de similaridade, localização de padrões, obtenção de frequências de padrões, dentre outros) e seu código é utilizado em algumas outras ferramentas de bioinformática. Essa utilização é consequência das constantes atualizações recebidas, visando otimizar seu algoritmo e agilizar o processo de alinhamento.

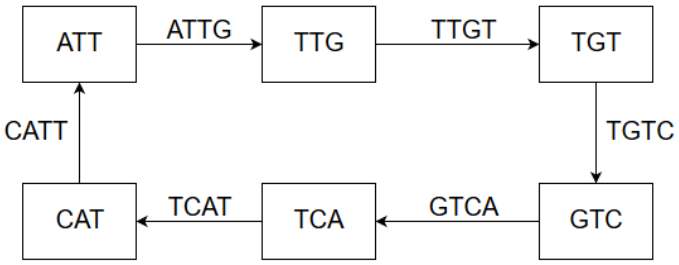
2.1.8 Evolução das Ferramentas de Montagem de Genoma

Nessa subseção será descrito um panorama histórico acerca dos avanços e modificações nas ferramentas para montagem de genoma. Optou-se por separar as ferramentas em três gerações, sendo elas a geração atual (Pós PacBio), a geração PacBio e a geração Illumina.

2.1.8.1 Montagem Illumina

O método de montagem Illumina consiste em montar grafos De Brujin (COMPEAU; PEVZNER; TESLER, 2011), conforme ilustrado pela figura 4 para arquivos de entrada compostos por sequências pequenas de 100 a 250 bases nitrogenadas em média (devido à tecnologia utilizada) (ILLUMINA, 2010). As sequências são quebradas em fragmentos menores, denominados *k-mers* para realizar a montagem do grafo.

Figura 4: Utilização de grafo De Brujin para montagem de genoma



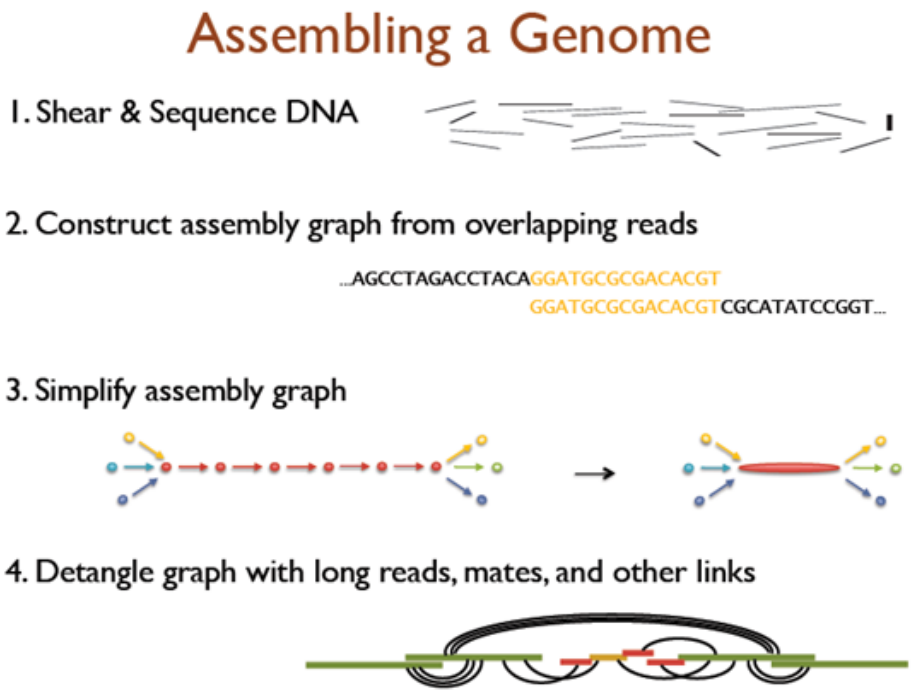
Fonte: Autor

Esse método não é adequado para seres vivos com grande quantidade de genoma, pois o custo computacional para se realizar as diversas comparações e montar os grafos com as sequências pequenas torna-se inviável (FLOWERS; CREWS, 2018). Além disso, sequências de tamanho pequeno podem gerar ambiguidade no processo de montagem, resultando em genomas errados.

2.1.8.2 Montagem PacBio

O método de montagem PacBio, ilustrado pela figura 5, trabalha com sequências maiores que as Illumina, também devido à tecnologia utilizada (10000 bases nitrogenadas em média) (RHOADS; AU, 2015). Tal montagem também se utiliza de grafos, sendo que as arestas dos grafos têm ponderação, onde o peso é a similaridade apenas entre as extremidades de duas sequências. Através do caminho de maior peso no grafo, é possível obter os *contigs*. Um problema deste método é a baixa qualidade dos arquivos de entrada (composto pelas sequências grandes), isso pode levar a montagens errôneas (RHOADS; AU, 2015).

Figura 5: Processo de sequenciamento de genoma



Fonte: National Human Genome Research Institute (s.d.)

2.1.8.3 Montagem pós PacBio

Visando melhorar o processo de montagem PacBio, a geração atual busca realizar um processo de correção para melhor a acurácia das bases provenientes do sequenciamento com PacBio. Essa correção pode ser do tipo auto (*self*) ou do tipo híbrida (*hybrid*).

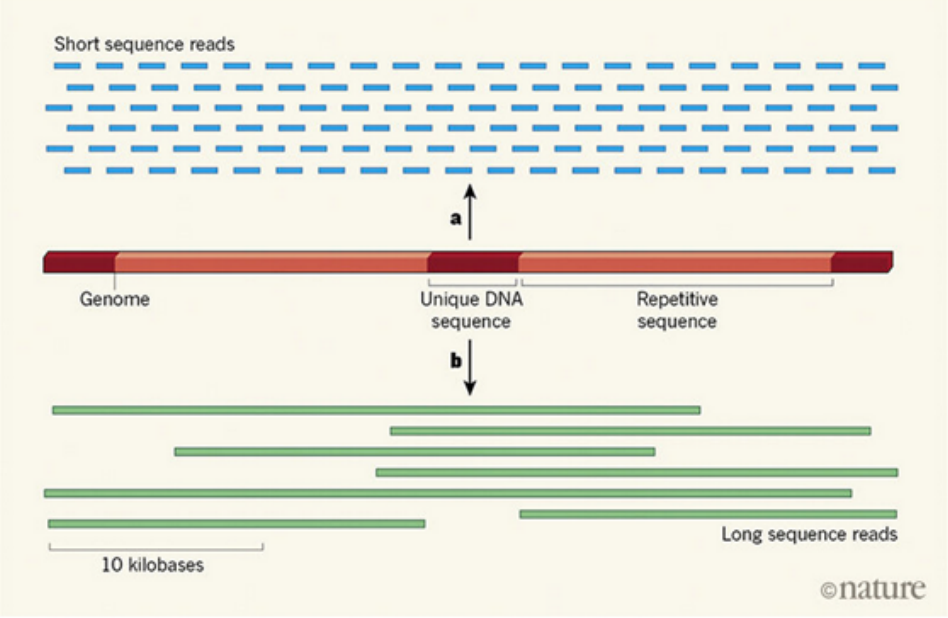
A correção do tipo auto utiliza apenas o próprio arquivo com as sequências PacBio para diminuir a taxa de erros. Ela parte do princípio de que como existe uma alta cobertura do genoma, fragmentos provenientes de uma mesma região do DNA irão se

repetir no arquivo. Logo, é possível buscar um consenso entre as diferentes sequências da mesma região. Esse consenso então seria o real valor da base, ou seja, uma base cuja qualidade seria alta o bastante para ser confiável.

Esse tipo de correção tem como ponto positivo precisar apenas do arquivo com sequências do tipo PacBio, que é o mesmo que será utilizado na montagem. Porém, devido à alta taxa de erro das bases e ao fato que a cobertura não garante igual repetição em todas as regiões do DNA, ele é suscetível a falhas. Além disso, também é necessário que a cobertura do arquivo seja superior à necessária apenas para o processo de montagem, caso contrário, não é possível obter o consenso no alinhamento.

A correção híbrida utiliza dois arquivos de entrada, onde um é composto por sequências pequenas e com alta qualidade (em geral Illumina) e outro com sequências PacBio. Tal correção utiliza-se da premissa de que como ambos os arquivos provenientes de diferentes sequenciadores são de um mesmo material genético, as sequências presentes neles devem ser referentes às mesmas regiões do DNA, conforme ilustrado pela figura 6.

Figura 6: Processo de correção de sequências



Fonte: (FLOWERS; CREWS, 2018)

Assim, os atuais métodos de montagem de genomas possuem em geral os seguintes passos:

1. Correção das sequências de bases nitrogenadas.
2. Alinhamento das sequências.

3. Montagem dos grafos de alinhamento.

4. Escolha dos caminhos de maior peso nos grafos e montagem dos *contigs*.

Nesse contexto, existem ferramentas que realizam apenas a correção, apenas a montagem, a montagem com auto correção e a montagem com correção híbrida. Elas serão apresentadas no capítulo 3.

2.2 Aspectos Computacionais

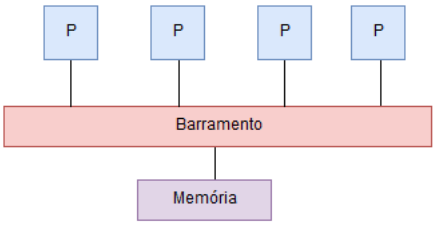
2.2.1 Arquiteturas Paralelas

Desde o início do século XXI, projetistas de processadores têm enfrentado dificuldades para promover ganhos significativos entre as gerações de processadores (PACHECO, 2011). O fenômeno conhecido como *powerwall* marcou um divisor de águas na história do desenvolvimento de processadores, fazendo com que os projetistas tivessem que mudar o paradigma de desenvolvimento. Tendo relação direta com a velocidade de execução de instruções e, por conseguinte, o desempenho, o aumento da frequência do *clock* nas máquinas era o principal método de fornecer ganhos significativos para as gerações de processadores. A técnica consistia em introduzir um número maior de transistores nos circuitos integrados a cada geração. Isso foi possível até o momento em que a potência gerada pelo chaveamento dos transistores começou a afetar a integridade dos transistores, impulsionando a criação das arquiteturas paralelas.

Existem duas grandes classes de arquiteturas para a programação paralela. São elas os sistemas de memória compartilhada e os sistemas de memória distribuída (PACHECO, 2011).

Nos sistemas de memória compartilhada, ilustrados pela figura 7, todos os núcleos (*cores*) do computador são conectados em um mesmo barramento de memória. Dessa forma, as variáveis utilizadas pelos programas podem ser compartilhadas entre todas as unidades de processamento, facilitando a comunicação entre elas. Um exemplo de *framework* é o OpenMP, que permite a programação em C neste tipo de sistema.

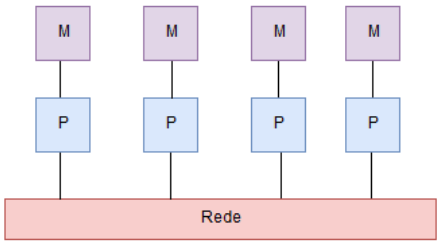
Figura 7: Arquitetura do sistema de memória compartilhada



Fonte: Autor

Nos sistemas de memória distribuída, ilustrados pela figura 8, cada núcleo é chamado de nó, e possui sua própria memória, isolando assim as variáveis utilizadas pelos programas. A comunicação entre os nós é possível com o auxílio de protocolos de troca de mensagens. Um exemplo de *framework* é o OpenMPI.

Figura 8: Arquitetura do sistema de memória distribuída

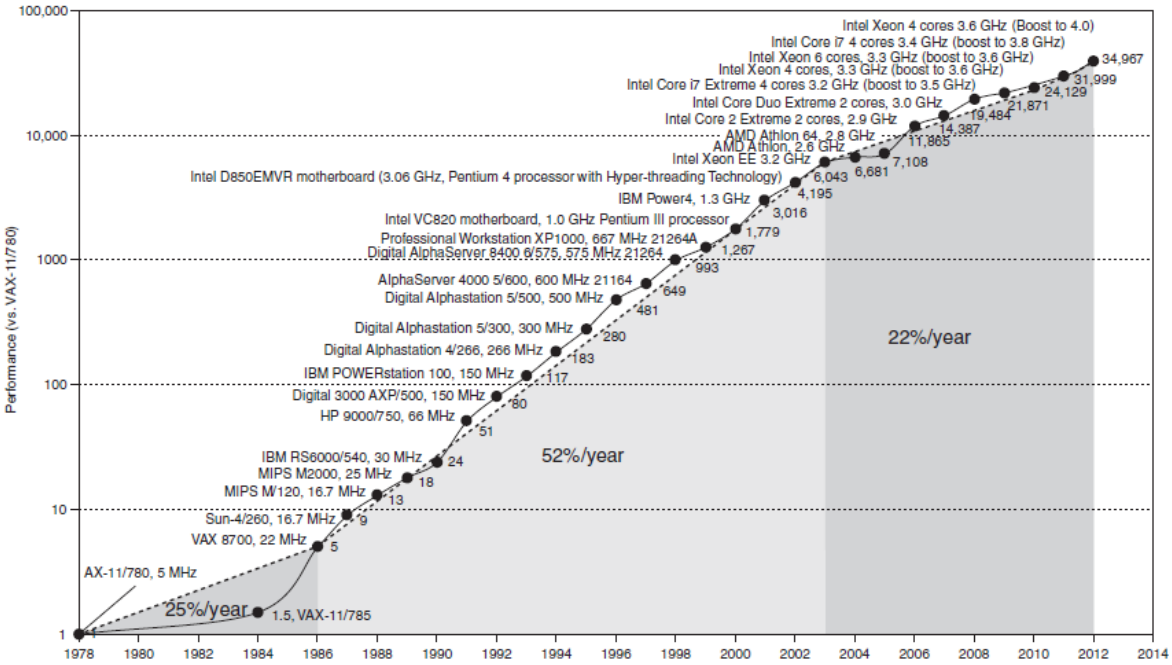


Fonte: Autor

Como exemplo de sistemas de memória compartilhada temos os *desktops* e *notebooks*, enquanto os sistemas de memória distribuídas são mais robustos e utilizados nos supercomputadores e *clusters*. Alguns exemplos de supercomputadores podem ser observados na página do projeto TOP500 (STROHMAIER et al., 2018).

A figura 9 apresenta o ganho histórico dos processadores. Observa-se que o ganho anual da arquitetura paralela é inferior ao antigo ganho proveniente do aumento do *clock*, sendo responsabilidade do desenvolvedor de software paralelizar suas aplicações para melhor aproveitar os recursos do processador *multicore*.

Figura 9: Evolução do ganho dos processadores



Fonte: (PATTERSON; HENNESSY, 2013)

2.2.2 Métricas de Desempenho do Paralelismo

Dentre as métricas de desempenho utilizadas para avaliar o impacto do paralelismo na execução de um programa, destacam-se o *speedup* e a eficiência.

O *speedup* (S) é a razão entre o tempo de processamento sequencial e o tempo de processamento paralelo. Sua função é permitir a análise do ganho obtido utilizando múltiplos processadores. Existem três casos de *speedup* obtidos através da aplicação do paralelismo. São eles:

- *Speedup* normal: Nesse caso a adição de mais processadores não promove uma melhoria proporcional no tempo de execução. Ele é o caso comum, onde o paralelismo não é explorado plenamente.

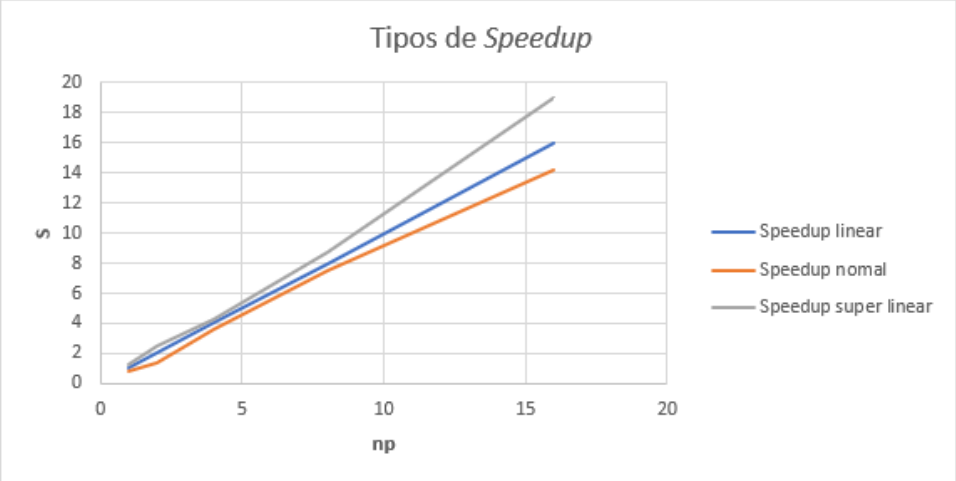
- *Speedup* linear: Aqui a adição de mais processadores promove uma melhoria proporcional no tempo de execução. É o caso almejado, onde o paralelismo é explorado plenamente, porém não é possível se obter em todas as situações devido a atrasos como o *overhead* de comunicação.

- *Speedup* super-linear: Nesse caso a adição de mais processadores promove uma melhoria maior que a proporção esperada no tempo de execução. Esse caso é raro, pois

sua justificativa se encontra em aspectos internos do processador, como na melhor utilização de caches de memória, resultado em um ganho em operações de leitura e escrita de dados.

A figura 10 ilustra graficamente os tipos de *speedup*.

Figura 10: Tipos de *speedup*



Fonte: Autor

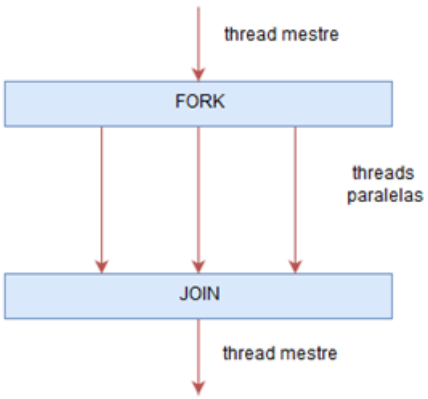
A eficiência (E) é a razão entre o *speedup* adquirido com a execução paralela e o número de processadores utilizados. Sua função é explicitar o quanto de paralelismo é explorado no programa.

2.2.3 OpenMP

OpenMP é uma Application Program Interface (API) desenvolvida com o intuito de facilitar a programação em ambientes de memória compartilhada. Ela é composta por um conjunto de diretivas de compilação (*pragmas*), construídas para habilitar o paralelismo em nível de *threads* nas linguagens C, C++ e Fortran (CHAPMAN; JOST; PAS, 2007).

Para tal, é aplicado o modelo *fork-join*, onde uma *thread* principal ou *thread* mestre gera um conjunto de *threads* paralelas através de um processo de *fork*. Estas *threads* então são executadas em paralelo, até acontecer o processo de *join*, conforme ilustrado pela figura 11.

Figura 11: Modelo *fork-join*



Fonte: Autor

2.2.4 FPGA

FPGAs são circuitos integrados programáveis, desenvolvidos com o propósito de atender as necessidades do usuário através de configuração variável. Elas utilizam linguagens de descrição de hardware (*HDL - Hardware Description Languages)* ou diagramas esquemáticos para alterar a configuração interna dos blocos lógicos. Seu emprego é adequado para aplicações de alto desempenho, como processamento digital de sinais, processamento de imagens, roteadores/switches de alto desempenho, bioinformática, dentre outras.

Para a geração do código executável da placa, é necessário utilizar um ambiente de desenvolvimento específico do fabricante, capaz de transformar os comandos escritos na HDL em uma arquitetura lógica específica da placa FPGA em questão. Dentre as duas maiores fabricantes de placas FPGA do mercado, utiliza-se o Quartus para placas da Intel (ex Altera) ou o Vivado para placas da Xilinx.

Um dos principais limitadores na sua utilização é a quantidade restrita de memória disponível dentro do circuito integrado (células lógicas), sendo necessária a utilização de uma arquitetura adequada ou de comunicação com uma memória externa para grandes projetos. Essa comunicação com um *slot* de memória externa é possível através do barramento que interliga a memória com as células lógicas. Para isso, utiliza-se um processador lógico existente dentro da placa FPGA (Nios II da Intel ou MicroBlaze da Xilinx), capaz de abstrair detalhes baixo nível da comunicação entre os elementos.

Com a aquisição da Altera pela Intel, o processo de integração entre FPGA e CPU vem se intensificando. A nova linha de processadores Xeon lançada em 2018 realiza um

processamento híbrido entre o Xeon e a Arria 10, prometendo ganhos significativos de desempenho (HUFFSTETLER, 2018). Desta forma, estima-se que a utilização das placas FPGA no cenário da computação de alto desempenho heterogênea (diferentes recursos de processamento) tende a aumentar ao longo dos próximos anos.

2.2.5 VHDL

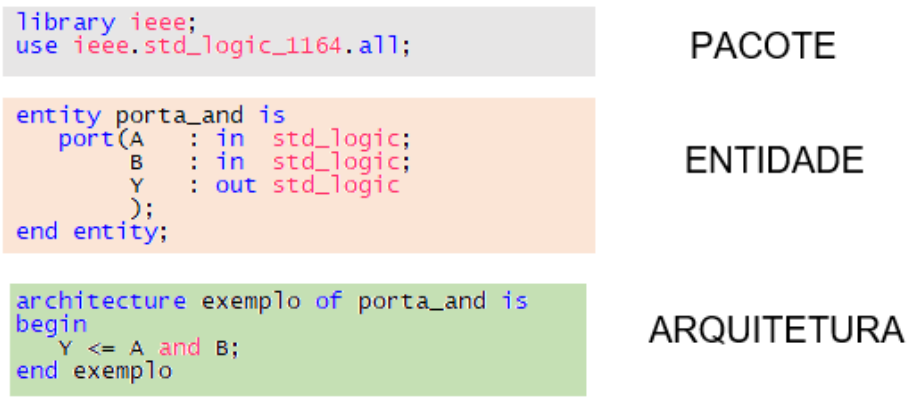
VHSIC Hardware Description Language ou *Very High Speed Integrated Circuits Hardware Description Language* (VHDL) é uma *Hardware Description Language* (HDL) desenvolvida pelo departamento de defesa dos Estados Unidos na década de 1980. No inicio sua aplicação era limitada a documentação de componentes vendidos às Forças Armadas, porém, após sua padronização pelo *Institute of Electrical and Electronics Engineers* (IEEE) em 1987 houve um aumento da sua aplicação para outras finalidades.

A estrutura básica do VHDL divide um componente em três partes, sendo elas:

- Biblioteca/pacote (*library/package*): Conjunto de constantes, tipos de variáveis e funções que poderão ser utilizadas pelo componente.
- Entidade (*entity*): Visão caixa preta do componente digital, onde são descritas apenas as entradas e as saídas dele.
- Arquitetura (*architecture*): Descrição interna da arquitetura do componente. Nesta parte é definida a lógica digital necessária para obter as saídas a partir das entradas do componente.

A figura 12 apresenta a descrição de um componente digital em VHDL.

Figura 12: Estrutura de um componente digital em VHDL



Fonte: Autor

2.2.6 Comunicação entre FPGA e CPU

Esta subseção descreve algumas possíveis formas de realizar a comunicação entre a FPGA e a CPU, visando à elaboração de um ambiente híbrido, semelhante ao que foi projetado na nova linha Xeon da Intel.

2.2.6.1 Comunicação Serial (RS-232)

A comunicação serial utilizando o protocolo RS-232 é um dos modos mais simples de ser realizado, porém, possui baixa taxa de transmissão, com valores máximos próximos a 1MB/s dependendo do tamanho do cabo. Esse tipo de comunicação consegue enviar um caractere ASCII (7 ou 8 bits) por vez, sendo necessário também enviar 1 bit de início (*start bit*), 1 bit de término (*stop bit*) e podendo enviar também um bit de paridade, para verificar se o envio foi feito corretamente.

O *overhead* do projeto de comunicação neste modo é baixo. Para sua implementação, é necessário utilizar um terminal serial pelo lado da CPU, enquanto uma *Universal Asynchronous Receiver/Transmitter* (UART) deve ser projetada na placa FPGA para enviar e receber os dados serialmente. Deve-se ter o devido cuidado com a diferença de tensões entre a FPGA e a CPU, para não danificar nenhum dos circuitos internos, sendo necessária a utilização de circuitos integrados conversores de tensão. Por ser uma comunicação do tipo assíncrona, o controle deve ser implementado pelo projetista.

Um ponto importante de destaque ao se utilizar esse modo é que ambos os *clocks* da CPU e da FPGA são superiores à taxa de transmissão, ou seja, o limitante aqui na transmissão dos dados é a própria taxa do protocolo.

2.2.6.2 PCI Express

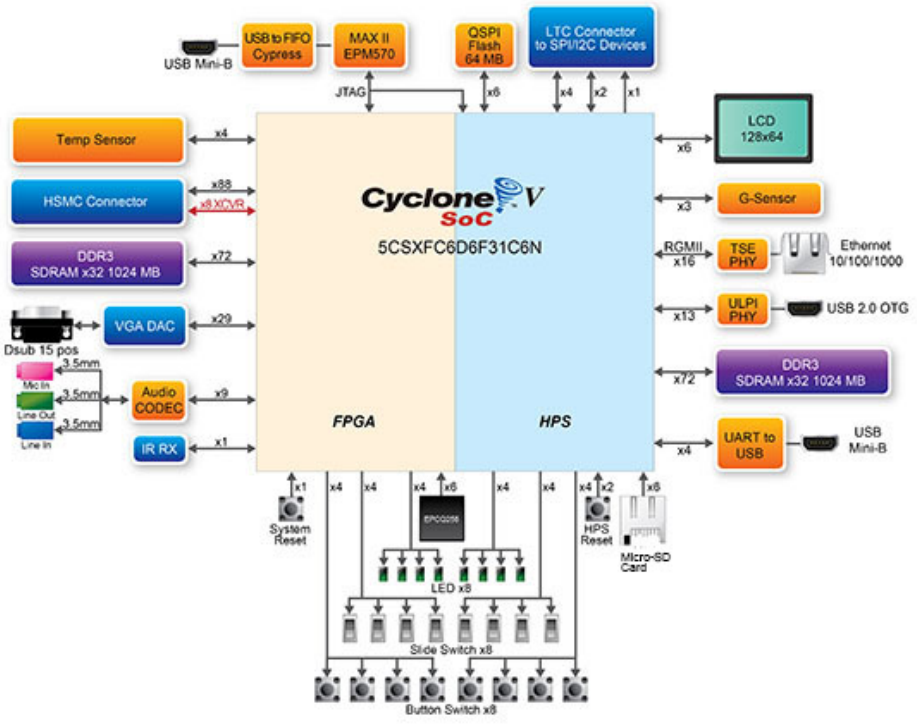
A comunicação via *Peripheral Component Interconnect Express* (PCI Express) utiliza o conector PCI para ligar a FPGA na CPU. Esse tipo de conector suporta taxas da ordem de GB/s, porém, devido ao clock da FPGA ser inferior à taxa de transmissão (em média 50MHz ou 100MHz em FPGAs convencionais), ele se torna o limitante para a comunicação.

O *overhead* do projeto de comunicação neste modo é médio, onde é necessário projetar um módulo para fazer a comunicação tanto pelo lado da FPGA quanto pela CPU.

2.2.6.3 Comunicação via *Bridge*

A comunicação via ponte (*bridge*) é possível nos sistemas híbridos, onde em uma mesma placa ou chip existe uma FPGA e uma CPU, também conhecido aqui como *Hard Processor System* (HPS). Tal comunicação consiste em mapear endereços de memória do sistema na *bridge*, de forma que tanto a FPGA quanto a CPU possam acessar o endereço. A figura 13 ilustra uma placa híbrida FPGA e ARM.

Figura 13: Terasic SoC Kit



Fonte: Terasic (s.d.)

Essa comunicação possui alta velocidade, podendo chegar a taxas de transmissão da ordem de GB/s, porém, é a que possui maior *overhead* de programação tanto pelo lado do software quanto hardware modelado para transmitir os dados. Neste tipo de comunicação, assim como no PCI Express, o limitante da comunicação é o *clock* da FPGA, que costuma ter frequência inferior à taxa de transmissão.

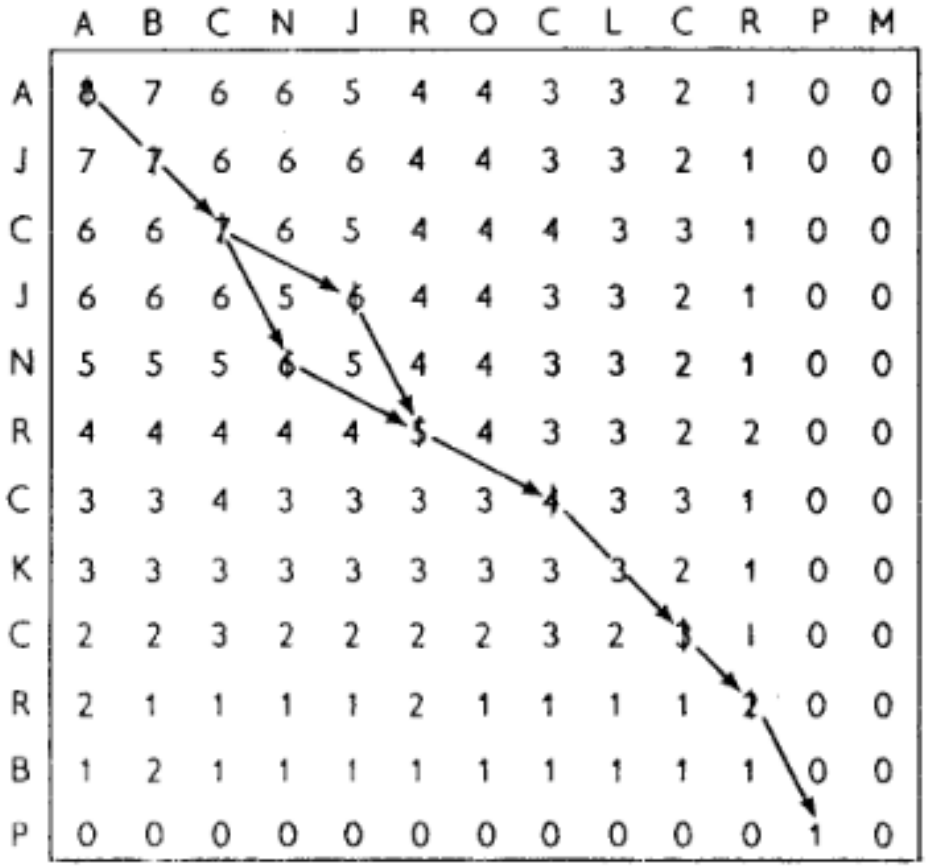
2.2.7 Técnicas de Alinhamento

Desde meados do século XIX, pesquisadores dedicam-se ao problema do alinhamento de sequências genômicas. Um dos primeiros algoritmos de alinhamento desenvolvidos e com ampla aplicação foi o algoritmo de Needleman–Wunsch sendo uma das primeiras

aplicações da computação dinâmica para comparar sequências (SETUBAL; MEIDANIS, 1997).

Esse algoritmo recebe duas sequências genômicas, podendo elas ser apresentadas em nível de base nitrogenada ou aminoácido, e encontra o alinhamento ideal entre ambas, ou seja, o alinhamento que possui maior similaridade. Para tal, uma malha é construída com as possíveis combinações das bases nitrogenadas das sequências. Seguindo o caminho com a maior pontuação na malha é possível obter o alinhamento ideal das sequências. Esse processo é ilustrado pela figura 14.

Figura 14: Algoritmo de Needleman–Wunsch



Fonte: (NEEDLEMAN; WUNSCH, 1970)

3 TRABALHOS RELACIONADOS

O capítulo apresenta algumas ferramentas mais utilizadas pela comunidade científica para promover a correção e/ou montagem das sequências genômicas.

3.1 Trimmomatic

Trimmomatic (BOLGER; LOHSE; USADEL, 2014) é uma ferramenta para melhorar a qualidade geral do arquivo de sequências Illumina. Mesmo as sequências possuindo boa qualidade, o sequenciamento Illumina ainda é passível de erros, podendo gerar bases e/ou fragmentos com baixa qualidade.

Ela recebe como entrada um arquivo fastq contendo um conjunto de sequências Illumina juntamente com uma série de parâmetros opcionais para auxiliar o processo de corte das regiões de pior qualidade. Sua saída é também um arquivo fastq porém apenas com sequências de boa qualidade. Um ponto negativo no aspecto computacional desta ferramenta é que enquanto o arquivo original possui sequências de mesmo tamanho devido à tecnologia utilizada, o arquivo de saída possui sequências de tamanho variável, por consequência do corte.

3.2 SPAdes

SPAdes (BANKEVICH et al., 2012) é uma ferramenta para montagem de sequências Illumina. Essa ferramenta é a oficial da Illumina, sendo utilizada amplamente em diversos projetos envolvendo a montagem desse tipo de sequências. Atualizações têm sido realizadas desde sua criação. A versão atual é a 3.9.0.

O SPAdes realiza a montagem através da quebra das sequências em k-mers e montagem dos grafos De Brujin, levando um tempo maior do que as ferramentas que realizam a montagem PacBio, porém, sem a necessidade de correção.

3.3 Canu

Canu (KOREN et al., 2017) é uma ferramenta para montagem de sequências PacBio que realiza correção do tipo auto. Seu processo de montagem é dividido em três etapas.

1. Correção - Na etapa de correção um arquivo fastq é utilizado como entrada, e um arquivo fasta é gerado, ou seja, o arquivo de saída não possui mais as qualidades das bases. Nela, as sequências são alinhadas integralmente, visando a obtenção do consenso, por isso é a etapa com maior custo computacional.

2. Corte - Na etapa de corte são verificadas as sequências que são destoantes das demais, ou seja, não são semelhantes a nenhuma outra. Essas sequências são retiradas do arquivo, de forma que a entrada é um arquivo fasta e a saída também é um arquivo fasta, porém de menor tamanho. Esta é a etapa com menor custo computacional.

3. Montagem - Na etapa de montagem é realizada a montagem do genoma, conforme apresentado no capítulo 2. Sua entrada é um arquivo fasta e a saída é o genoma montado do organismo. Esta etapa possui custo computacional intermediário entre as etapas anteriores.

O Canu permite que cada etapa seja executada individualmente, de forma que é possível utilizá-lo apenas para a montagem, complementando a ferramenta desenvolvida neste projeto.

3.4 Marvel

O Marvel (GROHME et al., 2018), (NOWOSHILOW et al., 2018) é uma ferramenta para montagem de sequências PacBio que realiza correção do tipo híbrida. Ele foi desenvolvido em 2018, e sua documentação para uso ainda está incompleta, portanto, ele não foi utilizado comparativamente neste trabalho.

3.5 Smartdenovo

Smartdenovo (RUAN,) é uma ferramenta para montagem de sequências PacBio que não realiza correção. Sua entrada é um arquivo fasta já corrigido, caso contrário, o

genoma resultante da montagem terá grande diferença do genoma de referência ou então a montagem não será feita.

A ferramenta utiliza um processo otimizado para a montagem, onde as sequências são quebradas em k-mers, e os k-mers são comparados com uma tabela *hash* para verificar a similaridade e realizar a montagem.

3.6 LorDEC

LorDEC (SALMELA; RIVALS, 2014) é uma ferramenta para correção de sequências PacBio. Para realizar a correção, a ferramenta utiliza as sequências Illumina montadas em um grafo De Brujin, onde o grafo é montado através da comparação entre as sequências e o genoma de referência do organismo.

A ferramenta apresenta desempenho superior quando comparado às outras que possuem o mesmo propósito (SALMELA; RIVALS, 2014). Isso é consequência do fato que ela utiliza o genoma de referência e apenas uma parte do arquivo de sequência Illumina, onde é necessária uma cobertura aproximada de 50x.

4 FERRAMENTA DE MONTAGEM DE GENOMA

O capítulo apresenta a ferramenta projetada e desenvolvida durante o projeto de formatura. Inicialmente foram levantados e especificados os requisitos do sistema, para depois ocorrer a definição de sua arquitetura.

4.1 Especificação de Requisitos

Foram levantados os seguintes requisitos para o sistema, divididos em funcionais e não funcionais.

Requisitos Funcionais:

- Manusear conjuntos de amostras de dados (fragmentos de genoma)

  Os arquivos texto utilizados são compostos por uma série de sequências que devem ser devidamente organizadas e alinhadas, para que o processo seja adequado.

- Atender às métricas de alinhamento

  As métricas estabelecidas de alinhamento com sequências de referência devem ser seguidas, para ser possível estabelecer comparações adequadas com as ferramentas já existentes.

- Formato de entrada e saída semelhante com os utilizados por outras ferramentas

  As entradas e saídas devem ser semelhantes para o processo de validação da ferramenta.

- Apresentação de resultados intermediários importantes para pesquisadores da área

  Durante o processo de correção, algumas informações podem ser extraídas a respeito de como o sequenciamento foi feito. Essas informações devem ser armazenadas em um *log* para poderem ser utilizadas por pesquisadores da área.

Requisitos Não Funcionais:

- Alto desempenho

  Principal justificativa do trabalho.

- Adequação da solução com o tamanho de memória da FPGA

  O sistema necessita possuir arquitetura compatível com a memória interna da placa FPGA.

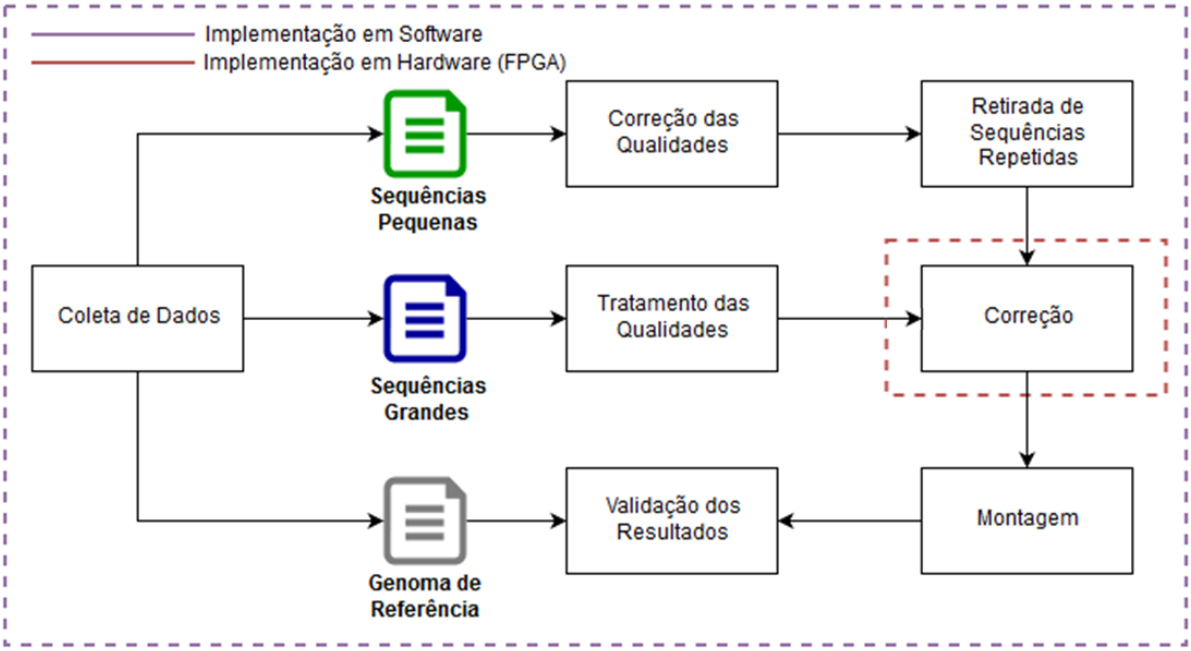
- Possibilidade de integração com outras ferramentas de montagem

  A ferramenta desenvolvida deve poder ser integrada com outras ferramentas, para finalizar o processo de montagem do genoma.

4.2 Arquitetura do Sistema

A arquitetura, apesentada pela figura 15, foi desenvolvida de maneira modular, onde se destaca sua implementação em software e em hardware (FPGA).

Figura 15: Arquitetura da ferramenta



Fonte: Autor

A seguir serão descritas suas etapas.

4.2.1 Coleta de Dados

Durante a etapa de coleta de dados, todos os dados relativos ao genoma dos seres vivos necessários para a montagem e validação são coletados. Os dados extraídos do banco podem ser divididos em três tipos. São eles as sequências pequenas Illumina, as sequências grandes (PacBio) e um genoma de referência para comparação.

Deve-se ter um cuidado para coletar dados correspondentes a organismos de uma mesma espécie. Variações na especie podem resultar em erros durante a montagem e a validação dos resultados.

4.2.2 Correção das Qualidades das Sequências Illumina

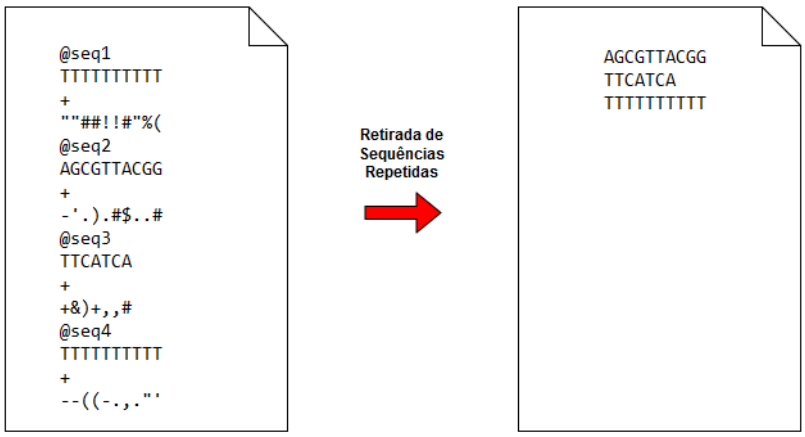
Mesmo com as sequências Illumina possuindo qualidade superior às sequências PacBio, erros provenientes do processo de sequenciamento não podem ser descartados. Na etapa de correção das qualidades, o arquivo Illumina precisa passar por um processo para garantir que todas as suas sequências possuem boa qualidade. Isso é necessário para que a correção não gere resultados errados. Uma ferramenta já desenvolvida pode ser utilizada aqui para este propósito.

4.2.3 Retirada das Sequências Repetidas

Conforme discutido anteriormente, os arquivos de sequências Illumina possuem alta cobertura para garantir a montagem adequada do genoma. Porém, a cobertura pode se refletir em sequências exatamente iguais, que ao invés de facilitar o processo de montagem, dificulta-o por aumentar o volume de dados a ser processado. Por isso optou-se em realizar um processamento dos dados referentes às sequências Ilumina, visando retirar as sequências repetidas.

Uma premissa adotada nesta etapa é que como já foi realizado um tratamento nas qualidades das sequências Illumina, pode-se confiar que todas as bases nitrogenadas remanescentes possuem alta qualidade. Assim, a informação relativa à qualidade de cada base não possui mais importância. Portanto, além da retirada das sequências repetidas, toda a informação presente no arquivo não correspondentes às sequências (cabeçalho da sequência, cabeçalho da base e qualidades) foi retirada, restando apenas as linhas do arquivo relativas às bases nitrogenadas. A figura 16 demonstra o processo.

Figura 16: Etapa de retirada de sequências repetidas



Fonte: Autor

4.2.4 Tratamento de Qualidades das Sequências PacBio

O *Q-score* presente nos arquivos PacBio costuma ser baixo, variando em média de 1 a 10 na codificação do tipo Sanger. Devido à alta probabilidade de erro consequente do baixo *Q-score*, é possível presumir que determinadas bases necessariamente serão substituídas durante a etapa de correção.

Para facilitar o processo de correção, nesta etapa é realizado um tratamento nas qualidades do arquivo PacBio. O propósito aqui é esgotar a utilidade das qualidades do arquivo, de forma que enquanto a entrada é um arquivo fastq, a saída é um arquivo fasta.

Para tal, inicialmente é necessário identificar o tipo de codificação utilizado no arquivo fastq, e então é estabelecido um *threshold* para o *Q-score*. O arquivo é percorrido de tal forma que caso o *score* da base seja inferior ao *threshold*, a base é substituída pela letra N. Durante a etapa da correção, a letra N terá um tratamento diferenciado pelo algoritmo. Esse *threshold* é flexível, com a possibilidade de obter diferentes resultados provenientes do tratamento.

4.2.5 Correção

A etapa de correção é a que possui maior custo computacional dentre as outras etapas da ferramenta. Por isso sua implementação foi feita em *software* e em *hardware* na placa FPGA. Seu algoritmo básico é a utilização de uma janela deslizante para deslocar as sequências pequenas (Illumina) nas sequências grandes (PacBio).

Implementar o algoritmo utilizando a janela deslizante é o método mais efetivo de se obter o melhor resultado na correção frente a outros métodos como a utilização de grafos, por exemplo. Porém, esse método é o mais custoso computacionalmente, de forma que critérios estatísticos e heurísticos devem ser utilizados para otimizar o processo.

Cabe ressaltar que a utilização aqui da janela deslizante é devido ao fato que sua implementação em *hardware* é facilitada, sendo necessário utilizar deslocadores, comparadores e somadores para fazer o cálculo das similaridades locais.

4.2.6 Montagem

A etapa de montagem é realizada utilizando alguma ferramenta de montagem de genoma. Pode-se optar por utilizar ferramentas puramente de montagem, como é o caso do Smartdenovo, ou então ferramentas que realizam a montagem e a correção, porém permitem realizar apenas o processo de montagem, como o Canu.

4.2.7 Validação dos Resultados

Para realizar a validação dos resultados, o genoma resultante do processo de montagem precisa ser comparado com o genoma de referência obtido durante a etapa de coleta dos dados. Esta comparação pode ser realizada através do alinhamento entre ambos, usando ferramentas já projetadas com este propósito.

5 PROJETO E IMPLEMENTAÇÃO

O capítulo apresenta a implementação da ferramenta, baseada na arquitetura que foi levantada no capítulo anterior.

5.1 Coleta de Dados

Optou-se por utilizar o banco de dados do NCBI, pois se trata de um banco com alta confiança e grande volume de dados disponíveis para análise. Devido a uma questão de terminologia, os arquivos contendo as sequências de DNA ainda não montadas são chamados *raw sequences*, e estão presente no *Sequence Raw Archive* (SRA).

Para a coleta de dados, o NCBI fornece um conjunto de ferramentas para acesso de seu banco. A ferramenta aqui utilizada foi a *fastq-dump* que recebe como entrada um identificador de determinado arquivo presente no SRA e retorna esse arquivo no formato .fastq. O identificador é uma *string* contendo 3 letras seguidas por 7 dígitos, que é gerada após o dado ser depositado no banco. Esse processo necessita ser feito tanto com o arquivo das sequências Illumina quanto no das sequências PacBio. O genoma de referência não necessita de ferramentas para ser coletado, sendo necessário apenas acessar o banco de genomas e escolher o arquivo fasta desejado.

Neste projeto foi realizada a montagem da espécie E. coli (*Escherichia coli*), pois se trata de um organismo amplamente estudado e com menor complexidade genética. Seus arquivos fastq possuem tamanho da ordem de centenas de MB.

5.2 Correção das Qualidades das Sequências Illumina

Para está etapa optou-se por utilizar a ferramenta *Trimmomatic*.

5.3 Retirada das Sequências Repetidas

O arquivo de entrada e o arquivo de saída foram mapeados em memória utilizando-se a função *mmap*, com o propósito de aproveitar a hierarquia de memória do sistema. Neste caso, o sistema operacional aloca uma região de memória para armazenar os dados de entrada e saída do algoritmo. No caso da leitura, é necessário apenas acessar o conteúdo da memória alocada. Para a escrita, além de escrever na região alocada, também é preciso escrever no arquivo de saída utilizando a função *msync*. Além disso, após o término do algoritmo é preciso liberar a memória alocada com o *mmap*. Para isso utiliza-se a função *munmap*, caso contrário ocorrerá *memory leak*.

O primeiro passo foi extrair as linhas que não correspondem às bases nitrogenadas em si. Conforme discutido, o arquivo fasta representa uma sequência como um bloco de 4 linhas, onde apenas a segunda linha do bloco são as bases nitrogenadas. Foi utilizado então o comando *awk* do UNIX para cortar as outras três linhas de cada bloco, gerando um arquivo intermediário contendo apenas as linhas com as bases.

Com esse arquivo intermediário foi necessário estabelecer uma abordagem para retirar as sequências repetidas.

Inicialmente a abordagem utilizada foi selecionar cada sequência do arquivo individualmente e percorrer o resto do arquivo para buscar uma sequência idêntica. Essa abordagem foi implementada em C e não se mostrou eficiente, levando algumas horas para realizar o processo em um arquivo com algumas centenas de MB. Isso é justificado pelo fato que a abordagem utilizada neste momento possui complexidade de n^2, sendo necessário então selecionar uma abordagem mais adequada devido ao aspecto escalável do tamanho dos arquivos.

Foi utilizada então a abordagem de primeiro se ordenar as sequências do arquivo para depois buscar sequências repetidas. Assim, é necessário apenas observar a sequência logo abaixo da sequência selecionada. Para a ordenação do arquivo foi utilizado o comando *sort* do UNIX, que implementa um algoritmo de ordenação com complexidade $nlog(n)$. A busca por sequências repetidas foi novamente implementada em C, porém agora a complexidade é de apenas n, sendo necessário varrer o arquivo apenas uma vez para detectar as sequências repetidas. Desta forma a complexidade total do processo foi de $n+nlog(n)$, sendo necessários poucos segundos para realizar o processo no mesmo arquivo da abordagem anterior

Optou-se por armazenar e ordenar de maneira decrescente, em outro aquivo de saída,

as sequências repetidas, pois esse dado pode ser utilizado por pesquisadores para identificar fatos como a repetição exata de certos tipos de proteína.

Como esta etapa não possui um número grande de operações aritméticas, predominando-se as operações de leitura e escrita, o algoritmo foi feito de maneira sequencial, pois o *speedup* obtido é baixo em situações como esta.

5.4 Tratamento de Qualidades das Sequências Pac-Bio

A implementação aqui foi dividida em duas partes, onde a primeira parte consiste em se identificar o tipo de codificação utilizado no arquivo fastq e a segunda parte compreende realizar o tratamento. A escolha de se realizar esta etapa de maneira modular está no fato que o código desenvolvido para identificar o tipo de codificação é uma contribuição parcial deste trabalho, o que será discutido na seção de contribuições.

Foi novamente utilizada a linguagem C com o OpenMP. Assim como na etapa de retirada das sequências repetidas Illumina, também foi realizado o mapeamento dos arquivos de entrada e saída em espaços de memória utilizando o *mmap* em ambos os algoritmos.

O algoritmo desenvolvido aqui consiste em dividir o arquivo de entrada entre as *threads* geradas usando um laço do tipo *while*. A importância de se utilizar o *while* ao invés do laço *for* é garantir um balanceamento de carga entre as *threads*, melhorando o desempenho do algoritmo. Desta forma, foi utilizado um ponteiro público para o conteúdo do arquivo, onde tanto a leitura do arquivo de entrada quanto a escrita no arquivo de saída ocorre em regiões críticas, para se garantir a sincronização do processo. Cada *thread* recebe um bloco de 4 linhas do arquivo fastq por vez, que corresponde a uma sequência genômica.

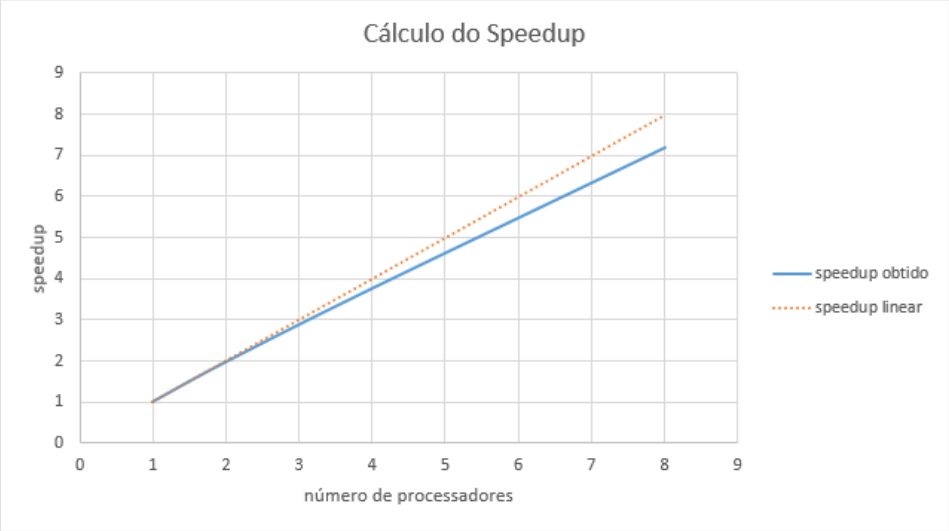
Para identificar o tipo de codificação utilizada, cada *thread* recebe apenas as linhas correspondentes às qualidades das bases. As linhas então são percorridas pelas *threads*, onde cada caractere é analisado individualmente. A análise consiste em verificar se o número ASCII do caractere se encontra nas faixas de ASCII de cada codificação. Caso alguma *thread* identifique um caractere não pertencente a determinada codificação, aquela codificação é descartada. O processo ocorre até que reste apenas uma codificação.

As linhas das bases nitrogenadas e das qualidades das bases são percorridas ao mesmo tempo, onde uma verificação é realizada na qualidade de cada base. Caso o *Q-score* da base seja menor que o *threshold* estabelecido, a base é substituída pela letra N. Após a verificação ser realizada em todas as bases da sequência, a *thread* escreve o bloco no

arquivo de saída e recebe um novo bloco, até o término dos dados do arquivo de entrada.

Nesta etapa, ambas as partes possuem um número considerável de operações aritméticas, devido às comparações que são realizadas com o número ASCII das qualidades. Por isso foi realizada uma implementação paralela do algoritmo. A figura 17 apresenta o gráfico de *speedup* da etapa. Observa-se que o *speedup* se aproxima do linear com um número pequeno de *cores*, e se distancia com o aumento do número. Isso é justificado pelas seções críticas necessárias para a leitura e escrita dos dados.

Figura 17: *Speedup* obtido no tratamento das qualidades



Fonte: Autor

5.5 Correção

Conforme ilustrado pela figura 15, a implementação desta etapa foi feita em *software* e em *hardware*. A seguir serão apresentadas as duas implementações.

5.5.1 Implementação em *Software*

A implementação em *software* foi feita com a linguagem C e o OpenMP, juntamente com o *mmap* para a leitura e escrita dos dados. Novamente foram utilizados laços do tipo *while* para realizar o balanceamento de carga entre as *threads*.

O algoritmo trabalha com dois arquivos de entrada, sendo um correspondente às sequências pequenas e o outro às sequências grandes. O ponteiro para a área de memória alocada das sequências grande deve ser público, enquanto o ponteiro para a área das

sequências pequenas é privado, para cada *thread* poder realizar a correção em sua sequência grande alocada de maneira independente.

Cada *thread* então recebe uma sequência grande e percorre a área de memória das sequências pequenas com o seu ponteiro, captando uma sequência pequena por vez e realizando a janela deslizante. Optou-se por considerar apenas as opções de acerto e erro durante o alinhamento, onde duas bases iguais ou uma base com a letra N equivalem a um acerto, e bases distintas equivalem a um erro. Um acerto acrescenta com 1 ponto para o cálculo da similaridade, enquanto um erro não incrementa a pontuação.

Para cada alinhamento na *thread* é armazenado o valor da similaridade máxima obtida até o momento mais um *offset* que representa o quanto a sequência pequena foi deslizada pela sequência grande. Esse *offset* é utilizado após o término da janela deslizante para realizar a correção.

Esse processo é repetido em cada *thread* para todas as sequências pequenas do arquivo, e em todas as sequências grades. Observa-se que o algoritmo possui complexidade elevada, sendo necessário então utilizar alguns critérios estatísticos e heurísticos para melhorar o tempo de execução.

Nesta implementação foram utilizados três critérios.

O primeiro é o tamanho do arquivo Illumina (sequencias pequenas). Como já foi discutido, a montagem Illumina necessita de uma alta cobertura para garantir a montagem, porém, como o propósito aqui com o arquivo não é realizar a montagem, mas sim a correção, pode-se utilizar apenas uma parte do arquivo, semelhante ao que a ferramenta LoRDEC faz. Assim, pode-se truncar o arquivo de forma a utilizar apenas uma parte das sequências pequenas presentes nele.

Um outro critério é o conceito de um "*threshold* forte". Considera-se aqui um valor tão alto para a similaridade obtida que existe uma alta possibilidade de ele não ser superado continuando o deslizamento da sequência pequena na grande. Portanto, pode-se realizar a correção logo após o valor ser obtido, e a sequência já pode ser descartada.

O último critério consiste em uma maneira de se pular sequências pequenas sem realizar o deslizamento sobre a sequência grande. Aqui considera-se primeiro se a sequência pequena anterior corrigiu ou não a sequência grande. Caso a correção tenha sido realizada, esse critério não tem funcionamento. Em caso contrário, observa-se a diferença de caracteres entre a sequência pequena anterior e a próxima. Essa diferença somada com a similaridade máxima obtida anteriormente deve ser superior ao *threshold* estabelecido. Se

isso não acontecer, é possível inferir que a sequência pequena atual não tem capacidade de corrigir a sequência grande, podendo passar para a próxima sequência pequena. Esse processo pode ocorrer inúmeras vezes, desde que o comprimento da sequência anterior e da sequência atual sejam iguais.

5.5.2 Implementação em *Hardware*

O circuito digital em *hardware* para realizar a correção das sequências depende inicialmente de uma codificação das bases. Como é utilizado apenas o conceito de bits, cada letra necessita ser codificada em um conjunto de três bits. A tabela 3 apresenta a codificação adotada. A codificação juntamente com a respectiva decodificação pode ser realizada tanto pela CPU quanto pela FPGA.

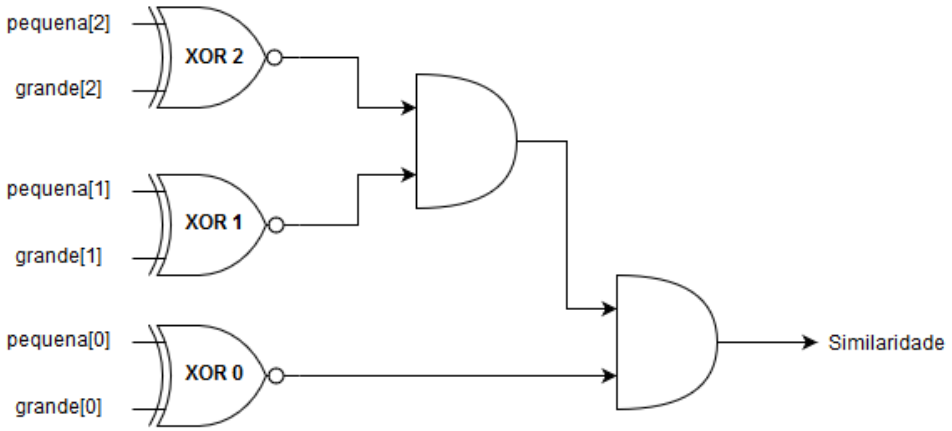
Tabela 3: Codificação utilizada para as bases nitrogenadas

Caractere	**Codificação**
A	000
T	001
C	010
G	011
término	100
N	111

Fonte: Autor

Os bits relativos às sequências grandes e pequenas são carregados em registradores, onde a sequência grande é armazenada em um registrador de deslocamento. Para realizar o alinhamento de uma base, os três bits da base da sequência grande precisam ser comparados com os da sequência pequena, utilizando portas XNOR. Os resultados das portas XNOR devem passar por portas AND para garantir que os três bits são iguais. A figura 18 ilustra o circuito descrito, enquanto a tabela 4 apresenta alguns resultados para diferentes alinhamentos de bases.

Figura 18: Circuito digital comparador de bases nitrogenadas



Fonte: Autor

Tabela 4: Resultados para diferentes comparações entre bases

Caractere 1	**Caractere 2**	**XOR 2**	**XOR 1**	**XOR 0**	**Similaridade**
000 (A)	000 (A)	1	1	1	1
000 (A)	001 (T)	1	1	0	0
C (010)	011 (G)	1	1	0	0
C (010)	C (010)	1	1	1	0

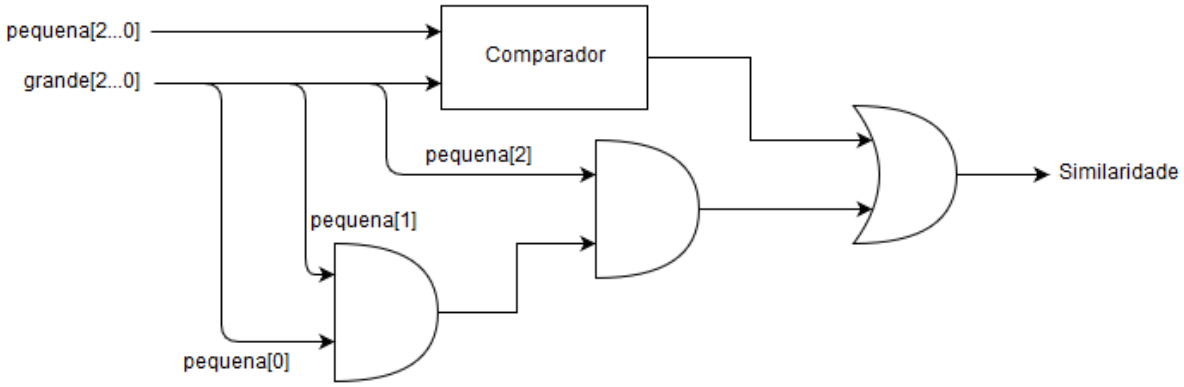
Fonte: Autor

Para o tratamento do caractere N nas sequências grandes, o resultado obtido no bloco comparador deve passar por uma porta OR, para verificar se os três bits do caractere remetem à letra N. Em caso positivo, a similaridade sempre será 1, como mostra a figura 19.

O alinhamento então ocorre entre todos os bits das sequências pequenas com os bits das sequências grandes em paralelo, devido à utilização de somadores cascateados. O valor da similaridade é armazenado em um registrador caso seja superior ao *threshold* estabelecido, juntamente com um *offset* de quanto a sequência grande foi deslocada, para realizar a posterior correção.

Ocorre então a janela deslizante, onde a cada ciclo de *clock* o registrador de deslocamento da sequência grande desloca a sua saída em 3 bits, que corresponde a um caractere. Isso ocorre até o caractere referente ao término da sequência grande ser obtido na saída. Nesse momento é verificado se o registrador da similaridade possui algum valor armazenado. Em caso positivo, a sequência pequena é deslocada até o *offset* armazenado e a

Figura 19: Circuito digital comparador de bases nitrogenadas com caractere N

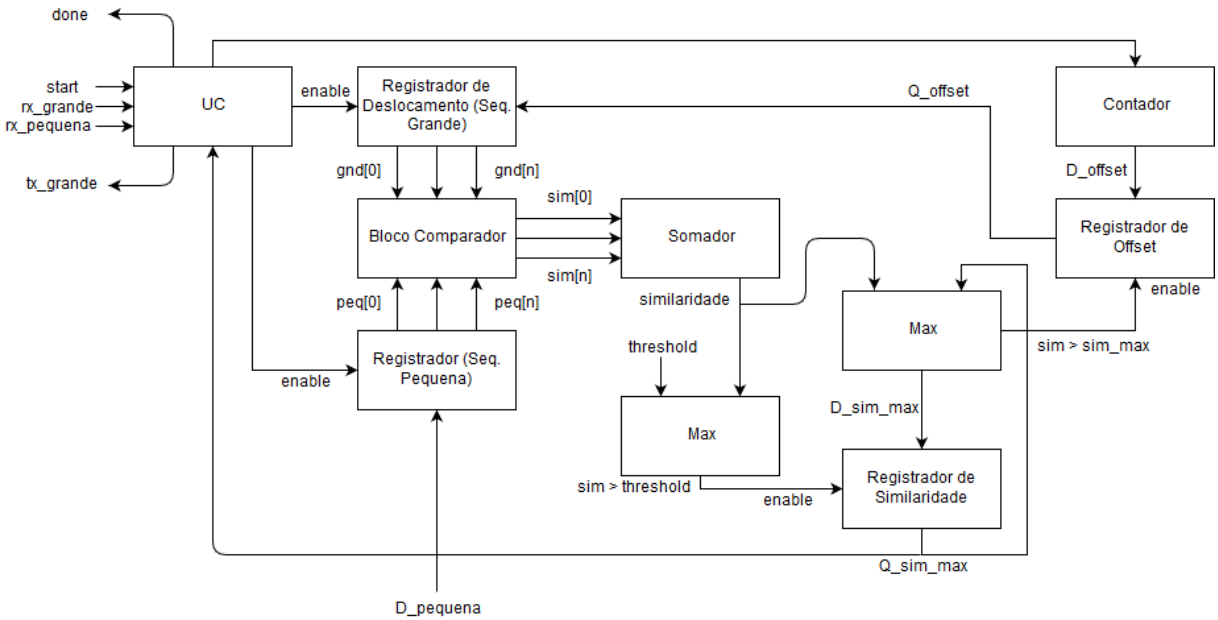


Fonte: Autor

correção é realizada.

A figura 20 apresenta o núcleo do circuito para correção das sequências grandes. O circuito projetado é totalmente paralelizável, onde um número qualquer de sequências grandes pode ser corrigido ao mesmo tempo, mediante a disponibilidade de memória da placa FPGA. Um ponto positivo deste tipo de abordagem é que o *speedup* é linear em um caso onde todas as sequências estão na memória da placa.

Figura 20: Circuito para correção de sequencias PacBio



Fonte: Autor

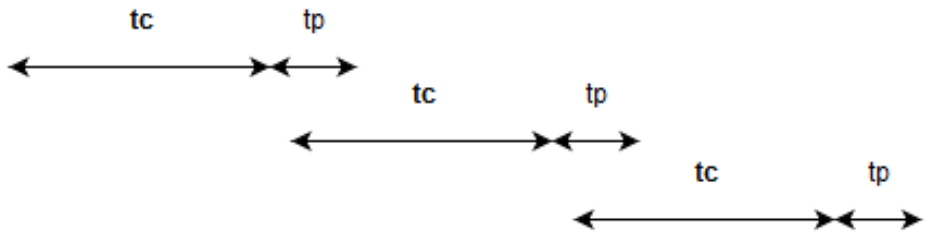
Foi estabelecida então uma estratégia de comunicação entre a FPGA e a CPU. Isso é necessário pois a FPGA possui pouca memória, sendo possível corrigir um número

pequeno de sequências grandes ao mesmo tempo. A estratégia estabelecida é apresentada a seguir.

Inicialmente um bloco de sequências grandes igual ao número de processos é enviado para placa. Logo após, as sequências pequenas são enviadas, uma a uma, para ocorrer os deslocamentos (janela deslizante) delas em todas as sequeêcias grandes ao mesmo tempo. Cada sequência pequena requer um tempo (tc) para ser enviada e um tempo (tp) para ocorrer o deslizamento por toda a sequência grande e realizar a correção. Após todas as sequências pequenas serem deslocadas nas sequências grandes, é necessário enviar o bloco de sequências grandes já corrigidas para a CPU, e receber um novo bloco de sequências grandes. O processo então é repetido até que todas as sequências grandes sejam corrigidas.

Observa-se que enquanto uma sequência pequena está sendo deslocada nas sequências grandes, a próxima sequência pequena já pode estar sendo enviada para a FPGA, ocorrendo então um *overlap* (tc-tp), conforme ilustrado pela figura 21.

Figura 21: *Overlap* entre comunicação e processamento



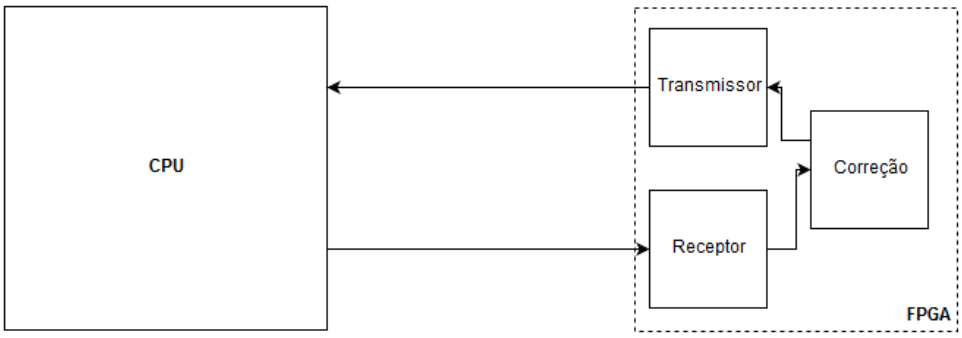
Fonte: Autor

Assim, o tempo total para correção de um arquivo de sequências grandes utilizando um arquivo de sequências pequenas é dado pela equação:

$$t = (\frac{tc_{pequena} + n_{pequenas} * tp + (n_{pequenas} - 1) * (tc_{pequena} - tp)}{n_{processos}} + tc_{grande}) * n_{grandes} + tc_{grandes} * n_{processos} \quad (5.1)$$

Desta forma, a figura 22 ilustra o diagrama de comunicação entre a CPU e a FPGA. Observa-se que ela é composta de 4 blocos, que são a CPU, o transmissor da FPGA, o receptor da FPGA e o circuito de correção que foi ilustrado na figura 20.

Figura 22: Comunicação entre CPU e FPGA



Fonte: Autor

A seguir são apresentados os fluxogramas da cada um dos blocos em questão.

A figura 23 apresenta o fluxograma da CPU. Inicialmente é enviado um bloco de sequências grandes de tamanho igual ao número de processos na placa. Então são enviadas sequências pequenas de maneira que enquanto uma sequência pequena é deslizada na FPGA, uma nova sequência já é enviada para ocorrer o *overlap*. Após a FPGA realizar o deslizamento de todas as sequências pequenas nas sequências grandes do bloco, a CPU recebe o bloco de sequências grandes corrigidas, onde uma *thread* escreve as sequências corrigidas no arquivo de saída e uma outra *thread* envia um novo bloco de sequências grandes para a FPGA. Esse processo se repete até a CPU receber o último bloco de sequências grandes corrigidas, realizando a última escrita no arquivo de saída e indicando o término da correção.

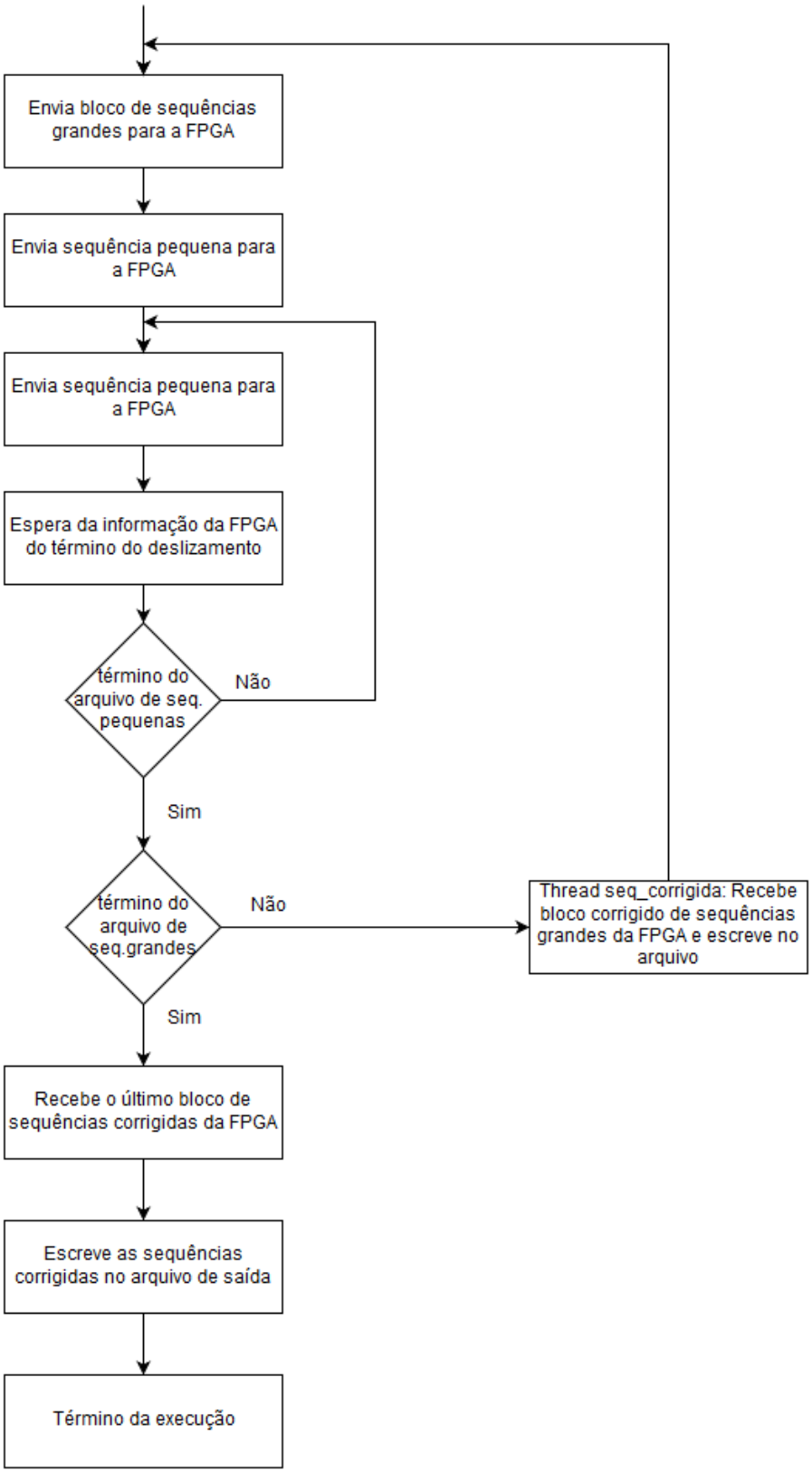
A figura 24 apresenta o fluxograma do circuito de correção da FPGA. O processo começa quando o receptor indica ao módulo de correção que um bloco de sequências grandes foi recebido. Logos após, uma sequência pequena é recebida pelo receptor, e esta é deslizada pelas sequências grandes em paralelo por cada processo. Caso a similaridade obtida durante o deslizamento seja superior ao *threshold* estabelecido, a correção é realizada. O término do deslizamento então é informado, fazendo a requisição de mais uma sequência pequena para a CPU. Isso ocorre até todas as sequências pequenas serem deslizadas pelo bloco de sequências grandes, que então é armazenado no *buffer* do transmissor. Esse processo se repete até a FPGA corrigir o último bloco de sequências grandes, colocando-o no *buffer* do transmissor e finalizando a correção

A figura 25 apresenta o fluxograma do circuito transmissor da FPGA. Sua função é observar a todo momento se houve um término da janela deslizante no circuito de correção e se o arquivo de sequências pequenas foi integralmente recebido. Em caso positivo, ele

envia para a CPU o bloco de sequências grandes corrigidas. Em caso negativo, ele indica para a CPU que houve o término da janela deslizante.

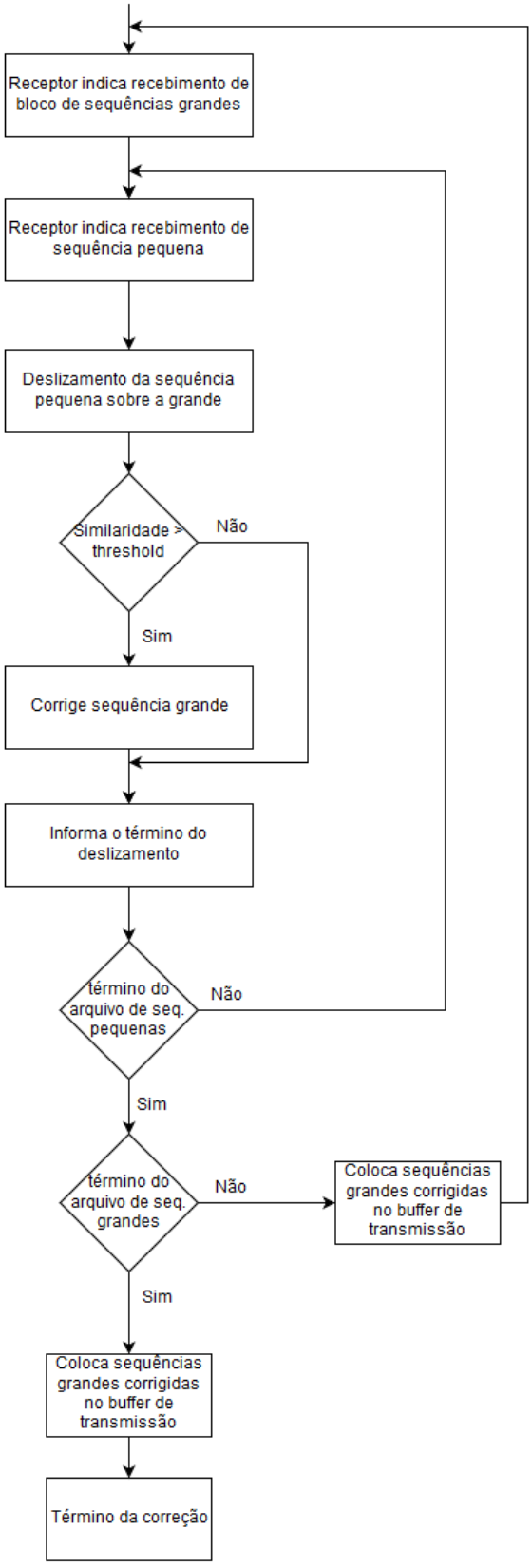
A figura 26 apresenta o fluxograma do circuito receptor da FPGA. Inicialmente ele recebe um bloco de sequências grandes da CPU, indicando o recebimento para o módulo de correção. Então ele recebe as sequências pequenas da CPU juntamente com a ocorrência da janela deslizante no módulo de correção para haver o *overlap*. Quando todas as sequências pequenas são recebidas, ele passa a receber um novo bloco de sequências grandes, e isso ocorre até o término de todas as sequências grandes vindas da CPU.

Figura 23: Fluxograma da CPU



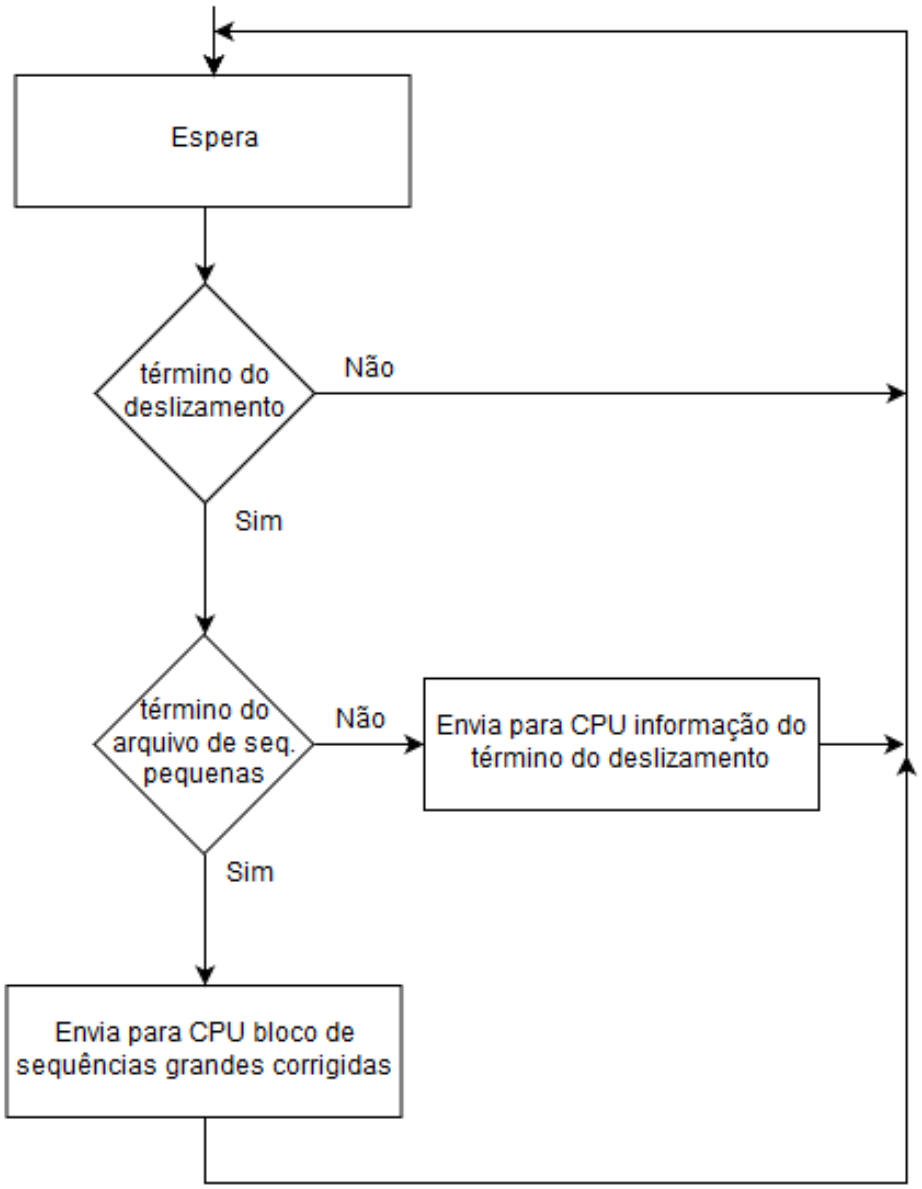
Fonte: Autor

Figura 24: Fluxograma do módulo corretor da FPGA



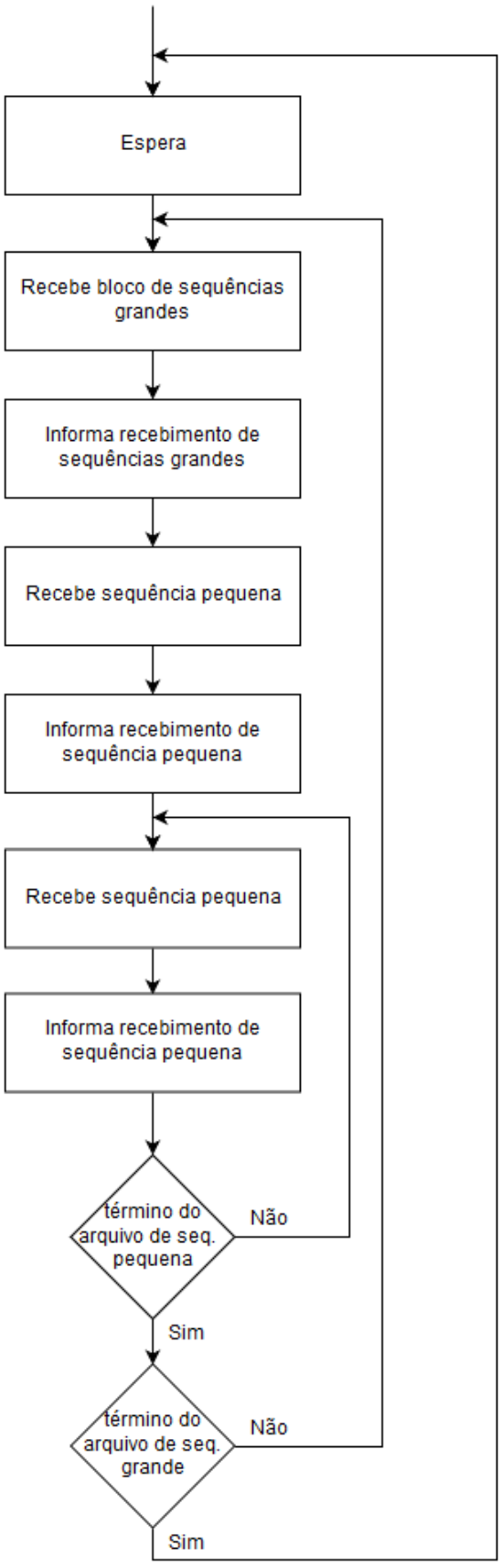
Fonte: Autor

Figura 25: Fluxograma do módulo transmissor da FPGA



Fonte: Autor

Figura 26: Fluxograma do módulo receptor da FPGA



Fonte: Autor

5.6 Montagem do Genoma

A montagem do genoma foi feita utilizando-se ferramentas já desenvolvidas para montagem com sequências PacBio. Optou-se por utilizar as ferramentas Canu e Smartdenovo, pois ambas além de possuírem boa visibilidade por parte da comunidade científica, possuem interface simples de ser utilizada, sem a necessidade de realizar processos de configuração.

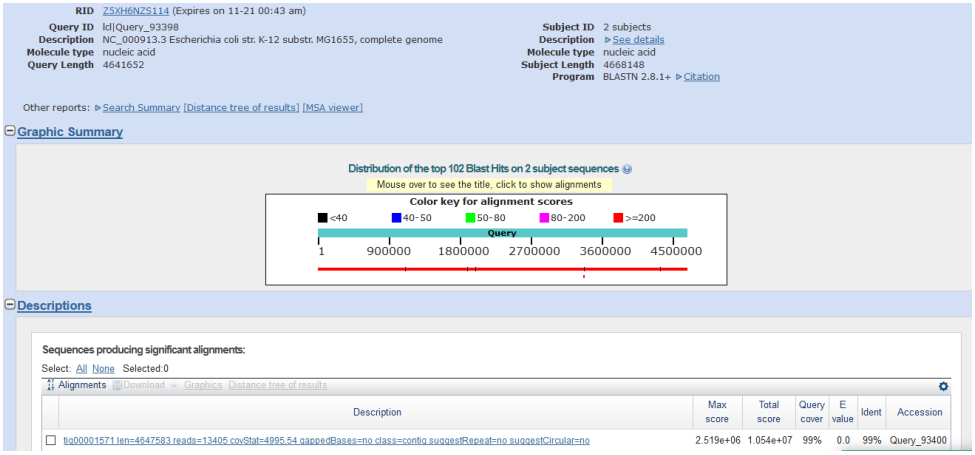
5.7 Validação dos Resultados

O alinhamento entre o genoma obtido e o de referência foi realizado com o BLAST, sendo necessário apenas enviar os dois arquivos para o *cluster* mantido pelo NCBI e receber como resposta um *log* contendo os resultados do alinhamento. Cabe ressaltar aqui que poderia ter sido utilizada uma versão local (para *desktop*) do BLAST, porém ela não foi escolhida devido ao grande custo computacional de se executar o mesmo.

A figura 27 apresenta um *log* proveniente de um alinhamento entre o genoma de referência e um genoma montado após a correção. A figura 28 apresenta um *log* proveniente de um alinhamento entre o mesmo genoma de referência e o mesmo genoma montado, porém sem a correção. Observa-se que a diferença entre ambos os casos é a maior presença das faixas pretas (representando baixa similaridade no alinhamento) no caso da montagem sem correção. Além disso, a montagem com correção apresentou identidade de 99% com o genoma de referência, enquanto a montagem sem correção apresentou identidade de 89%. Em ambos os casos existe a predominância das faixas vermelhas (representando alta similaridade) pois o exemplo de organismo aqui utilizado possui baixa complexidade genética.

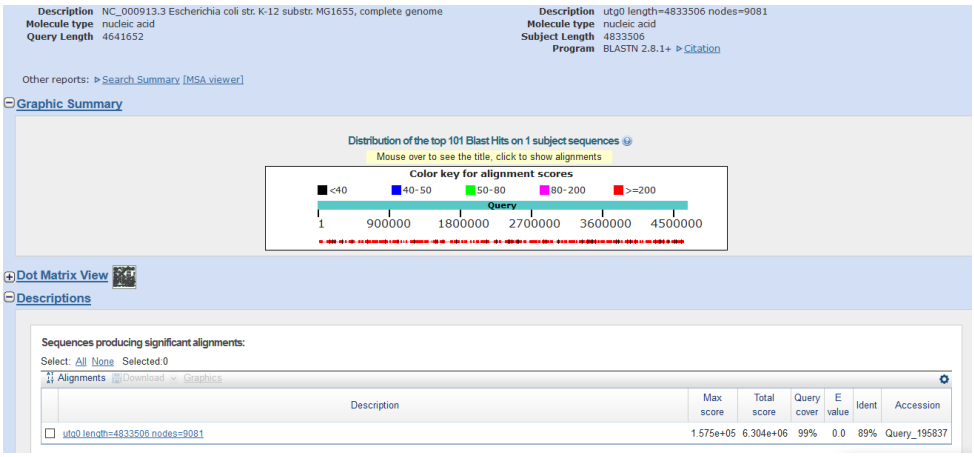
Ressalta-se aqui que o *log* completo gerado pelo BLAST apresenta mais informações que são importantes para os pesquisadores da área. Um exemplo desse tipo de informação é apresentar cálculos de similaridade para fragmentos do genoma de referência com fragmentos do genoma montado. Desta forma, é possível isolar as regiões distintas do DNA, conforme já discutido anteriormente.

Figura 27: *Log* gerado pelo BLAST para alinhamento entre genoma de referência e genoma corrigido



Fonte: Autor

Figura 28: *Log* gerado pelo BLAST para alinhamento entre genoma de referência e genoma sem correção



Fonte: Autor

6 TESTES E AVALIAÇÃO

O capítulo discute os testes feitos para validação da ferramenta para comparar sua implementação em *software* e em *hardware*.

6.1 Teste e Validação da Implementação em *Software*

Para testar todas as etapas da ferramenta e realizar sua validação, foi utilizado o material genético da E. coli. A escolha dela se dá em função de sua pouca complexidade genética, e por conseguinte, pouco tempo de execução necessário para sua correção e montagem. Foi realizada a correção e montagem da mesma com as ferramentas apresentadas no capítulo 3, visando a comparação entre a ferramenta aqui desenvolvida e o estado da arte. Os resultados são apresentados na tabela 5.

Tabela 5: Tempos de execução para deslocamento de uma sequência pequena em uma grande

Método	**Tempo Total (min)**	**Similaridade**
Canu	33	99,5%
Canu + Ferramenta	243	99,9%
Smartdenovo	2	89%
Smartdenovo + Canu	50	99,3%
Smartdenovo + Ferramenta	252	99,8%

Fonte: Autor

6.2 Desempenho Teórico da Implementação em *Hardware*

Para avaliar o desempenho da implementação feita na FPGA, foi realizada uma análise teórica dos resultados. Para isso, utilizou-se os parâmetros de uma placa FPGA DE0-CV, que possui *clock* de 50MHz e 3000 Kbits de memória (TERASIC, s.d.).

O levantamento dos dados teóricos foi possível utilizando a equação 5.1. A equação foi utilizada com os dois arquivos de teste com tamanho de 1MB. O arquivo das sequências grandes possui 122 sequências com tamanho médio de 1200 bases cada, enquanto o arquivo das sequências pequenas possui 5735 sequências com tamanho médio de 174 bases cada. Baseado nesses tamanhos médios e o tamanho de memória da placa, estimou-se um máximo de 20 processos ocorrendo em paralelo na FPGA, visando uma margem de segurança adequada nos cálculos.

Foram feitos cálculos para a comunicação do tipo RS-232, com taxa de 900KB/s e PCI Express com taxa de 50MB/s devido ao *clock* da placa ser o limitante. Os resultados são apresentados na tabela 6.

Tabela 6: Resultados teóricos obtidos em *hardware*

Comunicação	**Nº processos**	**Tempo de execução (s)**
RS-232	1	135,44
RS-232	20	6,95
PC Express	1	14,36
PC Express	20	0,72

Fonte: Autor

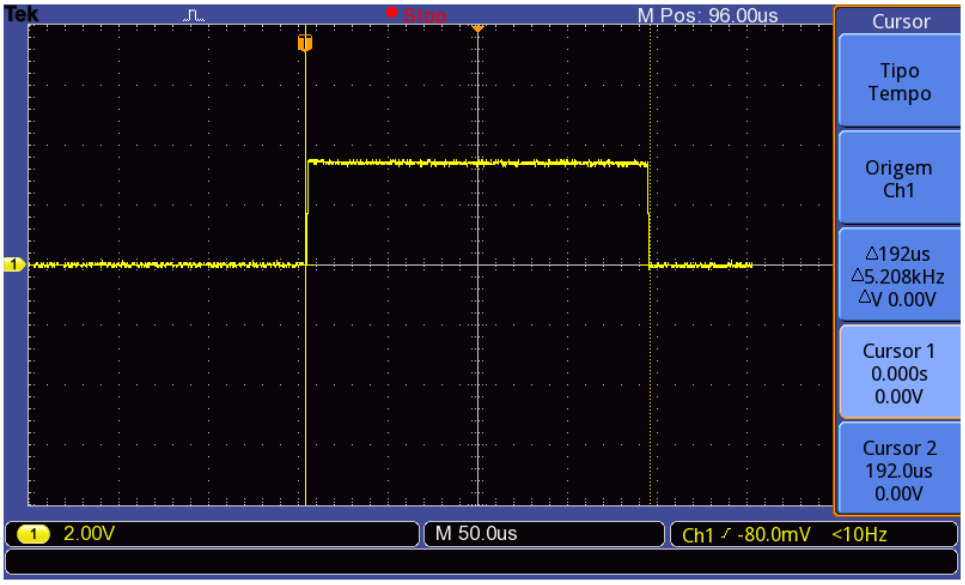
6.3 Comparação entre Implementação em *Software* e em *Hardware*

Para a comparação entre as duas implementações realizadas na etapa da correção foram feitos dois testes.

O primeiro teste consistiu em calcular o tempo de processamento para deslocar uma sequência pequena de 500 bases em uma sequência grande de 10000 bases. Esse tempo foi escolhido por se tratar de uma unidade pequena de processamento do algoritmo, que é executada múltiplas vezes. A captura do tempo de processamento na FPGA foi feita

utilizando-se um osciloscópio, conforme ilustrado pela figura 29. A tabela 7 apresenta os resultados obtidos em *software* e em *hardware* considerando um único processo (sequencial).

Figura 29: Captura de tempo de correção da FPGA no osciloscópio



Fonte: Autor

Tabela 7: Tempos de execução para deslocamento de uma sequência pequena em uma grande

Implementação	**Tempo de execução (ms)**
software	1.578
hardware	0.192

Fonte: Autor

O segundo teste foi calcular o tempo de execução em *software* para os arquivos testes utilizados nos cálculos teóricos do *hardware* (arquivos com 1MB). O propósito aqui foi verificar a escalabilidade quanto ao número de *cores*, observando como o desempenho do *software* e do *hardware* varia com o aumento do processamento. Os resultados são apresentados na tabela 8.

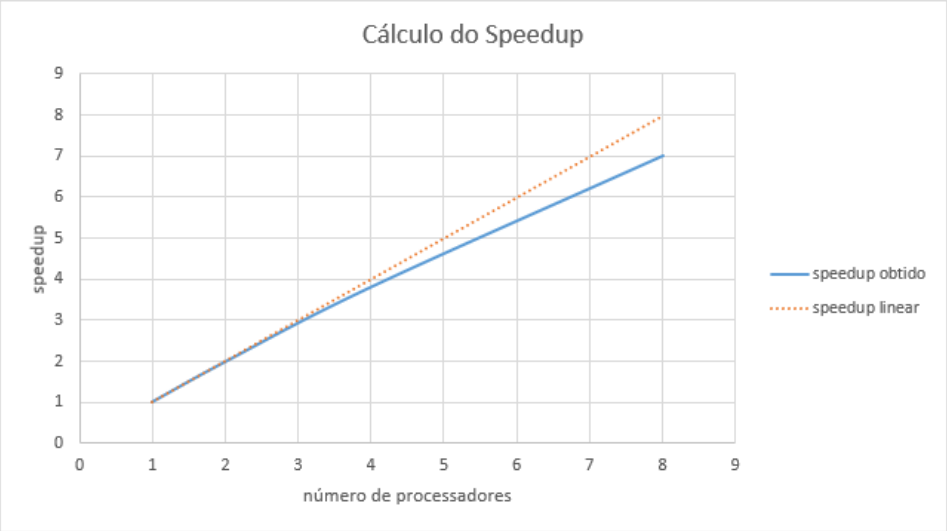
Tabela 8: Tempos de execução para correção de arquivos de 1MB em *software*

Nº *cores*	**Tempo de execução (s)**
1	221,85
2	112,03
4	58,21
8	31,6

Fonte: Autor

A figura 30 apresenta graficamente o *speedup* da correção em *software*.

Figura 30: *Speedup* da correção em *software*



Fonte: Autor

Com base nos resultados dos dois testes, observa-se que nesta implementação o *hardware* tem um desempenho cerca de 200x superior ao *software* sequencial.

7 CONSIDERAÇÕES FINAIS

O capitulo realiza o encerramento da monografia, apresentando as conclusões obtidas no projeto de formatura, suas contribuições e perspectivas de trabalhos futuros.

7.1 Conclusões do Projeto de Formatura

O projeto de formatura alcançou seu objetivo, que foi a implementação de um algoritmo para realizar a montagem de sequências genômicas através da correção híbrida. Observou-se que o desempenho da ferramenta desenvolvida possui tempo de execução inferior em *software* a outras ferramentas com o mesmo propósito, porém, o genoma resultante deste processo possui maior qualidade. A implementação em *hardware* apresentou resultados teóricos superiores a todos os resultados em *software* para até um número determinado de *cores*, mostrando a viabilidade desta implementação.

7.2 Contribuições

O projeto de formatura resultou na criação de uma ferramenta para montagens de sequências genômicas através da correção híbrida. A novidade desta ferramenta frente a outras já usadas no estado da arte é a utilização da FPGA com este propósito, permitindo sua aplicação em pesquisas futuras de maneira mais intensiva.

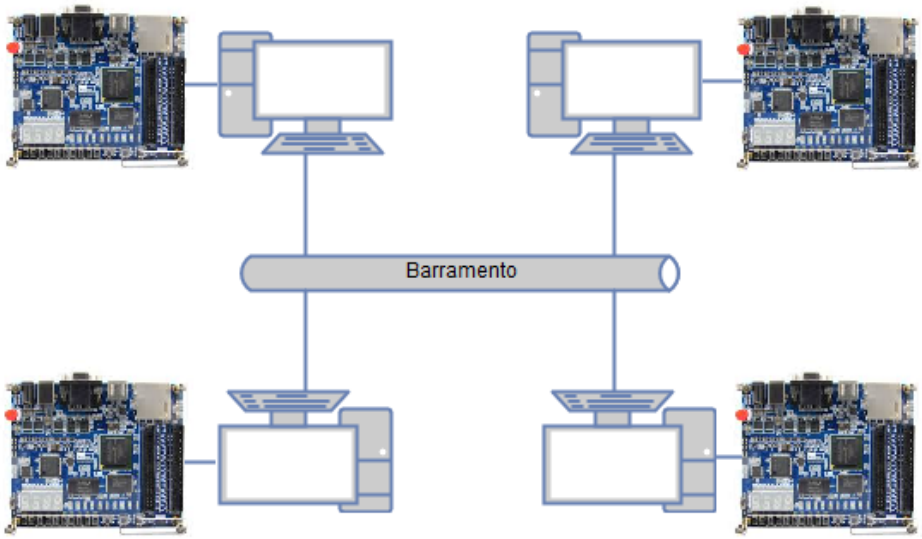
Algumas das etapas aqui desenvolvidas também podem ser melhoradas e transformadas em ferramentas de propósito específico. A etapa de retirada das sequências repetidas pode ser utilizada por pesquisadores da área não apenas para a montagem PacBio, mas também para a montagem Illumina. O algoritmo para detectar o tipo de codificação utilizada no arquivo fastq também pode ser empregado para diversos propósitos.

7.3 Trabalhos Futuros

O projeto de formatura realizado possui como propósito sua continuação no programa de mestrado da instituição. Para tal, uma série de melhorias e aprofundamentos na montagem ainda podem ser implementados. Dentre os trabalhos futuros possíveis de se implementar no mestrado destacam-se:

- Desenvolvimento da ferramenta em um ambiente de memória compartilhada e distribuída. Conforme os valores apresentados nos testes realizados, a utilização da FPGA como aceleradora de algoritmo possui desempenho superior ao *sofware*. Para melhorar ainda mais o desempenho, poder-se-ia utilizar múltiplos computadores com CPUs *multicore*, cada um com uma placa FPGA, conforme ilustrado pela figura 31.

Figura 31: Ambiente de memória compartilhada e distribuída com FPGAs



Fonte: Autor

- Estudo de novas heurísticas. No tempo estabelecido para este projeto foi possível apenas utilizar uma heurística que foi a cobertura do arquivo Illumina. É possível, com base no conhecimento do organismo, melhorar o desempenho da ferramenta, utilizando informações como o tamanho do genoma de referência, o número de cromossomos do organismo, a cobertura do arquivo PacBio dentre outras.

- Maior utilização de critérios estatísticos. Os critérios estatísticos aqui utilizados auxiliaram na melhoria do desempenho do algoritmo. Uma possibilidade de melhoria na ferramenta está na utilização de novos critérios, além de um aprofundamento maior nos critérios escolhidos.

- Análise prática de desempenho na FPGA. Os dados aqui levantados para o desempenho na FPGA foram teóricos, com exceção da captura no osciloscópio. O levantamento prático dos dados é necessário para reforçar as conclusões obtidas neste projeto.

- Montagem de outros organismos. Os testes realizados neste trabalho foram com a E. coli, devido ao tempo necessário para se realizar a montagem de organismos mais complexos. Um exemplo de organismo cujo genoma pode ser futuramente montado é a levedura *Saccharomyces cerevisiae*, que também é amplamente estudada.

REFERÊNCIAS

ANGEL, V. D. D. et al. Ten steps to get started in genome assembly and annotation. *F1000Research*, Faculty of 1000, v. 7, 2018.

BANKEVICH, A. et al. Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, Mary Ann Liebert 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA, v. 19, n. 5, p. 455–477, 2012.

BLAST. 2018. ⟨https://blast.ncbi.nlm.nih.gov/Blast.cgi⟩. Acesso: 25/07/2018.

BOLGER, A. M.; LOHSE, M.; USADEL, B. Trimmomatic: a flexible trimmer for illumina sequence data. *Bioinformatics*, Oxford University Press, v. 30, n. 15, p. 2114–2120, 2014.

BROWN, T. A. *Genomes*. 2nd.. ed. [S.l.]: NCBI Bookshelf, 2002. ISBN 0471250465.

BRUDER, C. E. et al. Phenotypically concordant and discordant monozygotic twins display different dna copy-number-variation profiles. *The American Journal of Human Genetics*, Elsevier, v. 82, n. 3, p. 763–771, 2008.

CHAPMAN, B.; JOST, G.; PAS, R. v. d. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. [S.l.]: The MIT Press, 2007. ISBN 0262533022, 9780262533027.

COCK, P. J. et al. The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants. *Nucleic acids research*, Oxford University Press, v. 38, n. 6, p. 1767–1771, 2009.

COMPEAU, P.; PEVZNER, P.; TESLER, G. How to apply de bruijn graphs to genome assembly. *Nature biotechnology*, v. 29, n. 11, p. 987–991, 2011.

CONSORTIUM, . G. P. et al. A map of human genome variation from population-scale sequencing. *Nature*, v. 467, n. 7319, p. 1061, 2010.

CONSORTIUM, . G. P. et al. A global reference for human genetic variation. *Nature*, v. 526, n. 7571, p. 68, 2015.

FLOWERS, G. P.; CREWS, C. M. *Regeneration writ large*. [S.l.]: Nature, 2018.

GOODWIN, S.; MCPHERSON, J. D.; MCCOMBIE, W. R. Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, v. 17, n. 6, p. 333, 2016.

GROHME, M. A. et al. The genome of schmidtea mediterranea and the evolution of core cellular mechanisms. *Nature*, v. 554, n. 7690, p. 56, 2018.

HUFFSTETLER, J. *Intel Processors and FPGAs—Better Together*. 2018. ⟨https://itpeernetwork.intel.com/intel-processors-fpga-better-together/⟩. Acesso: 08/10/2018.

ILLUMINA. *De Novo Assembly Using Illumina Reads*. 2010.

KOREN, S. et al. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research*, Cold Spring Harbor Lab, v. 27, n. 5, p. 722–736, 2017.

MISALE, C. et al. Sequence alignment tools: one parallel pattern to rule them all? *BioMed research international*, Hindawi, v. 2014, 2014.

NCBI. *NCBI Handbook*. 2nd. ed. [S.l.]: U.S. National Institutes of Health, 2013.

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*, Elsevier, v. 48, n. 3, p. 443–453, 1970.

NOWOSHILOW, S. et al. The axolotl genome and the evolution of key tissue formation regulators. *Nature*, v. 554, n. 7690, p. 50, 2018.

PACHECO, P. *An Introduction to Parallel Programming*. 1st. ed. San Francisco, CA, USA: Morgan Kaufmann, 2011. ISBN 9780123742605.

PATTERSON, D. A.; HENNESSY, J. L. *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*. 5th. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2013. ISBN 0124077269, 9780124077263.

RHOADS, A.; AU, K. F. Pacbio sequencing and its applications. *Genomics, Proteomics Bioinformatics*, v. 13, n. 5, p. 278 – 289, 2015. ISSN 1672-0229. SI: Metagenomics of Marine Environments. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S1672022915001345⟩.

RUAN, J. *SMARTdenovo*. ⟨https://github.com/ruanjue/smartdenovo⟩. Acesso: 02/11/2018.

SALMELA, L.; RIVALS, E. Lordec: accurate and efficient long read error correction. *Bioinformatics*, Oxford University Press, v. 30, n. 24, p. 3506–3514, 2014.

SETUBAL, J. C.; MEIDANIS, J. *Introduction to Computational Molecular Biology*. [S.l.]: PWS Pub., 1997. (Computer Science Series). ISBN 9780534952624.

SIMS, D. et al. Sequencing depth and coverage: key considerations in genomic analyses. *Nature Reviews Genetics*, v. 15, n. 2, p. 121, 2014.

STROHMAIER, E. et al. *TOP 500*. 2018. ⟨https://www.top500.org/lists/2018/11/⟩. Acesso: 13/11/2018.

TERASIC. *DE0-CV board*. s.d. ⟨http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=921⟩. Acesso: 15/11/2018.

TESLA, N. *My Inventions: The Autobiography of Nikola Tesla*. [S.l.]: Wildside Press, 2005. (Prime classics library). ISBN 9781557423634.

VENTER, J. C. et al. The sequence of the human genome. *science*, American Association for the Advancement of Science, v. 291, n. 5507, p. 1304–1351, 2001.

WHEELER, D. A. et al. The complete genome of an individual by massively parallel dna sequencing. *Nature*, v. 452, n. 7189, p. 872, 2008.

ZHANG, J. et al. The impact of next-generation sequencing on genomics. *Journal of Genetics and Genomics*, v. 38, n. 3, p. 95–109, 2011. ISSN 1673-8527.