FELIPE VALENCIA DE ALMEIDA

**Ferramenta para montagem de sequências genômicas
em ambientes de memória compartilhada e
distribuída com FPGAs**

São Paulo
2020

FELIPE VALENCIA DE ALMEIDA

**Ferramenta para montagem de sequências genômicas
em ambientes de memória compartilhada e
distribuída com FPGAs**

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Ciências.

Área de Concentração:

Engenharia de Computação

Orientador:

Profa. Dra. Liria Matsumoto Sato

São Paulo
2020

Nome: Almeida, Felipe Valencia de

Título: Ferramenta para montagem de sequências genômicas em ambientes de memória compartilhada e distribuída com FPGAs

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Ciências

Aprovado em:

Banca Examinadora

Profa. Dra. Liria Matsumoto Sato

Prof. Dr. Calebe de Paula Bianchini

Profa. Dra. Denise Stringhini

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catálogo-na-publicação

Almeida, Felipe Valencia de

Ferramenta para montagem de sequências genômicas em ambientes de memória compartilhada e distribuída com FPGAs / F. V. Almeida -- versão corr. - São Paulo, 2020.

117 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Programação Paralela 2.FPGA 3.Sequenciamento de Genoma 4.Bioinformática I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

Aos profissionais da área da saúde

AGRADECIMENTOS

À professora Liria, por me acompanhar nesta jornada de alto desempenho durante e me dar todo o suporte necessário.

Ao professor Edson, que me auxiliou durante o projeto de mestrado.

À professora Edith, que continua a me surpreender a cada dia mostrando como a paixão por ensinar é a melhor forma de ensino.

Aos meus pais que sempre me deram o apoio necessário em minha vida acadêmica.

À professora Tatiana, que me ajudou a ingressar nesta nova área.

À Erina, que caminhou comigo nas trilhas do mestrado e futuramente será a mestra do *blockchain*.

À tia Fatima e ao tio Daniel que assim como durante a graduação, continuaram a proporcionar momentos muito agradáveis durante o mestrado.

À Nathalia, pelas conversas e séries que muito ajudaram na realização deste projeto.

À Ana Maria, que me deu uma grande ajuda na reta final da escrita da dissertação.

À tia Natália que me mostrou maneiras de superar os desafios e obstáculos da vida.

À todos os meus professores que me forneceram não apenas um ideal de ensino, mas sim de valores para levar adiante.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo fomento desta pesquisa através de uma bolsa de mestrado.

*“Before I built a wall I’d ask to know
What I was walling in or walling out,
And to whom I was like to give offense.
Something there is that doesn’t love a
wall, That wants it down.”*

-- Robert Frost

RESUMO

O estudo do genoma vem sendo cada vez mais explorado pela comunidade científica como forma de entender o funcionamento não apenas do ser humano, mas de todos os seres vivos. Para tal, é necessário que exista a montagem do genoma, que é fragmentado em um processo denominado sequenciamento. A montagem do genoma possui alto custo computacional, devido ao grande volume de dados utilizados e às operações de comparação e alinhamento que precisam ser realizadas múltiplas vezes, fazendo com que o tempo de execução varie de poucos minutos a até meses, a depender da infraestrutura computacional. Este trabalho propõe uma ferramenta híbrida, utilizando a implementação de algoritmos tanto em *software* quanto em *hardware* para melhorar o desempenho da montagem do genoma. Para tal, é apresentada uma comparação de desempenho de um algoritmo base desenvolvido em diversas versões distintas. Os resultados demonstram que a implementação do algoritmo híbrido em *hardware* e *software* resulta em ganhos significativos de desempenho, apontando para as possibilidades de um maior emprego na área.

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

ABSTRACT

The genome study has been increasingly explored by the scientific community, as a way of understanding the functioning not only of humans, but of all living beings. Therefore, it is necessary to have the assembly of the genome, which is fragmented in a process called sequencing. The genome assembly has a high computational cost, due to the large figures of data used and the comparison and alignment operations that need to be performed multiple times, making the execution time vary from a few minutes to even months, depending on the computational infrastructure. This study proposes a hybrid tool, using the implementation of algorithms in both software and hardware, improving the performance of the genome assembly. For this, a performance comparison of a base algorithm developed in several different versions is presented. The results demonstrate that the implementation of a hybrid algorithm using both hardware and software results in significant performance gains, pointing out the possibilities of greater application in this field.

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

LISTA DE FIGURAS

1	Estrutura molecular das bases nitrogenadas	24
2	Sequenciador de DNA	25
3	Sequenciamento <i>shotgun</i>	26
4	Algoritmo de Needleman–Wunsch	31
5	Utilização de grafo De Bruijn para montagem de genoma	32
6	Sequenciamento do tipo <i>paired-ended</i>	32
7	Processo de sequenciamento de genoma	33
8	Processo de correção de sequências	34
9	Arquitetura do sistema de memória compartilhada	35
10	Arquitetura do sistema de memória distribuída	36
11	Tipos de <i>speedup</i>	37
12	Modelo <i>fork-join</i>	38
13	Estrutura de um componente digital em VHDL	40
14	Projeto hierárquico de hardware	41
15	<i>High-Level Synthesis</i>	47
16	Placa FPGA DE0-CV	49
17	Célula lógica da FPGA Cyclone V	50
18	Exemplo de pacote no protocolo RS-232	52
19	Circuito MAX3232	53
20	Circuito para comunicação entre CPU e FPGA com RS-232	53
21	Forma de onda correspondente ao caractere U com <i>baud rate</i> de 115.200 . .	54
22	Forma de onda correspondente ao caractere U com <i>baud rate</i> de 230.400 . .	55
23	Kit de desenvolvimento FPGA Stratix 10GX	55

24	Terasic SoC Kit	57
25	Arquitetura da ferramenta	59
26	Etapa de retirada de sequências repetidas	61
27	Deslizamento de uma sequência pequena em uma sequência grande	63
28	Validação dos resultados obtidos pela ferramenta	66
29	Diagrama de blocos em alto nível do circuito na FPGA	71
30	Circuito corretor	72
31	Soma de 4 bits em cascata e em árvore	73
32	Ambiente de memória compartilhada e distribuída com FPGAs	74
33	Teste de escalabilidade do tamanho do arquivo PacBio para versão sequencial da ferramenta	82
34	Teste de escalabilidade do tamanho do arquivo Illumina para versão sequencial da ferramenta	82
35	Teste de escalabilidade do tamanho do arquivo PacBio para versão <i>multicore</i> da ferramenta	84
36	Teste de escalabilidade do tamanho do arquivo Illumina para versão <i>multicore</i> da ferramenta	84
37	Teste de escalabilidade do tamanho do arquivo PacBio para versão para <i>cluster</i> da ferramenta	86
38	Teste de escalabilidade do tamanho do arquivo Illumina para versão para <i>cluster</i> da ferramenta	86
39	Processo de empacotamento das bases nitrogenadas	91
40	Unidade de comparação entre bases nitrogenadas	92
41	Unidade de comparação completa	92
42	Tempo total do estudo experimental	95
43	Teste de escalabilidade do tamanho do arquivo PacBio para versão para <i>multicore</i> com FPGA da ferramenta	96
44	Teste de escalabilidade do tamanho do arquivo Illumina para versão para <i>multicore</i> com FPGA da ferramenta	96

45	Armazenamento das sequências pequenas na memória periférica	101
----	---	-----

LISTA DE TABELAS

1	Formatos de codificação FASTQ	29
2	Relação entre Q -score e probabilidade de erro no sequenciamento	29
3	Comparação entre trabalhos relacionados	45
4	Artigos obtidos por base de pesquisa	45
5	Comparação entre diferentes placas FPGA	51
6	Codificação adotada para as bases nitrogenadas	88
7	Comparação dos resultados de teste das versões da ferramenta	98
8	Tempo total de execução para montagem da E. coli	99
9	Tempo total de execução para montagem da E. coli considerando resultados estimados	104

LISTA DE SIGLAS

API - *Application Program Interface*
ASCII - *American Standard Code for Information Interchange*
ASIC - *Application Specific Integrated Circuit*
BLAST - *Basic Local Alignment Search Tool*
DNA - *Deoxyribonucleic acid*
FPGA - *Field Programmable Gate Array*
GPIO - *General Purpose Input/Output*
GPU - *Graphics Processing Unit*
HDL - *Hardware Description Language*
HLS - *High-Level Synthesis*
HPS - *Hard Processor System*
IEEE - *Institute of Electrical and Electronics Engineers*
LUT - *Look-Up Table*
NCBI - *National Center for Biotechnology Information*
PacBio - *Pacific Biosciences*
PCI - *Peripheral Component Interconnect*
RNA - *Ribonucleic acid*
SIMD - *single Instruction, Multiple Data*
UART - *Universal Asynchronous Receiver-Transmitter*
VHDL - *VHSIC Hardware Description Language*
VHSIC - *Very-High-Speed Integrated Circuit*

SUMÁRIO

1	INTRODUÇÃO	18
1.1	Motivação	19
1.2	Objetivos	20
1.3	Justificativa	21
1.4	Materiais e Métodos	21
1.5	Organização do Texto	22
2	ASPECTOS CONCEITUAIS	23
2.1	Aspectos Biológicos	23
2.1.1	Bases Nitrogenadas	23
2.1.2	Cromossomos	24
2.1.3	Genoma	24
2.1.4	Sequenciadores	25
2.1.5	Sequenciamento <i>Shotgun</i>	26
2.1.6	Arquivos para Tratamento de Genomas	27
2.1.6.1	Arquivo FASTA	27
2.1.6.2	Arquivo FASTQ	28
2.1.7	Alinhamento	30
2.1.8	Tipos de Montagem de Genoma	30
2.1.8.1	Montagem Illumina	31
2.1.8.2	Montagem PacBio	33
2.2	Aspectos Computacionais	35
2.2.1	Sistemas de Memória Compartilhada	35
2.2.2	Sistemas de Memória Distribuída	35

2.2.3	Métricas de Desempenho do Paralelismo	36
2.2.4	OpenMP	37
2.2.5	MPI	38
2.2.6	VHDL	39
2.2.7	Projeto Hierárquico	40
3	TRABALHOS RELACIONADOS	42
3.1	Ferramentas de Montagem	42
3.1.1	Trimmomatic	42
3.1.2	SPAdes	43
3.1.3	Canu	43
3.1.4	MARVEL	44
3.1.5	Smartdenovo	44
3.1.6	LorDEC	44
3.1.7	Considerações das Ferramentas	45
3.2	Implementações Utilizando Aceleradores em <i>Hardware</i>	45
4	FPGAs E TÉCNICAS DE COMUNICAÇÃO	48
4.1	FPGA	48
4.2	Técnicas de Comunicação	51
4.2.1	Comunicação Serial (RS-232)	51
4.2.2	PCI <i>Express</i>	55
4.2.3	Comunicação via Barramento	56
5	FERRAMENTA DE MONTAGEM DE GENOMA	58
5.1	Arquitetura da Ferramenta	58
5.1.1	Coleta de Dados	59
5.1.2	Filtro das Qualidades das Sequências Illumina	60

5.1.3	Retirada das Sequências Repetidas	60
5.1.4	Tratamento de Qualidades das Sequências PacBio	61
5.1.5	Correção	62
5.1.6	Montagem	65
5.1.7	Validação dos Resultados	65
5.2	Arquitetura da Ferramenta <i>Multicore</i>	66
5.3	Arquitetura da Ferramenta para <i>Cluster</i>	68
5.4	Arquitetura da Ferramenta <i>Multicore</i> com FPGA	69
5.4.1	Arquitetura do <i>Software</i>	70
5.4.2	Arquitetura do <i>Hardware</i>	71
5.4.2.1	Circuito Receptor	71
5.4.2.2	Circuito Corretor	72
5.4.2.3	Circuito Transmissor	74
5.5	Arquitetura da Ferramenta para <i>Cluster</i> com FPGAs	74
6	IMPLEMENTAÇÃO E ANÁLISE DOS RESULTADOS	77
6.1	Implementação Geral da Ferramenta	77
6.1.1	Coleta de Dados	77
6.1.2	Filtro de Qualidades das Sequências Illumina	78
6.1.3	Retirada das Sequências Repetidas	78
6.1.4	Tratamento de Qualidades das Sequências PacBio	79
6.1.5	Correção	80
6.1.6	Montagem	80
6.1.7	Validação dos Resultados	80
6.1.8	Análise dos Resultados	81
6.2	Implementação da Ferramenta <i>Multicore</i>	83
6.2.1	Implementação da Ferramenta	83

6.2.2	Análise dos Resultados	83
6.3	Implementação da Ferramenta para <i>Cluster</i>	85
6.3.1	Implementação da Ferramenta	85
6.3.2	Análise dos Resultados	85
6.4	Implementação da Ferramenta <i>Multicore</i> com FPGA	87
6.4.1	Implementação da Ferramenta	87
6.4.1.1	Implementação do <i>Software</i>	90
6.4.1.2	Implementação do <i>Hardware</i>	91
6.4.2	Estimativa dos Resultados Experimentais	92
6.4.3	Análise dos Resultados	97
6.5	Implementação da Ferramenta para <i>Cluster</i> com FPGAs	97
6.6	Comparação dos Resultados	97
6.7	Estimativas de Resultados Futuros	100
6.7.1	Mudança no Modo de Comunicação	100
6.7.2	Redução no Número de Mensagens Trocadas	101
6.7.3	Aumento do Número de Circuitos de Correção	102
6.7.4	Análise de Resultados Estimados	102
7	CONCLUSÕES	105
7.1	Trabalhos Futuros	106
	Referências	108
	Apêndice A – Configuração da porta serial via <code>termios.h</code>	112
	Apêndice B – Codificação das sequências genômicas	114
	Apêndice C – Decodificação das sequências genômicas	116

1 INTRODUÇÃO

Desde o término do século XX, com o fenômeno conhecido como *powerwall*, responsável por limitar os antigos ganhos significativos anuais dos processadores, o emprego da computação paralela, ou computação de alto desempenho, vem crescendo no cenário da computação. Este paradigma de computação tem como propósito aproveitar ao máximo todos os recursos computacionais disponíveis, para otimizar algoritmos muitas vezes já existentes. Dentre algumas das técnicas de otimização utilizadas pode-se citar a adaptação para ambientes com múltiplos núcleos (*multicore*), adaptação para ambientes distribuídos (*clusters*), melhor utilização da hierarquia de memória, utilização de ambientes heterogêneos, dentre outras.

O advento da computação heterogênea, resultando no surgimento dos chamados ambientes heterogêneos, é algo recente no cenário da computação de alto desempenho. Aqui utilizam-se aceleradores para otimizar os algoritmos, como por exemplo as *Graphics Processing Units* (GPUs), as *Field Programmable Gate Arrays* (FPGAs) ou os *Application Specific Integrated Circuits* (ASICs). Com estes aceleradores é possível aumentar cada vez mais o desempenho das aplicações, o que antes não era possível apenas com as *Central Processing Units* (CPUs). O emprego crescente dos aceleradores neste caso é justificado pelo fato de se tratarem de circuitos de propósito específico, frente à CPU, que é um circuito de propósito geral. No caso da GPU, por exemplo, sua arquitetura é projetada para resolver problemas do tipo *Single Instruction, Multiple Data* (SIMD), como as operações gráficas.

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

O término do Projeto Genoma Humano em 2003 foi um divisor de águas na história da genética. Desde então as técnicas de sequenciamento de genoma vêm sendo aprimoradas, diminuindo o custo do sequenciamento e melhorando a qualidade das amostras (GOODWIN; MCPHERSON; MCCOMBIE, 2016). Porém, a necessidade cada vez maior

do estudo de diferentes organismos exige a montagem de um número crescente de genomas para sua posterior análise. Em acréscimo, mutações sequenciais existentes no ácido desoxirribonucleico (DNA) dificultam a tarefa de análise dos pesquisadores, de forma que ainda é difícil realizar um mapeamento perfeito no genoma dos organismos.

Visando melhorar esse cenário, novos sequenciadores vêm surgindo, juntamente com ferramentas computacionais responsáveis pela montagem do DNA. Isso é consequência do desenvolvimento de novos métodos tanto para o sequenciamento quanto para a análise do DNA (MISALE et al., 2014). Um destes métodos é a montagem de DNA através da correção híbrida de sequências genômicas, que será descrito posteriormente.

1.1 Motivação

Conforme já foi mencionado, a área de pesquisa genética está em amplo crescimento nos últimos anos, devido à necessidade de análise de diversos organismos. 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 poucos dias, atualmente, com um *cluster* adequado. Um dos desafios desta área é a diferença existente entre um genoma obtido através do processo de montagem do DNA e o chamado genoma de referência do organismo. Devido às mutações no DNA, é probabilisticamente impossível que dois genomas sejam 100% idênticos, porém não é fácil identificar se a diferença é proveniente de uma mutação ou de um erro de montagem ou sequenciamento. Até mesmo no caso do ser humano, gêmeos monozigóticos (gerados por um mesmo espermatozoide e óvulo) possuem diferenças em seu DNA, por consequência das mutações (BRUDER et al., 2008).

Um dos objetivos da análise dos genomas é a comparação de diferentes genomas de uma mesma espécie, com o intuito de se entender quais são as proteínas responsáveis pelas suas diferenças. O enfoque, no caso, é o ser humano.

Após o término do Projeto Genoma Humano (1990 - 2003) novas iniciativas foram lançadas para continuar o estudo do genoma do ser humano. A mais ambiciosa delas foi o Projeto 1000 Genomas (2008 - 2015), que tinha o intuito de aumentar a base de genomas humanos de referência (CONSORTIUM et al., 2010). Uma das informações mais importantes obtidas por consequência desse projeto foi a de que existe uma diferença aproximada de 20 milhões de bases nitrogenadas entre genomas de diferentes pessoas (CONSORTIUM et al., 2015). Isso resulta em um percentual de apenas 0,6% das bases.

São nessas diferenças que se encontra a chave para o entendimento e a cura das doenças. Porém, para que esta análise seja feita corretamente, é necessário garantir que a montagem não possua erros, eliminando assim uma variável de incerteza na análise de diferentes genomas de uma mesma espécie.

Para tal, vem ocorrendo uma evolução nos sequenciadores de DNA, melhorando-se o processo de extração das bases nitrogenadas a partir do material genético. Porém, novos sequenciadores exigem novas ferramentas de montagem, devido à adequação com as recentes técnicas de sequenciamento. Por isso, existe uma necessidade constante de modificação das ferramentas já desenvolvidas.

Dentre os tipos de sequenciamento mais utilizados pelos pesquisadores na atualidade destacam-se o sequenciamento Illumina e o PacBio. O primeiro é adequado para seres vivos com genoma de menor extensão, pois neste tipo de montagem são utilizadas sequências pequenas. O segundo resulta em sequências maiores, o que auxilia no processo de montagem, porém, estas sequências possuem baixa qualidade, ou seja, não é possível se inferir com certeza que as bases nitrogenadas apresentadas estão corretas. Desta forma, algoritmos de correção destas sequências são utilizados, visando aumentar a sua qualidade. Estes algoritmos possuem alto custo computacional, pois consistem basicamente em múltiplas comparações entre diversas sequências, tendo complexidade computacional proporcional à complexidade do organismo a ser montado. Por isso, este é um problema propício a constantes otimizações, sendo uma delas a utilização de aceleradores.

1.2 Objetivos

O objetivo principal desta dissertação é propor uma estratégia para a montagem através da correção híbrida de sequências genômicas, e com ela desenvolver uma ferramenta paralela para ambientes heterogêneos, sendo esta executada em CPUs e FPGAs. Este objetivo relaciona-se à hipótese de que é possível melhorar o desempenho das ferramentas existentes no estado da arte através da adição de aceleradores implementados em FPGA, como também melhorar a qualidade do genoma montado. Desta forma, pretende-se confirmar a hipótese em questão.

Além do objetivo principal, outros objetivos secundários consistem na proposição de estratégias e no desenvolvimento de ferramentas para ambientes com menos recursos computacionais. São elas a versão sequencial, versão *multicore*, versão para *cluster* e versão *multicore* com FPGA. A análise e comparação de desempenho entre elas resultam

em informações importantes para os pesquisadores da área, podendo guiá-los na concepção de ferramentas futuras.

1.3 Justificativa

Dentre as etapas necessárias para a montagem do genoma, a que possui maior custo computacional é a correção, exigindo uma quantidade de tempo que varia de minutos a até centenas de milhares de CPUs hora, a depender da complexidade do organismo e do ambiente computacional em questão (ANGEL et al., 2018).

Observa-se, na literatura presente no estado da arte, que existe um *tradeoff* nas ferramentas já existentes entre o tempo de execução necessário para a correção e a qualidade do genoma montado. Atualmente, parte da qualidade é desconsiderada para se obter um tempo de execução viável, de tal forma que nem todos os alinhamentos entre as sequências genômicas são realizados.

1.4 Materiais e Métodos

Para a realização desta dissertação foram realizadas diversas pesquisas bibliográficas, visando inicialmente o entendimento dos conceitos básicos da genômica. Em seguida foram realizadas pesquisas no estado da arte, para se identificar as ferramentas já existentes e suas carências no desempenho e/ou qualidade da montagem.

Após o término da etapa de pesquisa, foi realizada uma especificação e implementação incremental da ferramenta de montagem, com o desenvolvimento das diversas versões citadas na seção Objetivos. Iniciou-se pela versão sequencial, passando para a versão *multicore*, seguida da versão para *cluster* e por último a versão *multicore* com FPGA. A versão utilizando um *cluster* conectado com várias FPGAs não foi implementada, porém o seu desempenho foi estimado com base no levantamento de dados das outras versões.

Por último, foi realizada uma análise geral crítica, tanto no sentido de desempenho quanto na escalabilidade das versões desenvolvidas, além de uma comparação de resultados com algumas ferramentas presentes no estado da arte.

As tecnologias utilizadas neste projeto dividem-se em *software* e *hardware*. No *software* foi utilizada a linguagem C, com o OpenMP (CHAPMAN; JOST; PAS, 2007) para ambientes *multicore* e o MPI (SNIR et al., 1998) para ambientes distribuídos. No *hardware* foram utilizadas placas FPGA, responsáveis por simular o comportamento de um

circuito lógico e a VHDL como ferramenta modeladora do circuito na placa.

1.5 Organização do Texto

Esta dissertação está organizada da seguinte forma.

O capítulo 2 apresenta um levantamento bibliográfico sobre os aspectos conceituais importantes para o entendimento deste trabalho, sendo divididos em aspectos biológicos e computacionais.

O capítulo 3 resume os trabalhos relacionados, ou seja, as ferramentas já existentes no estado da arte e as situações onde já foi utilizada a FPGA no contexto da montagem do genoma.

O capítulo 4 apresenta uma descrição das FPGAs, com enfoque nos seus diferentes atributos e formas de comunicação entre CPU e FPGA que podem ser utilizadas na ferramenta.

O capítulo 5 exhibe as arquiteturas das ferramentas (principal e secundárias), mostrando suas adequações para diferentes ambientes.

O capítulo 6 apresenta as implementações das ferramentas, juntamente com seus resultados parciais.

O capítulo 7 apresenta as considerações finais e as oportunidades de trabalhos futuros relacionados ao trabalho aqui apresentado.

2 ASPECTOS CONCEITUAIS

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

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.

2.1.1 Bases Nitrogenadas

Bases nitrogenadas são compostos orgânicos com átomos de nitrogênio que compõem o DNA e o ácido ribonucleico (RNA) dos organismos. Elas podem ser classificadas como dois tipos, sendo as bases púricas e as pirimídicas, a depender da quantidade de ligações de hidrogênio que se realizam entre moléculas. Além desta classificação, elas são divididas em 5 tipos: Adenina (A), Citosina (C), Guanina (G), Timina (T) e Uracila (U), sendo estas duas últimas presentes apenas no DNA e no RNA, respectivamente (NELSON; COX, 2018). A figura 1 apresenta a estrutura molecular das bases.

No RNA mensageiro, responsável pela transferência de informações dentro da célula, uma trinca de bases nitrogenadas forma um códon, que codifica um aminoácido. Um conjunto de aminoácidos resultam em uma proteína, sendo estas fundamentais na análise de um organismo (NELSON; COX, 2018).

Figura 1: Estrutura molecular das bases nitrogenadas

Fonte: Autor

2.1.2 Cromossomos

Os cromossomos são grandes moléculas de DNA que condensam todo o genoma do organismo, juntamente com proteínas que empacotam a fita de DNA, sendo encontrados no núcleo das células eucariontes (mais complexas), ou dispersos no citoplasma das células procariontes (menos complexas) (ALBERTS et al., 2018). As células eucariontes podem ser haploides ou diploides, dependendo do número de cromossomos de cada tipo presente na célula. Os cromossomos por sua vez podem ser sexuais, responsáveis por definir o sexo do organismo, ou autossomos, responsáveis por definir as demais características.

O número de cromossomos presentes em uma célula varia conforme a complexidade do organismo. Bactérias como por exemplo a *Escherichia coli* (E. coli) possui apenas 1 molécula de DNA, enquanto os seres humanos possuem 46 cromossomos em suas células (44 autossomos + 2 cromossomos sexuais).

2.1.3 Genoma

Genoma é o nome dado para o conjunto de todos os genes do organismo, ou seja, toda a informação presente nos cromossomos. A importância do genoma está em fornecer aos pesquisadores uma visão geral do organismo, onde através da análise de suas proteínas,

obtidas pela visualização de suas bases nitrogenadas, é possível entender melhor o seu funcionamento (VOET; VOET, 2011).

2.1.4 Sequenciadores

Sequenciadores são as máquinas responsáveis pelo processo de sequenciamento do DNA. Um exemplo de sequenciador é apresentado na figura 2. Eles possuem como entrada uma amostra de material genético do organismo, e sua saída é um arquivo texto contendo as bases nitrogenadas que compõem o genoma, juntamente com a respectiva qualidade de cada base.

Figura 2: Sequenciador de DNA

Fonte: Illumina (s.d.)

A qualidade é um parâmetro de saída de cada base nitrogenada que possui relação direta com o próprio processo de sequenciamento. Nele, amostras de DNA recebem um tratamento bioquímico de tal forma que cada base emite luz em um espectro eletromagnético distinto. Essa luz então é capturada por câmeras que distinguem as bases com relação às faixas de espectro que cada uma pode emitir (PAREEK; SMOCZYNSKI; TRETYN, 2011). A qualidade aqui atua como um indicador de confiança de que aquele registro de caractere feito no arquivo de saída corresponde fielmente à base nitrogenada presente naquela região do DNA. Destaca-se que nunca existe uma situação onde a certeza de que determinada base foi sequenciada corretamente é absoluta. O que existe é uma probabilidade de erro tão baixa que se assume que aquela base foi sequenciada de maneira correta.

Os arquivos provenientes do sequenciamento são utilizados nas ferramentas de montagem de DNA, que agrupam estes fragmentos em grandes aglomerados denominados *contigs* (conjunto de fragmentos de DNA sobrepostos, resultantes de um processo de montagem). A função das ferramentas neste caso é agrupar fragmentos semelhantes, com o intuito de

recuperar o genoma do organismo. Assim, almeja-se obter o menor número possível de *contigs* no término da montagem, onde este número é limitado pelo número de cromossomos do organismo. Caso não houvesse mutações no DNA, seria possível que o número de *contigs* fosse igual ao de cromossomos na grande maioria dos casos, porém na prática isso não ocorre, sendo a quantidade de *contigs* superior.

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 pode sofrer mudanças em cada geração de sequenciadores (ZHANG et al., 2011).

2.1.5 Sequenciamento *Shotgun*

O sequenciamento *shotgun* é um tipo de sequenciamento amplamente empregado pelos sequenciadores modernos. Ele consiste em quebrar o DNA em regiões aleatórias, e possui menor custo quando comparado à quebra sequencial do DNA (BROWN, 2002).

Para tal, inicialmente o material genético do organismo é clonado através de um processo bioquímico, gerando redundância do mesmo. O sequenciador então realiza a quebra nas regiões aleatórias, gerando fragmentos aqui denominados *reads*, que estarão presentes no arquivo de saída. Este processo é ilustrado pela figura 3.

Figura 3: Sequenciamento *shotgun*

Fonte: Autor

A razão entre o material genético clonado e o original é denominada cobertura (*coverage*). É importante que a cobertura seja alta, para garantir que todas as bases nitro-

genadas do genoma estejam sequenciadas em fragmentos suficientemente grandes. Ela é representada por um multiplicador seguido de um número onde, por exemplo, um arquivo com multiplicador x10 apresenta 10 vezes mais bases nitrogenadas que o genoma do organismo. Deve-se ter um cuidado com o quão alta é a cobertura do sequenciamento, pois em determinadas situações o aumento da cobertura prejudica o processo da montagem pois aumenta o custo computacional sem contribuir efetivamente para o processo (SIMS et al., 2014).

2.1.6 Arquivos para Tratamento de Genomas

Os arquivos texto provenientes dos sequenciadores possuem diversas extensões distintas, devido a uma falta de padronização. Nesta seção serão descritas as extensões fasta e fastq, que foram utilizadas neste projeto.

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*). Neste tipo de arquivo cada sequência de bases nitrogenadas é representada por 2 linhas. A primeira linha é o cabeçalho da sequência, que se inicia com o caractere ">" seguido de um conjunto de caracteres. A segunda linha é a própria sequência, onde as bases nitrogenadas são descritas com as letras A, C, T, G. A seguir é apresentado um exemplo de arquivo fasta.

```

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

```

Um limitante na utilização deste tipo de arquivo é que são necessários dois arquivos pra realizar a montagem, onde um dos arquivos corresponde às sequências de bases e o outro às suas respectivas qualidades. Este limitante foi responsável pela padronização de outro tipo de arquivo utilizado aqui, que é o fastq (*fasta quality*).

2.1.6.2 Arquivo FASTQ

Como foi dito, o arquivo fastq surge para suprir uma carência do arquivo fasta, que é conter todas as informações do genoma sequenciado em um único arquivo. Para tal, ele utiliza 4 linhas para descrever uma sequência. A primeira linha é o cabeçalho da sequência, semelhante ao arquivo fasta, porém aqui o caractere de início é o @. A segunda linha é a própria sequência, novamente semelhante ao arquivo fasta. A terceira é o cabeçalho das qualidades, sendo em geral representado apenas pelo caractere "+". Por último, na quarta linha estão presentes as qualidades de cada base, sendo aqui utilizados caracteres específicos para determinar valores de qualidade. A seguir é apresentado um exemplo de arquivo fastq.

```

1 @seq1
2 AATTCGAGAG
3 +
4 !!##!!#!!(
5 @seq2
6 AAAAAAAAAA
7 +
8 !!#(-$-(,)
9 @seq3
10 TCGATGGACG
11 +
12 )!##!!(-),(

```

Conforme observado, o arquivo fastq consegue suprir a carência do arquivo fasta, porém ainda existe um outro problema, que é a ausência de uma padronização para definir as qualidades (COCK et al., 2009).

Existem diferentes formas de codificação para as qualidades, e cada forma trabalha com diferentes faixas de caracteres ASCII (*American Standard Code for Information Interchange*). Por isso, é necessário que a ferramenta de montagem inicialmente identifique qual o tipo de codificação utilizada, antes de realizar a montagem. Caso contrário, é possível que a ferramenta infira que determinada sequência possui boa qualidade geral, enquanto sua qualidade é baixa, caso seja utilizada uma codificação errada. A tabela 1 apresenta os tipos de codificação mais utilizados para as qualidades, com suas respectivas amplitudes e valores ASCII.

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

Os valores apresentados na coluna de amplitude da tabela 1 são definidos como a pontuação da qualidade *Q-score*. A relação entre o *Q-score* da base nitrogenada e a probabilidade da mesma ter sido sequenciada erroneamente é dada pela seguinte equação:

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

Realizando-se a substituição de alguns valores distintos de *Q-score* na equação anterior, obtém-se a tabela 2:

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

<i>Q-score</i>	P(erro)
10	0.1
20	0.01
30	0.001
40	0.0001

Fonte: Autor

Novamente devido à ausência de uma padronização, não existe um valor padrão de *Q-score* para poder se inferir que determinada base foi sequenciada erroneamente. Portanto, é importante que este parâmetro seja flexível nas ferramentas de montagem, de forma que o usuário possa modificá-lo na medida de sua necessidade. Indo ao encontro do que foi dito durante a explicação dos sequenciadores, também é importante observar que o valor mais alto de *Q-score* resulta em uma base com a probabilidade de erro de 0,01% de chance de ter sido sequenciada erroneamente.

2.1.7 Alinhamento

Um dos principais desafios no processo de montagem do genoma é a obtenção da similaridade entre sequências de bases nitrogenadas. Para tal, é necessário realizar inúmeros alinhamentos entre duas ou mais sequências inteiras ou suas regiões. Dentre os algoritmos desenvolvidos inicialmente para este propósito, um dos que possuíram maior destaque e ampla aplicação foi o algoritmo de Needleman-Wunsch (NEEDLEMAN; WUNSCH, 1970). Este algoritmo utiliza-se de computação dinâmica para montar uma tabela, considerando as diferentes possibilidades de alinhamento entre as sequências, sendo uma das primeiras aplicações da computação dinâmica para comparar sequências genômicas (SETUBAL; MEIDANIS, 1997).

Tal algoritmo recebe duas sequências genômicas, podendo estas serem apresentadas em nível de base nitrogenada ou aminoácido, quando então 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 se obter o alinhamento ideal das sequências. Esse processo é ilustrado pela figura 4.

Logo após, surgiu o *Basic Local Alignment Search Tool* (BLAST), atualmente mantido pelo *National Center for Biotechnology Information* (NCBI) (BLAST, 2020). 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.

No contexto do alinhamento de sequências genômicas é necessário realizar aqui uma observação para não ocorrer o equívoco de relacionar a situação aqui descrita com o problema de localizar uma *substring* em uma *string*. Algoritmos como por exemplo o Knuth–Morris–Pratt (KMP) são otimizados para esta busca de *substring* (RÉGNIER, 1989), porém não possuem bom comportamento neste contexto devido ao fato que o alinhamento não é perfeito.

2.1.8 Tipos de Montagem de Genoma

Os sequenciadores de DNA são desenvolvidos por diversas empresas, onde cada uma desta empresa utiliza uma tecnologia distinta, resultando em diferentes processos de sequencia-

Figura 4: Algoritmo de Needleman–Wunsch

Fonte: (NEEDLEMAN; WUNSCH, 1970)

mento. Duas das empresas que possuem maior destaque na área de sequenciamento de DNA são a Illumina e a *Pacific Biosciences* (PacBio), sendo que cada uma utiliza uma ferramenta diferente para a montagem, devido às especificidades do sequenciamento. A seguir serão descritas as diferenças entre as montagens.

2.1.8.1 Montagem Illumina

A montagem Illumina é realizada após o sequenciamento Illumina. Este tipo de montagem também é conhecida como montagem de sequências pequenas (*short reads assembly*), devido ao fato de que as sequências provenientes deste tipo de sequenciamento são pequenas e com boa qualidade (média de *Q-score* variando entre 30 e 40). Aqui as sequências costumam possuir tamanho aproximado de 100 a 250 bases nitrogenadas.

O método de montagem consiste em se construir grafos De Bruijn, conforme ilustrado na figura 5. As sequências são quebradas em fragmentos menores, denominados *k-mers*, para realizar a montagem do grafo.

Figura 5: Utilização de grafo De Bruijn para montagem de genoma

Fonte: Autor

Existem dois limitantes deste tipo de montagem. O primeiro deles é para organismos com grande complexidade genômica. Pelo fato das sequências serem pequenas, o custo computacional para se montar todas as sequências torna-se inviável. O segundo é quando o organismo a ser montado possui muita redundância em seu genoma, gerando situações de ambiguidade, onde não é possível afirmar com precisão onde determinada sequência deve ser montada.

Uma característica específica das sequências do tipo Illumina é consequência do seu sequenciamento que pode ser de dois tipos. São eles o *single-read* e o *paired-ended*.

O sequenciamento do tipo *single-read* é o mais simples e intuitivo. Neste, o arquivo de saída do sequenciador possui um conjunto de sequências onde cada sequência corresponde a uma região contínua do DNA do organismo.

O sequenciamento do tipo *paired-ended* surgiu posteriormente e tem como propósito sequenciar simultaneamente duas extremidades de uma mesma sequência de DNA, conforme apresentado na figura 6.

Figura 6: Sequenciamento do tipo *paired-ended*

Fonte: Illumina (s.d.)

A ideia deste tipo de sequenciamento é auxiliar no processo da montagem Illumina. Dado que a região cinza da figura, separando as regiões laranja e azul que são conhecidas, possui tamanho fixo, é possível realizar um mapeamento do genoma do ser vivo a partir de alinhamentos sucessivos. A implicação prática deste novo tipo de sequenciamento nas ferramentas de montagem é que estas precisam identificar qual o tipo do sequenciamento Illumina que resultou no arquivo de entrada da ferramenta, pois o tratamento das sequências em questão é diferente. Dada a saída do sequenciador que realiza o sequenciamento do tipo *paired-ended*, as duas extremidades sequenciadas podem estar presentes em um mesmo arquivo de saída ou então estarem em arquivos de saída distintos.

2.1.8.2 Montagem PacBio

A montagem PacBio difere da montagem Illumina, pois suas sequências são maiores, porém possuem baixa qualidade (média de *Q-score* variando entre 1 e 10). Aqui o tamanho das sequências é de aproximadamente 10.000 bases nitrogenadas (RHOADS; AU, 2015); devido à tecnologia diferente utilizada, e elas são chamadas de sequências grandes (*long reads*).

Pelo fato das sequências aqui serem maiores, durante o alinhamento é realizada uma comparação entre os extremos de tais sequências, onde a pontuação obtida neste alinhamento também auxilia na montagem de grafos. Seguindo o caminho de maior pontuação no grafo é possível se obter os *contigs*. Esse processo é ilustrado pela figura 7.

Figura 7: Processo de sequenciamento de genoma

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

O problema neste tipo de montagem está na baixa qualidade das sequências, o que pode resultar em montagens erradas (RHOADS; AU, 2015). Por isso é necessário realizar um processo de correção nas sequências provenientes do sequenciador, visando melhorar a qualidade geral do arquivo.

A correção das sequências PacBio podem ser de dois tipos; são elas a autocorreção (*self*) ou a correção híbrida (*hybrid*). A autocorreção consiste em se utilizar o próprio arquivo contendo as sequências PacBio para se auto corrigir. Isso é feito através de alinhamentos múltiplos entre as sequências, com o intuito de se obter consensos nestes alinhamentos. Um empecilho deste tipo de correção é que caso as sequências tenham uma qualidade muito baixa, tal consenso torna-se mais difícil de ser obtido, de forma que a correção pode não ocorrer.

Na correção híbrida é utilizado um outro arquivo junto com o arquivo PacBio, contendo sequências com melhor qualidade provenientes de outra tecnologia (em geral Illumina). Ela se baseia no fato que ambos os arquivos devem ser provenientes do mesmo material genético, refletindo então o mesmo genoma. Assim, é possível, através da comparação entre ambas as sequências, corrigir a sequência grande com a pequena, conforme ilustrado pela figura 8.

Figura 8: Processo de correção de sequências

Fonte: (FLOWERS; CREWS, 2018)

Tanto a autocorreção quanto a correção híbrida são compostas por três tipos básicos de operação. São elas a adição, a remoção e a substituição de bases nitrogenadas. Após

o processo de correção, a sequência corrigida pode ser maior, menor ou possuir mesmo tamanho que a sua versão não corrigida.

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

2.2 Aspectos Computacionais

Nesta seção são apresentados os aspectos computacionais necessários para o entendimento deste trabalho. É dado foco especial nos aspectos relativos à computação paralela e ao projeto de *hardware*.

2.2.1 Sistemas de Memória Compartilhada

Sistemas de memória compartilhada são ambientes que possuem um ou mais núcleos, conectados a uma mesma memória através de um barramento comum, conforme ilustrado pela figura 9 (HENNESSY; PATTERSON, 2012). Neste tipo de sistema a comunicação através dos núcleos pode ser realizada com o auxílio de variáveis compartilhadas, e o sincronismo feito através de semáforos. Um exemplo de *framework* para este tipo de sistema é o OpenMP, que permite a programação em C utilizando todos os núcleos de uma máquina.

Figura 9: Arquitetura do sistema de memória compartilhada

Fonte: Autor

2.2.2 Sistemas de Memória Distribuída

Sistemas de memória distribuída são ambientes compostos por múltiplos sistemas computacionais, interconectados através de um barramento de alta velocidade, conforme

ilustrado na figura 10 (HENNESSY; PATTERSON, 2012). Eles costumam ser divididos em nós, onde cada nó é uma unidade independente dentro do ambiente. Para estabelecer a comunicação entre os múltiplos nós neste tipo de ambiente é necessária a utilização de protocolos de troca de mensagem. Um exemplo de *framework* que implementa um protocolo deste tipo é o OpenMPI.

Figura 10: Arquitetura do sistema de memória distribuída

Fonte: Autor

2.2.3 Métricas de Desempenho do Paralelismo

Dentre as métricas de desempenho que são utilizadas no contexto da programação de alto desempenho para avaliar o impacto do paralelismo na implementação de um algoritmo, destacam-se aqui o *speedup* (S) e a eficiência (E).

O *speedup* (S) é a razão entre o tempo de processamento da versão sequencial e o tempo de processamento da versão paralela de um algoritmo, conforme equação:

$$S = \frac{t_{sequencial}}{t_{paralelo}} \quad (2.2)$$

Ele permite analisar os impactos no desempenho do algoritmo resultantes do uso de múltiplos processadores. Existem três casos de *speedup* obtidos através da aplicação do paralelismo. São estes:

- *Speedup*: Este é o caso convencional, que não possui um nome característico. Nele, a adição de mais processadores não promove uma melhoria proporcional no tempo de execução, ou seja, o paralelismo não é explorado plenamente.
- *Speedup* linear: Neste caso 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 que se obtenha o mesmo em todas as situações devido a atrasos como o *overhead* de comunicação.

- *Speedup* super-linear: Nesse caso a adição de processadores promove uma melhoria superior à esperada pela proporção de recursos (exemplo: o dobro de processadores resulta em uma queda no tempo de execução por um fator três). A justificativa para tal encontra-se em aspectos internos do processador, como na melhor utilização de caches de memória, resultando em um ganho em operações de leitura e escrita de dados.

A figura 11 ilustra graficamente os tipos de *speedup*.

Figura 11: 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, conforme equação apresentada a seguir. Sua função é explicitar o quanto de paralelismo é explorado no programa, onde valores próximos de 1 indicam um melhor aproveitamento do paralelismo.

$$E = \frac{S}{n_{\text{processadores}}} = \frac{t_{\text{sequencial}}}{t_{\text{paralelo}} * n_{\text{processadores}}} \quad (2.3)$$

2.2.4 OpenMP

OpenMP é uma *Application Program Interface* (API) desenvolvida com o propósito de facilitar a programação em ambientes de memória compartilhada, utilizando o paralelismo em nível de *threads*. Ao contrário de outras APIs como por exemplo a Pthreads, a OpenMP torna transparente alguns parâmetros de funcionamento do algoritmo, como

por exemplo o comportamento individual de cada *thread*, fazendo com que seu uso seja mais simples.

Ela é composta por diretivas de compilação, sendo que os seus comandos se iniciam com a sintaxe *pragma omp*. Devido ao uso destas diretivas, o compilador utilizado no código precisa ter suporte ao OpenMP para o seu correto funcionamento. Compiladores mais utilizados, com o caso do *gcc* na linguagem C, possuem suporte nativo. Além desta, o OpenMP também pode ser utilizado nas linguagens C++ e Fortran (CHAPMAN; JOST; PAS, 2007).

O funcionamento das *threads* no OpenMP ocorre mediante um modelo do tipo *fork-join*. Nele, uma *thread* principal ou *thread* mestra realiza o *fork*, gerando uma série de *threads* paralelas, que podem executar um mesmo bloco de código ou blocos de código distintos. Após o término da execução, tais *threads* são unidas por meio do *join*, finalizando assim a execução paralela. A figura 12 ilustra esse processo.

Figura 12: Modelo *fork-join*

Fonte: Autor

2.2.5 MPI

A sigla MPI significa *Message-Passing Interface*, ou seja, Interface de Passagem de Mensagem. O MPI pode ser descrito como um padrão de comunicação de dados, que é implementado a partir de uma biblioteca composta por diversas funções que permitem a comunicação em sistemas de memória distribuída, e por isso costuma ser empregado em *clusters* (SNIR et al., 1998). Uma de suas implementações mais conhecidas é o OpenMPI, que implementa o MPI em uma biblioteca com suporte para as linguagens C, C++ e Java.

Dado um nó de um *cluster*, este pode enviar mensagens para outros nós por meio do

MPI, sendo necessário apenas o conteúdo e o endereço do destinatário. As mensagens podem ser do tipo *unicast* (apenas um destinatário), *multicast* (múltiplos destinatários) ou *broadcast* (envio de mensagem para todos os nós da rede). Além disso, sua comunicação pode ser do tipo bloqueante, onde o remetente só continua sua execução quando o destinatário recebe a mensagem ou não bloqueante, quanto o remetente dispara a mensagem e continua seu fluxo de execução sem aguardar o recebimento daquela.

2.2.6 VHDL

VHSIC Hardware Description Language ou *Very High Speed Integrated Circuits Hardware Description Language* (VHDL) é uma das linguagens de descrição de hardware disponíveis para este propósito. Além dela, também existe Verilog, SystemVerilog, AHDL dentre outras. Ela foi desenvolvida pelo Departamento de Defesa dos Estados Unidos na década de 1980. Sendo utilizada inicialmente apenas com o propósito de documentação de componentes, ela foi padronizada pelo *Institute of Electrical and Electronics Engineers* (IEEE), em 1987, resultando em um aumento da sua aplicação para outras finalidades (PEDRONI, 2004).

A estrutura básica de um componente digital descrito em VHDL é dividida em três partes. São elas:

- Biblioteca/pacote (*library/package*): Conjunto de constantes, tipos de variáveis e funções que poderão ser utilizadas pelo componente.
- Entidade (*entity*): Descrição externa do componente digital, representando aqui uma caixa preta. Nela são descritas apenas as entradas e as saídas do componente em questão.
- Arquitetura (*architecture*): Descrição interna do componente, apresentando sua arquitetura. Aqui são definidos os comandos lógicos segundo a sintaxe da linguagem que permitem obter as saídas do componente a partir de suas entradas.

A figura 13 apresenta a descrição de um componente digital em VHDL, sendo este uma porta do tipo "E" (porta AND). Na entidade são descritas suas duas entradas como sendo A e B, e a sua saída como sendo Y. Na arquitetura descreve-se sua estrutura de tal forma que a saída Y depende de uma operação lógica do tipo "E" entre as entradas A e B.

Figura 13: Estrutura de um componente digital em VHDL

<pre>library ieee; use ieee.std_logic_1164.all;</pre>	PACOTE
<pre>entity porta_and is port(A : in std_logic; B : in std_logic; Y : out std_logic); end entity;</pre>	ENTIDADE
<pre>architecture exemplo of porta_and is begin Y <= A and B; end exemplo</pre>	ARQUITETURA

Fonte: Autor

2.2.7 Projeto Hierárquico

O método de projeto hierárquico é comumente aplicado tanto no contexto de projeto de software quanto no projeto de hardware. Neste tipo de método busca-se reduzir a dificuldade de um projeto através do modelo "dividir para conquistar", onde a sua dificuldade global é diluída através de sua estruturação em blocos. Estes blocos então podem ser divididos em mais blocos, até o momento em que a granularidade esteja adequada para o projetista.

No caso do projeto de hardware, este método é amplamente utilizado, devido ao provimento de comandos na sintaxe de linguagens de descrição de hardware que permitem a sua aplicação. O VHDL por exemplo apresenta a possibilidade de instanciar componentes descritos anteriormente em um mesmo projeto utilizando a palavra reservada *component*. Esses componentes podem então ser acessados através de uma chamada do tipo *port map*, que representa aqui um mapeamento das entradas e das saídas do componente em questão. É utilizada a nomenclatura *top-level* para representar o componente que está localizado no topo da hierarquia do projeto. A figura 14 ilustra este método em um projeto de

hardware

Figura 14: Projeto hierárquico de hardware

Fonte: Autor

3 TRABALHOS RELACIONADOS

O capítulo apresenta os trabalhos relacionados encontrados através da pesquisa bibliográfica no estado da arte. Estes trabalhos são divididos em ferramentas para a montagem, sendo elas implementadas apenas em *software*, e utilizações pontuais de aceleradores implementados em FPGA para melhorar o desempenho de partes da montagem.

3.1 Ferramentas de Montagem

Aqui serão apresentadas as ferramentas utilizadas pelos pesquisadores da área. Estas ferramentas são aplicadas em diferentes situações, havendo pouca inter-relação entre as mesmas. No final da seção é apresentado um resumo das ferramentas, comparando-as com a ferramenta principal aqui desenvolvida.

3.1.1 Trimmomatic

A ferramenta Trimmomatic (BOLGER; LOHSE; USADEL, 2014) foi projetada com o propósito de melhorar a qualidade das sequências Illumina. Conforme foi dito, as sequências Illumina possuem uma qualidade geral boa, porém isso não impede que existam erros durante o processo de sequenciamento, resultando em regiões ou até mesmo sequências inteiras com baixa qualidade.

Para melhoria da qualidade, a ferramenta recebe como entrada um arquivo com extensão fastq, contendo um conjunto de sequências Illumina provenientes do sequenciador. Junto com o arquivo pode ser utilizada uma série de parâmetros adicionais, para auxiliar no processo de melhoria da qualidade geral do arquivo. Este processo consiste na poda ou na retirada de sequências inteiras, caso seja observado que sua qualidade média é menor que a aceitável pelos parâmetros executados com a ferramenta. Por se tratar de sequências Illumina, ela também possui dois modos de execução distintos, sendo um deles para sequências provenientes do sequenciamento *single-read* e o outro para *paired-ended*.

A importância da ferramenta Trimmomatic para o contexto deste trabalho foi facilitar o entendimento do processo de melhoria geral da qualidade das sequências Illumina. Isso permitiu uma implementação própria de um processo semelhante, que foi utilizado em uma das etapas iniciais da ferramenta que é apresentada no capítulo 5.

3.1.2 SPAdes

SPAdes (BANKEVICH et al., 2012) é a ferramenta oficial da Illumina para realizar a montagem de suas sequências. Por se tratar de uma ferramenta oficial da empresa, ela recebe atualizações constantes, visando melhorar o desempenho e correções de eventuais erros. Sua versão atual é a 3.14.1 (2020).

Ela realiza a quebra das sequências em k-mers e sua montagem com os grafos De Bruijn. Este processo leva um tempo maior que a montagem com a tecnologia PacBio, porém, não necessita de uma etapa de correção, devido à boa qualidade das sequências.

3.1.3 Canu

Canu (KOREN et al., 2017) é uma ferramenta baseada em outra ferramenta mais antiga chamada Celera Assembler. Seu propósito é a montagem de sequências PacBio, realizando a autocorreção. Na mesma, o processo de montagem é realizado por um *pipeline*, sendo o mesmo feito em três etapas descritas a seguir.

1. Correção - Na etapa de correção a ferramenta realiza a autocorreção das sequências PacBio. Para tal, é fornecido um arquivo com extensão fastq e sua saída é um arquivo com extensão fasta, pois após a correção as informações das qualidades não são mais necessárias. A autocorreção aqui é feita a partir de alinhamentos múltiplos, com o propósito de se obter regiões de consenso em diversas sequências. Esta etapa possui o custo computacional mais elevado dentre as três.
2. Corte - Na etapa de corte a ferramenta realiza a retirada das sequências que são mais destoantes das demais. Isso ocorre porque caso a diferença entre uma sequência seja muito maior que um conjunto de outras, esta sequência não irá contribuir para o processo de montagem. Por se tratar apenas de retirada de sequências, esta etapa possui baixo custo computacional.
3. Montagem - Na etapa de montagem ocorre a união das sequências corrigidas, resultando no genoma do organismo. Aqui o arquivo de entrada possui extensão fasta,

assim como o arquivo de saída. Esta etapa possui custo computacional médio.

Uma característica importante do Canu no âmbito desta pesquisa é que cada uma de suas etapas pode ser executada individualmente. Assim, é possível realizar a correção em uma ferramenta própria e em seguida finalizar a montagem com o Canu.

3.1.4 MARVEL

A ferramenta MARVEL (GROHME et al., 2018), (NOWOSHILOW et al., 2018) foi projetada para realizar a montagem através da correção híbrida das sequências PacBio. Seu *pipeline* de montagem é dividido em 7 etapas, onde todas as etapas são executadas em sequência. Por isso, ao contrário da Canu, não é possível acoplar outras ferramentas à mesma.

Por ter sido desenvolvida em 2018, sua documentação é reduzida, quando comparada às outras ferramentas aqui mencionadas.

3.1.5 Smartdenovo

Smartdenovo (RUAN, 2015) é uma ferramenta projetada com o propósito de realizar apenas a montagem das sequências PacBio. Devido ao fato dela não realizar nenhum tipo de correção nas sequências, é necessário que o arquivo de entrada desta ferramenta contenha as sequências já corrigidas, caso contrário, a montagem do genoma do organismo terá alta taxa de erro. Na pior hipótese, a montagem não consegue ser realizada, devido à ausência de um consenso de quando é feito o alinhamento entre as sequências.

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

3.1.6 LorDEC

LorDEC (Long Read DBG Error Correction) (SALMELA; RIVALS, 2014) é uma ferramenta projetada com o propósito de realizar apenas a correção híbrida das sequências PacBio. Nela, a correção é feita com o auxílio de um arquivo contendo as sequências Illumina provenientes do mesmo material genético. As sequências Illumina são montadas em um grafo De Bruijn, sendo que este é obtido através da comparação entre as sequências e o genoma de referência do organismo.

Esta ferramenta apresentou desempenho superior quando comparada com as outras que possuem o mesmo propósito (SALMELA; RIVALS, 2014). Isso é justificado pelo fato de que ela utiliza o genoma de referência do organismo como suporte.

3.1.7 Considerações das Ferramentas

A tabela 3 apresenta uma análise comparativa entre os trabalhos relacionados e a ferramenta aqui desenvolvida.

Tabela 3: Comparação entre trabalhos relacionados

Trabalho	Poda Illumina	Montagem Illumina	Correção PacBio	Montagem PacBio	Implementação
Trimmomatic	Sim	Não	Não	Não	<i>Software</i>
SPAdes	Não	Sim	Não	Não	<i>Software</i>
Canu	Não	Não	Auto (self)	Sim	<i>Software</i>
MARVEL	Não	Não	Híbrida	Sim	<i>Software</i>
Smartdenovo	Não	Não	Não	Sim	<i>Software</i>
LorDEC	Não	Não	Híbrida (com referência)	Não	<i>Software</i>
Ferramenta	Não	Não	Híbrida	Não	<i>Software/Hardware</i>

Fonte: Autor

3.2 Implementações Utilizando Aceleradores em *Hardware*

Inicialmente foi realizada uma busca em grandes bases indexadoras de artigos científicos utilizando a *query* de pesquisa "*FPGA and 'hybrid assembly'*". A *query* em questão mostrou-se ineficiente pois a parcela majoritária dos artigos apresentados não era ligados a área da bioinformática. Desta forma, optou-se por adicionar a palavra "genoma", resultando na *query* "*FPGA and 'hybrid assembly' and genome*". A tabela 4 apresenta o número de artigos obtidos por base de pesquisa.

Tabela 4: Artigos obtidos por base de pesquisa

Base de Pesquisa	nº de artigos
Elsevier	3
IEEE	1
ACM	0
Springer	0

Fonte: Autor

Dentre os quatro artigos apresentados nenhum tinha aderência com o que é proposto neste trabalho, ou seja, não foi encontrado na literatura nenhuma implementação ou aplicação da FPGA no contexto completo da correção híbrida. A partir de uma busca mais ampla na literatura, utilizando a *query* "*FPGA and genome*", observou-se que o foco dos trabalhos é a implementação de técnicas de alinhamento na FPGA.

Em (MAHRAM; HERBORDT, 2012) é apresentada uma implementação em FPGA de um circuito capaz de realizar um alinhamento múltiplo entre sequências, que é um outro problema explorado pela Genômica. (VARMA et al., 2013) apresenta uma ferramenta para montagem de sequências Illumina utilizando a comunicação entre CPU e FPGA com o barramento PCI Express. Em (RAMACHANDRAN et al., 2015) foi realizada uma comparação entre o circuito desenvolvido na FPGA para melhoria na qualidade das sequências Illumina e o *software*, mostrando que a implementação em FPGA gera um *speedup* significativo. (HU; GEORGIU, 2015) apresenta um circuito auxiliar implementado na FPGA para melhoria do desempenho do processo de montagem das sequências Illumina.

(SALAMAT; ROSING, 2020) apresenta um *survey* sobre o uso das FPGAs no contexto do alinhamento de sequências genômicas. É possível observar ganhos significativos de desempenho quando comparada-se uma versão do algoritmo em *software* e sua mesma implementação na FPGA. Outro *survey* é apresentado em (BESTA et al., 2019), sendo este relacionado aos desafios do processamento de grafos na FPGA. Por mais que este não tenha foco principal na genômica, o uso de grafos é algo recorrente nas ferramentas de correção e de montagem de sequências genômicas.

Por fim, é importante destacar que existe uma predominância na literatura no uso de técnicas de HLS (*High-Level Synthesis*). Estas técnicas têm como propósito reduzir o *overhead* de projeto do *hardware*, permitindo ao projetista gerar um circuito lógico a partir de uma linguagem de programação, como por exemplo a linguagem C, conforme ilustrado na figura 15. Para tal são utilizadas arquiteturas específicas para este propósito, como é o caso do OpenCL (STONE; GOHARA; SHI, 2010) juntamente com compiladores específicos. O uso da HLS não foi explorado neste trabalho pois a mesma resulta numa portabilidade de função, mas não de desempenho. Desta forma, circuitos gerados a partir da HLS costumam ter desempenho inferior quando comparados com circuitos gerados a partir de uma HDL (Hill et al., 2015). Porém, esta pode ser explorada em trabalhos futuros.

Figura 15: *High-Level Synthesis*

Fonte: Autor

4 FPGAS E TÉCNICAS DE COMUNICAÇÃO

O capítulo apresenta uma descrição sobre FPGAs, com enfoque em seus atributos e as formas distintas de comunicação entre CPU e FPGA que podem ser utilizadas na ferramenta.

4.1 FPGA

FPGAs são dispositivos semicondutores compostos por uma matriz de células lógicas programáveis, sendo elas por sua vez compostas por *Look-Up Tables* (LUTs) e Flip-flops. A função de uma FPGA é simular o comportamento de um circuito lógico qualquer, através da programação de suas células lógicas. Estas funcionam em conjunto com as linguagens de descrição de *hardware*, onde o código descrito em uma HDL gera um *bitstream*, responsável por preencher as LUTs com a função lógica desejada. Em geral as FPGAs são utilizadas com dois propósitos.

O primeiro deles é para prototipação, onde é aproveitada a velocidade de programação, testes e depuração do circuito lógico implementado na FPGA. O objetivo final aqui é a implementação do circuito lógico em outra tecnologia, como um ASIC por exemplo. Costuma-se ter esse propósito quando o circuito projetado vier a ser desenvolvido em larga escala.

O segundo é para o desenvolvimento de um produto final. Caso o circuito lógico seja empregado em pequena escala, em determinadas situações é utilizada a própria FPGA simulando o comportamento do circuito, ao invés de sua implementação com outra tecnologia.

Um ponto importante é a diferença entre a FPGA e o ASIC. Como foi dito anteriormente, o circuito implementado em uma FPGA pode ser fabricado futuramente com um ASIC. A vantagem aqui é que o circuito implementado no ASIC é específico; por isso o mesmo pode conter componentes adicionais para otimizar a lógica do circuito, como

por exemplo utilizar um *clock* mais rápido que o existente na FPGA. A desvantagem está no fato do circuito ser específico, de forma que ele não pode ser reprogramado como na FPGA. Ou seja, caso exista algum problema na implementação, torna-se mais difícil resolvê-lo. Além disso, a fabricação de um ASIC é cara, podendo chegar à ordem de centenas de milhares de dólares.

As FPGAs são geralmente vendidas com uma interface externa, permitindo ao projetista um fácil acesso às mesmas. O conjunto FPGA com interface externa é chamado de placa FPGA, conforme apresentado na figura 16. Observa-se que a placa contém uma série de periféricos, como leds, botões, chaves, displays hexadecimais, pinos *General Purpose Input/Output* (GPIO), memória externa, interfaces de comunicação, dentre outros.

Figura 16: Placa FPGA DE0-CV

Fonte: Terasic (s.d.)

Uma das interfaces de comunicação da placa é a responsável pela carga do *bitstream* na FPGA. Para tal, é necessário um ambiente de programação específico do fabricante da FPGA. As duas maiores fabricantes são a Intel (ex Altera) que possui o Quartus e a Xilinx, que possui o Vivado. Ambas as fabricantes possuem linhas específicas de FPGAs com diferentes atributos, com aplicações, desde acadêmicas a defesa e alto desempenho.

O advento da aplicação das FPGAs vem surgindo no contexto da computação heterogênea, com o emprego de aceleradores implementados em *hardware*. Em 2018 a Intel lançou um processador híbrido utilizando o Xeon e a FPGA Arria 10, prometendo ganhos significativos de desempenho (HUFFSTETLER, 2018). Por isso, estima-se que o emprego das FPGAs na computação de alto desempenho tenda a aumentar nos próximos anos, de forma que os computadores comerciais futuros terão CPU, GPU e FPGA. Um

dos benefícios do seu uso é o baixo consumo energético delas quando comparadas com a CPU. Trabalhos como o de (QASAIMÉH et al., 2019) mostram como o seu emprego pode resultar em quedas significativas no custo energético de uma aplicação.

Dentre os diversos atributos da FPGA, três possuem maior importância no projeto de um circuito lógico implementado na mesma. São eles a frequência do *clock*, a memória lógica programável e a memória comum.

A frequência do *clock* é o atributo mais simples e intuitivo. Este possui relação direta com os circuitos sequenciais, onde a frequência é a taxa de transições que o circuito pode ter em 1 segundo. Como os circuitos implementados na FPGA possuem quase que em sua totalidade uma parte sequencial, a frequência do *clock* então passa a ter forte relação com o desempenho do circuito. A grande maioria das FPGAs possui *clock* com frequências de 25 MHz a até 500 MHz.

A memória lógica programável é o conjunto de células lógicas que a FPGA possui. Conforme dito anteriormente, elas são as responsáveis pela simulação lógica do circuito implementado. Com o aumento da complexidade do circuito em questão, também aumenta-se a quantidade de células lógicas utilizadas, de forma que esse atributo é o principal limitante da implementação de circuitos na FPGA. Cada família de FPGAs possui uma célula lógica contendo um LUT com diferente número de entradas e um número variado de flip flops. No caso da Cyclone V da Intel, por exemplo, sua célula lógica possui uma LUT adaptada com 8 entradas e 4 flip flops, conforme apresentado na figura 17.

Figura 17: Célula lógica da FPGA Cyclone V

Fonte: (INTEL, 2018)

A memória comum também chamada de *memory bits* é um banco de armazenamento

de bits que a FPGA possui. Ao contrário da memória externa existente em diversas placas FPGA, a memória comum não possui tamanho de palavra nem endereçamento fixo, podendo o projetista criar uma memória qualquer desde que seja respeitado o número máximo de bits.

A tabela 5 apresenta uma comparação entre os três atributos citados anteriormente para quatro FPGAs, sendo as três primeiras mais simples e a última para projetos de alto desempenho. Cabe ressaltar aqui que cada família de FPGAs possui diferentes linhas, podendo haver razoável variação de tais atributos, além disso, a célula lógica possui variações ao longo das famílias das FPGAs, de tal forma que 1 célula lógica pode gerar um circuito de complexidade variável, a depender da família em questão.

Tabela 5: Comparação entre diferentes placas FPGA

Placa FPGA	FPGA	Clock (MHz)	Lógica Programável	Memória (Kb)
DE0-CV	Cyclone V	50	18.480	3.080
DE2-115	Cyclone IV	50	114.480	3.888
Cyclone 10 GX Development Kt	Cyclone 10 GX	50	80.330	11.740
Stratix 10 SX SoC Development Kit	Stratix 10	150	207.360	49.000

Fonte: Autor

4.2 Técnicas de Comunicação

Esta seção descreve algumas formas possíveis de se realizar a comunicação entre a CPU e a FPGA, visando à elaboração de um ambiente heterogêneo, semelhante ao que foi projetado na nova linha Xeon da Intel. Um aspecto positivo da comunicação com uma placa FPGA quando comparado com a comunicação com uma CPU ou uma GPU é que a placa FPGA não necessita de um tipo padronizado de barramento. Seus pinos GPIO podem ser utilizados para comunicação sem qualquer padrão, podendo o projetista estabelecer o seu próprio, caso exista a necessidade. Aqui será dada maior ênfase na comunicação serial utilizando o protocolo RS-232, que é utilizado na implementação da ferramenta devido à sua maior simplicidade e escopo frente aos outros modos.

4.2.1 Comunicação Serial (RS-232)

A comunicação serial utilizando o protocolo RS-232 é do tipo assíncrona, ou seja, não existe o compartilhamento de um *clock* comum para sincronização. Desta forma, seu modelo de comunicação utiliza-se de bits específicos, tanto no início quanto no final da

transmissão de um pacote para promover a sincronização.

Aqui o pacote é composto por 1 bit de início (*start bit*), 7 ou 8 bits de dados, 1 bit de paridade e 1 ou 2 bits de parada (*stop bits*). A figura 18 ilustra um exemplo de pacote. Ressalta-se aqui que o bit de paridade é utilizado apenas para identificar um erro na recepção de um pacote, não sendo suficiente para corrigi-lo. O circuito responsável por empacotar os bits correspondentes ao dado e transmiti-los conforme o protocolo em questão é a *Universal Asynchronous Receiver Transmitter* (UART).

Figura 18: Exemplo de pacote no protocolo RS-232

Fonte: Autor

Além da UART implementada na FPGA, é necessária uma interface na CPU para que se estabeleça a comunicação em ambas as partes. Pode-se utilizar aqui um terminal serial como o *TeraTerm* no Windows o *CuteCom* no Linux ou então acessar diretamente as portas seriais utilizando-se de comandos de programação. Na linguagem C por exemplo, a biblioteca *termios.h* possui uma série de comandos para acessar as portas seriais em um ambiente Linux.

A taxa de comunicação aqui é denominada *baud rate*, e significa a quantidade de bits transmitidos em 1 segundo. O valor máximo possível de comunicação possui forte dependência com a interface de comunicação, variando de dezenas de milhares a até centenas de milhares de bits.

Para se estabelecer a comunicação entre a CPU e a FPGA é necessária a utilização de circuitos integrados responsáveis pelo ajuste da tensão. Isso é necessário, pois os níveis de tensão utilizados pelo RS-232 são de +12V/-12V, onde o bit 1 é -12V (*Mark*) e o bit 0 é +12V (*Space*). Este ajuste é provido pelo MAX3232, cuja interface é apresentada na figura 19. Devido ao fato que sua tecnologia é CMOS, assim como a tecnologia da FPGA, não é necessário nenhum circuito de interfaceamento para conectar ambos, podendo então ser realizada a conexão direta entre os pinos de ambos. O mesmo não ocorreria caso o circuito em questão fosse de tecnologia TTL.

Figura 19: Circuito MAX3232

Fonte: (TEXAS, 2017)

Um teste foi realizado aqui para se estimar a velocidade máxima de comunicação entre a CPU e a FPGA utilizando o protocolo RS-232. O circuito montado é apresentado na figura 20, onde foi utilizado o *TeraTerm* para se acessar a porta serial da CPU e uma UART implementada em VHDL, em uma placa FPGA DEO-CV.

Figura 20: Circuito para comunicação entre CPU e FPGA com RS-232

Fonte: Autor

Optou-se pela transmissão do caractere U, que apresenta código ASCII 01010101 em binário (55 em hexadecimal). Os resultados obtidos foram capturados em um osciloscópio Tektronix TBS 1052B e são apresentados nas figuras 21 e 22. Observou-se aqui que a maior velocidade possível para se enviar um pacote sem a ocorrência de erros foi 115.200 bits/s, onde a forma apresenta simetria nos períodos em que a onda fica em ativo alto e ativo baixo. Com a velocidade em 230.400 bits/s, a simetria anterior não ocorre, de maneira que a forma de onda capturada possui maior quantidade de tempo em ativo alto do que em ativo baixo durante a transmissão. No primeiro caso a FPGA sempre recebe o valor 55, enquanto no segundo caso ocorre uma alternância entre o valor 55 e o valor D5 (valores em hexadecimal).

Estes resultados obtidos estão condizentes com a especificação do MAX3232, que estabelece como pico a velocidade de 250 kbit/s ¹, porém a mesma não é adequada para transmissão contínua por um longo período de tempo.

Assim, conclui-se que a taxa máxima de comunicação sem que ocorram erros durante a transmissão do pacote é muito menor que as frequências dos *clocks*, tanto da CPU quanto da FPGA. Desta forma, o limitante neste tipo de comunicação é a própria taxa do protocolo, limitada pela interface.

Figura 21: Forma de onda correspondente ao caractere U com *baud rate* de 115.200

Fonte: Autor

¹<http://www.ti.com/lit/ds/symlink/max3232.pdf>

Figura 22: Forma de onda correspondente ao caractere U com *baud rate* de 230.400

Fonte: Autor

4.2.2 PCI *Express*

A comunicação utilizando o *Peripheral Component Interconnect Express* (PCI *Express*) possui velocidades superiores quando comparada com o protocolo RS-232 e o USB. Uma questão crucial na utilização do PCI *Express* diz respeito à sua limitação apenas para placas FPGA que possuem esta conexão. Estas placas costumam ser mais caras que as demais, e são fixadas diretamente na CPU no conector PCIe. A figura 23 mostra um exemplo de placa FPGA com conexão PCI *Express*.

Figura 23: Kit de desenvolvimento FPGA Stratix 10GX

Fonte: Terrasic (s.d.)

As taxas de comunicação aqui chegam à ordem de GB/s; porém, tal valor é superior

ao *clock* das FPGAs, de tal forma que o *clock* costuma ser o limitante neste tipo de comunicação.

Aqui utiliza-se comumente um recurso denominado processador lógico, disponível nas FPGAs. A Intel utiliza o processador Nios II; já a Xilinx utiliza o processador Microblaze. A função destes processadores neste contexto é promover uma interface entre o circuito digital descrito na FPGA (aqui atuando como acelerador) e a CPU. Essa interface é feita a partir de operações de leitura e escrita em posições de memória, sendo necessário estabelecer um mecanismo de sincronização.

A seguir é apresentado um pseudocódigo simples, demonstrando como é feita tal comunicação entre a CPU e o Nios II.

Algorithm 1 Envio de dados da FPGA para a CPU

```

1: procedure NIOS II
2:   Escreve dados provenientes do circuito da FPGA em posição de memória.
3:   Seta flag em posição de memória indicando escrita dos dados
4: procedure CPU
5:   while Flag de leitura não foi setada do
6:     Espera
7:   Lê dados provenientes da FPGA

```

4.2.3 Comunicação via Barramento

A comunicação via barramento ou *bridge* é utilizada nos casos onde a própria placa FPGA é híbrida, possuindo uma placa FPGA e um ARM por exemplo, também conhecido aqui como *Hard Processor System* (HPS). Nestes casos, é possível se estabelecer uma comunicação direta entre a CPU e a FPGA, utilizando-se do próprio barramento da placa. A figura 24 ilustra o diagrama interno de uma placa híbrida Cyclone V SoC.

Figura 24: Terasic SoC Kit

Fonte: Terasic (s.d.)

Neste sistema, os endereços de memória do barramento são disponíveis tanto para a FPGA quanto para o HPS. É necessário mapeá-los em ambas as partes, de forma que a leitura e a escrita destes endereços permitam a comunicação, podendo novamente utilizar o processador lógico da FPGA para tal. Por se utilizar de um barramento, assim como o PCI *Express*, esse tipo de comunicação chega a taxas de GB/s, onde novamente o *clock* torna-se o limitante.

Um limitante específico da utilização desta placa híbrida, ao invés de realizar a comunicação entre uma CPU e uma placa FPGA comum, está no baixo volume de memória do HPS. As placas híbridas mais simples costumam ter uma memória de poucos GBs dedicada ao HPS. Portanto, não é viável a utilização deste tipo de placa em projetos com grande quantidade de dados.

5 FERRAMENTA DE MONTAGEM DE GENOMA

O capítulo apresenta a arquitetura geral da ferramenta projetada, juntamente com sua adequação para diversos ambientes. Tal ferramenta visa atender a proposta de realização da montagem através da correção híbrida, trabalhando com dois arquivos de tecnologias distintas.

A ferramenta em questão é dividida em etapas, onde a etapa de correção das sequências PacBio possui maior custo computacional que as demais. Por isso, foram especificadas arquiteturas para diferentes ambientes, visando explorar todos os recursos computacionais disponíveis, e com isso melhorar o desempenho da etapa de correção, que é a mais custosa computacionalmente. Também se destaca a seguir o processo de validação da ferramenta, onde o genoma montado pode ser comparado a um genoma de referência, visando qualificar a montagem.

5.1 Arquitetura da Ferramenta

A arquitetura da ferramenta, apresentada pela figura 25, foi estabelecida de maneira modular, onde se destaca sua implementação em *software* e em *hardware* (FPGA).

Figura 25: Arquitetura da ferramenta

Fonte: Autor

A seguir serão descritas suas etapas, juntamente com a possibilidade de validação dos resultados obtidos pela mesma.

5.1.1 Coleta de Dados

A primeira etapa a ser executada é a coleta de dados. Ela é responsável por adquirir todos os arquivos necessários para a execução completa da ferramenta. São eles o arquivo das sequências pequenas (Illumina) e o arquivo das sequências grandes (PacBio).

O processo aqui é feito de maneira manual, onde o usuário da ferramenta deve encontrar um banco de dados genético confiável para obter os arquivos. É importante que os arquivos sejam não somente da mesma espécie, mas também do mesmo organismo, pois organismos de uma mesma espécie possuem variação em seu genoma, como no caso do ser humano. Também deve-se ter um cuidado no momento da coleta do arquivo Illumina, de tal forma que caso o sequenciamento seja do tipo *paired-ended*, suas sequências não estejam juntas em um mesmo arquivo.

5.1.2 Filtro das Qualidades das Sequências Illumina

A segunda etapa é o filtro das qualidades das sequências Illumina. Conforme foi mencionado no capítulo de aspectos conceituais, sequências Illumina possuem qualidade superior às sequências PacBio, porém isso não as impede de conter erros provenientes do processo de sequenciamento. É vantajoso para a ferramenta não necessitar de um tratamento das qualidades das sequências pequenas durante a etapa de correção, pois isso diminui pela metade a quantidade de dados do arquivo.

Para tal, é necessário garantir que a qualidade geral de todas as sequências do arquivo seja alta. Esta garantia pode ser obtida com o auxílio de outras ferramentas específicas para este propósito, como no caso da Trimmomatic, citada anteriormente. Porém, no contexto deste trabalho, optou-se pelo desenvolvimento de um algoritmo próprio, visando a uma maior autonomia no controle dos parâmetros e da flexibilização do mesmo.

O algoritmo em questão trabalha com dois parâmetros, sendo eles a média de *Q-score* das sequências e o *Q-score* individual de cada base nitrogenada. Estes parâmetros são selecionados pelo usuário, tendo como valor padrão 39 para a média de *Q-score* e 30 para o *Q-score* de cada base. Uma observação importante relacionada ao parâmetro da média é que o valor 39 atua aqui como valor ideal. O motivo para tal é que as sequências Illumina costumam ser codificadas com a codificação do tipo Sanger, onde o valor máximo de *Q-score* é 40. Desta forma, 39 é o maior número viável para se utilizar neste parâmetro, pois caso este número fosse aumentado, a maior parte das sequências do arquivo seria descartada. Também não é recomendado aqui reduzir este valor, pois haveria uma maior chance de alguma das sequências apresentarem erro em uma de suas bases. Assim, partindo-se do valor 39 fixado para a média, espera-se que o usuário altere o parâmetro correspondente ao valor individual da qualidade da base, o que irá resultar em arquivos de saída com tamanho distintos devido ao descarte. Cabe ao usuário então, sabendo a cobertura necessária para o arquivo Illumina, fornecer o valor mais adequado para este parâmetro.

5.1.3 Retirada das Sequências Repetidas

A terceira etapa consiste em se retirar sequências pequenas repetidas que tenham permanecido após o processo de filtragem. O processo de sequenciamento gera redundância do material genético, fazendo com que os arquivos de saída dos sequenciadores tenham tamanho maior que o genoma do organismo. Isso resulta em sequências semelhantes, que

auxiliam no processo de montagem. Porém, no caso das sequências pequenas, devido ao seu tamanho, é possível que o sequenciamento resulte em sequências iguais, que não contribuem para a montagem. Neste caso, sua existência atrapalha a correção, por apenas aumentar o volume de dados. Assim, esta etapa visa retirar estas sequências, reduzindo o tamanho do arquivo.

Juntamente com a retirada das sequências pequenas, esta etapa também retira toda a informação correspondente às qualidades das sequências. Para tal, parte-se do princípio de que a etapa anterior foi executada corretamente, não sendo mais necessárias as informações das qualidades.

Inicialmente o arquivo é recortado, de forma que apenas o conteúdo das linhas correspondentes às sequências seja mantido. Em seguida é realizada uma comparação entre as sequências, para se encontrar sequências iguais e realizar a remoção. A figura 26 ilustra o resultado da etapa.

Figura 26: Etapa de retirada de sequências repetidas

Fonte: Autor

5.1.4 Tratamento de Qualidades das Sequências PacBio

A quarta etapa visa eliminar a necessidade de se utilizar os dados correspondentes às qualidades das sequências PacBio durante a correção, através de um tratamento das qualidades. Para tal, conforme foi apresentado, cada base possui um *Q-score* diferente, onde tal *Q-score* possui relação inversamente proporcional à probabilidade da base ter sido sequenciada erroneamente.

Antes de realizar o tratamento é necessário identificar qual o tipo de codificação utilizada no arquivo das sequências grandes. Isso é feito através da busca de caracteres

específicos de cada codificação no arquivo, sendo possível assim isolar o tipo de codificação por meio de um processo de exclusão. Após obter o tipo de codificação é estabelecido um *threshold* para o *Q-score*, de forma que caso determinada base possua um *Q-score* inferior ao *threshold*, ela é substituída pela letra N. Esta letra tem como propósito indicar que a base em questão é indefinida, possuindo um tratamento diferenciado na etapa de correção.

Esse *threshold* é estabelecido a partir de uma análise estatística do arquivo em questão. (PACBIO, 2020) afirma que as sequências em questão possuem acurácia de aproximadamente 90%, desta forma, o algoritmo em questão foi projetado para substituir 10% das bases do arquivo pela letra N. Para tal, inicialmente é realizada uma análise do arquivo das sequências grandes, contando a presença dos caracteres relacionados aos *Q-scores* distintos, variando aqui de 0 a 10 aproximadamente devido à tecnologia PacBio. Com base na distribuição de cada caractere é possível identificar qual valor de *Q-score* contém 10% das bases com pior qualidade. Estas então são substituídas pela letra N.

Após o término desta etapa, o arquivo das sequências grandes tem seu tamanho reduzido pela metade, onde uma sequência é representada por duas linhas, sendo elas o cabeçalho da sequência e a sequência.

5.1.5 Correção

A quinta etapa é a correção propriamente dita, sendo esta o foco da ferramenta desenvolvida neste projeto. Sua contribuição no custo computacional da ferramenta é a maior, sendo sucedida pela etapa de montagem, enquanto as outras etapas possuem custo desprezível (inferior a 1% do tempo total de execução). Por isso sua implementação foi feita em *software* e em *hardware*, visando a redução deste custo. O algoritmo base escolhido para a correção é baseado na janela deslizante. Este algoritmo realiza um número elevado de operações de alinhamento e de comparação, sendo este número superior a outras abordagens, como por exemplo a utilização de grafos. Isto resulta em um maior custo computacional, porém em melhores resultados.

O algoritmo começa com a leitura do conteúdo do arquivo das sequências grandes e das sequências pequenas, seguido da abertura do arquivo de saída, onde serão escritas as sequências corrigidas. Após a preparação dos dados, inicia-se a correção, onde uma sequência grande (representada por duas linhas) é lida do arquivo. Em seguida é lida uma sequência pequena do outro arquivo, que é então deslizada pela sequência grande. O deslizamento consiste em se comparar cada posição da sequência pequena em relação à

sequência grande, para que se calcule a similaridade obtida. Aqui é atribuída a pontuação 1 caso ocorra uma situação de bases iguais ou então a sequência grande possua o caractere N. No caso de bases distintas é atribuída a pontuação 0.

Desta forma, para cada deslizamento realizado existe um valor de similaridade calculado, onde tal valor de similaridade deve ser maior que um *threshold* estabelecido. São armazenados em variáveis o valor máximo de similaridade obtido em todas as situações do deslizamento, juntamente com um *offset* de deslizamento. Caso este valor máximo de similaridade seja superior ao *threshold*, a sequência pequena com o auxílio do *offset* corrige a sequência grande, conforme ilustrado na figura 27.

Este procedimento é repetido para todas as sequências pequenas do arquivo. Logo após, a sequência grande corrigida é escrita no arquivo de saída, sendo lida em seguida uma nova sequência grande, até o término de todas as sequências grandes.

Figura 27: Deslizamento de uma sequência pequena em uma sequência grande

Fonte: Autor

A seguir é apresentado o pseudocódigo do algoritmo de correção, resumindo seu processo.

Algorithm 2 Algoritmo para etapa de Correção

```

1: Carrega arquivo de sequências pequenas
2: Carrega arquivo de sequências grandes
3: Abre arquivo de saída
4: while Restam sequências grandes para serem corrigidas do
5:   Lê uma sequência grande do arquivo
6:   while Restam sequências pequenas para corrigir do
7:      $offset \leftarrow 0$ .
8:      $similaridademax \leftarrow 0$ .
9:      $i \leftarrow 0$ .
10:    Lê uma sequência pequena do arquivo
11:    while Deslizamento de sequência pequena na sequência grande do
12:      Compara localmente sequência grande com sequência pequena para obter
      pontuação
13:       $similaridade \leftarrow pontuacao$ .
14:      if  $similaridade > similaridademax$  then
15:         $similaridademax \leftarrow similaridade$ .
16:         $offset \leftarrow i$ .
17:      if  $similaridademax > threshold$  then
18:        Corrige sequência grande com pequena
19:      Escreve sequência grande corrigida no arquivo de saída
20: Fecha o arquivo de saída

```

Como pode-se observar, tal algoritmo possui alta complexidade computacional, pois necessita percorrer múltiplas vezes o arquivo das sequências pequenas. Com o intuito de se minimizar o custo do algoritmo, critérios estatísticos e/ou heurísticos podem ser utilizados. Foram elaborados aqui dois critérios.

O primeiro é o tamanho do arquivo Illumina (sequências 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. Esta truncagem no contexto desta ferramenta foi realizada na etapa de filtro das qualidades.

O segundo critério consiste em uma maneira de se pular sequências pequenas sem a necessidade de 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. 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-se 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.

A utilização deste tipo de algoritmo (janela deslizante) foi optada devido à facilidade de sua implementação em *hardware*, onde outras abordagens iriam resultar em um *overhead* maior de projeto. Acredita-se que a implementação com este algoritmo trivial possa resultar em implantações futuras com algoritmos mais sofisticados.

5.1.6 Montagem

A penúltima etapa é a montagem das sequências genômicas. Como o escopo deste trabalho é atuar diretamente na etapa de correção das sequências PacBio, esta etapa é realizada com o auxílio de alguma outra ferramenta já desenvolvida para este propósito. Pode-se utilizar neste caso ferramentas desenvolvidas apenas para a montagem das sequências PacBio, como é o caso da Smartdenovo, ou ferramentas que realizem a correção e a montagem, desde que seja possível executar suas etapas separadamente, como é o caso da Canu.

5.1.7 Validação dos Resultados

A validação dos resultados obtidos com a montagem do genoma é feita através do alinhamento do genoma montado com o genoma de referência do organismo, conforme ilustrado na figura 28. Este último pode ser obtido novamente com uma coleta de dados em um banco de dados confiável.

Aqui pode-se utilizar ferramentas já desenvolvidas com este propósito, capazes de realizar alinhamentos entre genomas visando à obtenção da similaridade. Ressalta-se que a similaridade obtida nunca será de 100%, devido à diferença entre genomas de organismos de uma mesma espécie, de maneira que o genoma de referência nunca será o exato genoma montado. A similaridade desejada entre o organismo montado e o genoma de referência vai depender do ser vivo em questão. No caso de bactérias como por exemplo a *E. coli*, a literatura apresenta valores de similaridade superiores as 99%.

Figura 28: Validação dos resultados obtidos pela ferramenta

Fonte: Autor

5.2 Arquitetura da Ferramenta *Multicore*

A adequação da ferramenta para um ambiente *multicore* foi realizada modificando-se apenas a etapa de correção, apresentada na figura 25. Desta forma, todas as outras etapas foram executadas de maneira sequencial, devido ao seu baixo custo computacional.

O processo de correção consiste na criação de *threads* e divisão do arquivo das sequências grandes para cada *thread*, enquanto o arquivo das sequências pequenas é compartilhado. Cada *thread* lê uma sequência grande com um ponteiro público e realiza a correção com as sequências pequenas utilizando um ponteiro privado. Aqui é utilizado um balanceamento de carga (*self scheduling*), onde cada *thread* é independente das demais e lê uma nova sequência grande à medida que uma anterior é corrigida, escrevendo-a no arquivo de saída.

Com o intuito de se evitar situações de corrida, onde duas ou mais *threads* tentam ler ou escrever o arquivo das sequências grandes ao mesmo tempo, é necessário utilizar um mecanismo de sincronização entre as *threads*. O sincronismo aqui é realizado com o auxílio de seções críticas, utilizando a diretiva de compilação *pragma omp critical*.

A seguir é apresentado o pseudocódigo da etapa de correção da ferramenta em um ambiente *multicore*, desta vez adicionando o critério de pular sequências pequenas que foi citado anteriormente na seção de implementação sequencial do algoritmo da correção. Este critério, descrito nas linhas 22 a 31 do pseudocódigo em questão, tem como função reduzir o custo computacional do algoritmo, descartando sequências pequenas que não teriam o potencial de corrigir uma sequência grande em questão.

Algorithm 3 Algoritmo *Multicore*

```

1: Carrega arquivo de sequências pequenas
2: Carrega arquivo de sequências grandes
3: Abre arquivo de saída
4: Cria threads
5: procedure THREAD
6:   while Restam sequências grandes para serem corrigidas do
7:     Lê uma sequência grande do arquivo
8:     while Restam sequências pequenas para corrigir do
9:        $offset \leftarrow 0$ .
10:       $similaridademax \leftarrow 0$ .
11:       $i \leftarrow 0$ .
12:      Lê uma sequência pequena do arquivo
13:      while Deslizamento de sequência pequena na sequência grande do
14:        Compara localmente sequência grande com sequência pequena para obter pontuação
15:         $similaridade \leftarrow pontuação$ .
16:        if  $similaridade > similaridademax$  then
17:           $similaridademax \leftarrow similaridade$ .
18:           $offset \leftarrow i$ .
19:        if  $similaridademax > threshold$  then
20:          Corrige sequência grande com pequena
21:        else
22:          while Sequência pequena não corrige sequência grande do
23:            Lê nova sequência pequena
24:            if Comprimento da sequência pequena atual é diferente da nova then
25:              break
26:             $caracteredif \leftarrow 0$ .
27:            while Comprimento da sequência do
28:              if Caractere da sequência pequena atual é diferente da nova then
29:                 $caracteredif \leftarrow caracteredif + 1$ .
30:              if  $caracteredif + similaridademax \geq threshold$  then
31:                break;
32:          Escreve sequência grande corrigida no arquivo de saída
33: Espera o término da execução das threads
34: Fecha o arquivo de saída

```

5.3 Arquitetura da Ferramenta para *Cluster*

Anterior à definição da melhor estratégia de adaptação da ferramenta para um ambiente de *cluster*, foram implementadas e analisadas 4 estratégias distintas, cujos resultados estão disponíveis em (ALMEIDA et al., 2019). As versões do algoritmo foram:

- Algoritmo Distribuído: Esta versão utiliza-se de apenas um processo criado em cada nó do *cluster*.
- Algoritmo Distribuído Multitarefa: Esta versão utiliza-se de um número de processos igual ao número de recursos disponíveis no *cluster*. Desta forma, é realizado apenas paralelismo em nível de processos.
- Algoritmo Distribuído *Multicore*: Esta versão utiliza-se de um processo criado em cada nó do *cluster* e cada processo cria *threads*, alocando assim todos os recursos disponíveis. Desta forma, essa versão utiliza paralelismo em nível de processos e *threads*.
- Algoritmo Distribuído *Multicore* com *Broadcast*: Esta versão é um incremento da versão anterior, utilizando o *broadcast* na etapa inicial de comunicação entre os nós.

A estratégia apresentada aqui é a versão distribuída *multicore* com *broadcast* do algoritmo de correção, que foi a que obteve melhor resultado.

Nesta versão, inicialmente o arquivo contendo as sequências pequenas é distribuído para a rede, utilizando o *broadcast*. Desta forma, todos os nós terão uma cópia do arquivo. Em seguida um bloco de sequências grandes é enviado para cada nó, onde o número de sequências é igual ao número de *cores* do nó. Dentro de cada nó são criadas *threads*, sendo cada *thread* responsável pela correção de uma sequência grande, utilizando o arquivo das sequências pequenas. À medida que as sequências grandes são corrigidas, o nó da rede envia ao nó mestre a sequência corrigida, e solicita uma nova sequência grande para sua *thread*. Esse processo ocorre até que todas as sequências grandes sejam corrigidas utilizando uma estratégia do tipo *bag of tasks*.

A seguir é apresentado o pseudocódigo da etapa de correção da ferramenta em um ambiente de *Cluster*. O pseudocódigo em questão está simplificado, visando apenas ao resumo da comunicação entre os nós da rede. Nele também foi realizada a divisão entre a função do nó mestre e dos nós escravos.

Algorithm 4 Algoritmo *Cluster*

```

1: procedure NÓ MESTRE
2:   Carrega arquivo de sequências pequenas
3:   Distribui arquivo para a rede via broadcast
4:   Carrega arquivo de sequências grandes
5:   Abre arquivo de saída
6:   for cada nó escravo da rede do
7:     Lê bloco de sequências grandes do arquivo
8:     Envia bloco de sequências grandes para o nó escravo
9:   while Restam sequências grandes para serem corrigidas do
10:    Recebe sequência grande corrigida
11:    Escreve sequência grande corrigida no arquivo de saída
12:    Lê sequência grande não corrigida
13:    Envia sequência grande não corrigida
14:   for cada nó escravo da rede do
15:     Recebe últimas sequências grandes corrigidas
16:     Escreve últimas sequências grandes corrigidas no arquivo de saída
17:     Envia sinal de término para o nó escravo
18:   Fecha arquivo de saída

19: procedure NÓS ESCRAVOS
20:   Recebe arquivo de sequências pequenas
21:   Recebe bloco de sequências grandes
22:   Cria threads
23:   procedure THREAD
24:     while Não recebe sinal de término do
25:       Realiza correção da sequência grande
26:       Envia sequência grande corrigida para nó mestre
27:       Recebe nova sequência grande ou sinal de término
28:   Espera o término da execução de todas as threads

```

5.4 Arquitetura da Ferramenta *Multicore* com FPGA

A arquitetura da ferramenta em um sistema de memória compartilhada com FPGA é dada de tal forma que todas as etapas são executadas em uma mesma máquina e em apenas uma placa FPGA. No caso da arquitetura geral da ferramenta, todas as etapas, com exceção da correção, são executadas na CPU, enquanto esta é executada conjuntamente na CPU e na FPGA. Desta forma, os dados relativos às sequências grandes e às pequenas

são compartilhados em sua totalidade entre a CPU e a FPGA na etapa de correção. A seguir são descritas separadamente as arquiteturas do *software* e do *hardware* (FPGA).

5.4.1 Arquitetura do *Software*

O algoritmo implementado na CPU é responsável pela manipulação dos dados provenientes de ambos os arquivos, e escrita no arquivo da saída. Inicialmente os arquivos contendo as sequências são carregados na memória, e o arquivo de saída é aberto. Em seguida, um bloco de sequências grandes é enviado para a FPGA, utilizando-se do meio de comunicação. Após a FPGA receber o bloco, sequências pequenas são enviadas uma a uma, até o término do arquivo. A CPU então envia um sinal de término das sequências pequenas para a FPGA, recebe o bloco de sequências grandes corrigidas, escreve-as no arquivo de saída e envia um novo bloco de sequências grandes para a FPGA. Este processo é repetido até o término das sequências grandes.

O pseudocódigo a seguir apresenta o algoritmo em questão.

Algorithm 5 Algoritmo *Software*

```

1: Carrega arquivo de sequências pequenas
2: Carrega arquivo de sequências grandes
3: Abre arquivo de saída
4: while Restam sequências grandes para serem corrigidas do
5:   Lê um bloco de sequências grandes do arquivo
6:   Envia bloco de sequências grandes para a FPGA
7:   while Restam sequências pequenas para corrigir do
8:     Lê uma sequência pequena do arquivo
9:     Envia sequência pequena para a FPGA
10:  Envia sinal de término das sequências pequenas para a FPGA
11:  Recebe bloco de sequências grandes corrigidas
12:  Escreve bloco de sequências corrigidas no arquivo de saída
13: Fecha arquivo de saída

```

Observa-se a partir do pseudocódigo fornecido que neste caso é apresentada uma versão sequencial do algoritmo, visto que existe apenas um processo na CPU responsável por realizar a comunicação com a FPGA. O aspecto *multicore* desta implementação está na possibilidade de utilizar eventuais núcleos ociosos para que estes também realizem a correção das sequências grandes, a qual é feita integralmente em *software*. Desta forma, uma parcela das sequências teria sua correção feita em *hardware*, pela FPGA, e a outra parcela teria sua correção feita em *software* pelos núcleos ociosos, analogamente ao que

foi descrito na implementação da versão *multicore*. O pseudocódigo completo desta versão será apresentado no capítulo 6, pois o mesmo apresenta uma restrição devido a quantidade limitada de recursos utilizados no contexto da implementação deste trabalho.

5.4.2 Arquitetura do *Hardware*

Dentro da FPGA é implementado um circuito de propósito específico para a correção. O circuito, apresentado na figura 29, é composto por três blocos, sendo eles o receptor, o transmissor e o corretor.

Figura 29: Diagrama de blocos em alto nível do circuito na FPGA

O algoritmo implementado na FPGA é apresentado sumariamente pelo pseudocódigo a seguir. Nas subseções seguintes são melhor especificados os três blocos.

Algorithm 6 Algoritmo *Hardware*

- 1: **while** Existem sequências grandes para corrigir **do**
 - 2: Recebe bloco de sequências grandes da CPU
 - 3: **while** Restam sequências pequenas para corrigir as grandes **do**
 - 4: Recebe sequência pequena
 - 5: Realiza deslocamento da sequência pequena pela sequência grande
 - 6: **if** *similaridademáximaobtida* > *threshold* **then**
 - 7: Corrige sequência grande com base na sequência pequena
 - 8: Envia bloco de sequências grandes corrigidas para a CPU
-

5.4.2.1 Circuito Receptor

O circuito receptor é dividido em duas partes, que são o receptor propriamente dito e um *buffer* de recepção. O receptor recebe os pacotes provenientes da CPU, e os encaminha

para o *buffer*. A função do *buffer* aqui é montar a sequência recebida pela FPGA, sendo pequena ou grande, pois o tamanho da sequência é muito superior ao de um pacote recebido. Para tal, ele realiza deslocamentos sucessivos nos pacotes recebidos, visando ordenar os dados encaminhados pelo receptor, para montar a sequência.

5.4.2.2 Circuito Corretor

O circuito corretor é responsável pela correção das sequências grandes. Este promove o deslizamento das sequências pequenas, armazenadas em um registrador de deslocamento, nas sequências grandes. A figura 30 ilustra o diagrama de blocos do circuito corretor.

Figura 30: Circuito corretor

Fonte: Autor

A comparação para obtenção da similaridade entre as sequências é realizada com o auxílio de portas XNOR. Dada uma posição que a sequência pequena tenha sido deslocada na sequência grande, suas bases são então comparadas. Nos casos onde as bases são iguais, o circuito composto pelas portas XNOR retorna o valor 1, já nos casos onde as bases são diferentes, o circuito retorna o valor 0. Cada dupla de bases alinhada então vai retornar um valor que pode ser 1 ou 0, resultando em um vetor de números binários com tamanho igual ao número de bases comparadas. Os valores obtidos nas comparações então são adicionados utilizando-se diversos somadores, com o intuito de se realizar o cálculo da similaridade em apenas um ciclo de *clock*.

É importante ressaltar que os somadores foram projetados seguindo-se um modelo de

soma em árvore binária (somador paralelo), ao invés da soma em cascata (*ripple carry adders*). Isso é necessário, pois o atraso total da adição na soma em cascata é da ordem de $n-1$ atrasos do somador de um bit, onde n é o número de bits somados. Já o atraso da soma em árvore binária é de n , onde $2^n \geq$ que é o número de bits somados. A importância aqui está em se garantir que o tempo total de atraso seja inferior ao período do *clock* da placa; caso contrário, a operação do cálculo da similaridade não será feita corretamente. A figura 31 ilustra a diferença no processo de soma de 4 bits em cascata e em árvore, onde o segundo caso possui um atraso total inferior ao primeiro. Ressalta-se que esse é um exemplo simplificado do circuito somador que foi implementado. Pode-se entender os 4 bits somados na figura como bits correspondentes a comparação de 4 bases da sequência pequena com 4 bases da sequência grande.

Figura 31: Soma de 4 bits em cascata e em árvore

Fonte: Autor

Assim, cada deslizamento da sequência pequena na sequência grande terá sua similaridade calculada no mesmo ciclo de *clock*. Caso esse valor seja superior ao *threshold* estabelecido, ele é armazenado em um registrador juntamente com o *offset* de deslocamento, controlado a partir de um contador. Após o término do deslizamento, observa-se o conteúdo do registrador de similaridade, para verificar se alguma posição obteve similaridade superior ao *threshold*. Em caso afirmativo, é realizada uma série de deslizamentos sucessivos na sequência pequena, utilizando o valor do *offset* armazenado para voltá-la até a posição de maior similaridade com a sequência grande. Por fim, é realizada a correção naquela posição. Este processo então é repetido para todas as sequências pequenas. Então, a sequência grande corrigida é encaminhada para o circuito transmissor.

5.4.2.3 Circuito Transmissor

Assim como o circuito receptor, o circuito transmissor também é dividido em duas partes, sendo o transmissor propriamente dito e um circuito *buffer* de transmissão. O circuito *buffer* recebe a sequência grande corrigida do circuito corretor e, em seguida, quebra-a em pacotes, encaminhando-os para o transmissor. Este é responsável por enviar os pacotes para a CPU, utilizando o meio de comunicação.

5.5 Arquitetura da Ferramenta para *Cluster* com FPGAs

A arquitetura da ferramenta em um sistema de memória distribuída com FPGAs é projetada visando otimizar a etapa de correção, que é a que possui maior custo computacional. Esse sistema é uma união dos dois últimos apresentados, de maneira que, para realizar sua implementação, é necessário apenas realizar uma união dos anteriores. Uma visão em alto nível do ambiente em questão é ilustrada pela figura 32.

Figura 32: Ambiente de memória compartilhada e distribuída com FPGAs

Fonte: Autor

Cada nó é representado por uma CPU e uma FPGA. Um dos nós da rede é o nó mestre, enquanto os outros nós são escravos. O nó mestre é responsável inicialmente por executar todas as etapas da ferramenta, até o momento da correção, de maneira sequencial. Na correção, o arquivo relativo às sequências pequenas é distribuído para

todos os nós da rede, e o arquivo das sequências grandes será distribuído para a rede conforme a demanda, realizando assim um balanceamento de carga. Desta forma, cada nó será responsável por corrigir uma parcela das sequências grandes. Cabe ressaltar que cada nó possui múltiplos núcleos, sendo possível utilizar-se da programação paralela em memória compartilhada em cada um deles. No quesito da FPGA, o processamento é o mesmo utilizado na arquitetura *multicore* com FPGA.

Novamente é apresentado o pseudocódigo simplificado da ferramenta nas CPUs dos nós, com o foco para a comunicação entre os nós e entre a CPU e a FPGA.

Algorithm 7 Algoritmo *Cluster* com FGAs

```

1: procedure NÓ MESTRE
2:   Carrega arquivo de sequências pequenas
3:   Distribui arquivo para a rede via broadcast
4:   Carrega arquivo de sequências grandes
5:   Abre arquivo de saída
6:   for cada nó da rede do
7:     Lê bloco de sequências grandes do arquivo
8:     Envia bloco de sequências grandes para o nó
9:   while Restam sequências grandes para serem corrigidas do
10:    Recebe sequência grande corrigida
11:    Escreve sequência grande corrigida no arquivo de saída
12:    Lê sequência grande não corrigida
13:    Envia sequência grande não corrigida
14:  for cada nó da rede do
15:    Recebe últimas sequências grandes corrigidas
16:    Escreve últimas sequências grandes corrigidas no arquivo de saída
17:    Envia sinal de término para o nó
18:  Fecha arquivo de saída

19: procedure NÓS ESCRAVOS
20:   Recebe arquivo de sequências pequenas
21:   Recebe bloco de sequências grandes
22:   while Não recebe sinal de término do
23:     Envia bloco de sequências grandes para a FPGA
24:     while Restam sequências pequenas para corrigir do
25:       Envia sequência pequena para FPGA
26:     Recebe bloco de sequências grandes corrigidas da FPGA
27:     Envia bloco de sequências grandes corrigidas para nó mestre
28:     Recebe novo bloco de sequências grandes ou sinal de término

```

Após a correção, a montagem também é realizada de maneira distribuída, onde os nós da rede trabalharão em conjunto para realizar os alinhamentos necessários para a montagem. Por último, o genoma montado será validado localmente, sendo necessário apenas o nó mestre para tal.

6 IMPLEMENTAÇÃO E ANÁLISE DOS RESULTADOS

O capítulo apresenta a implementação da ferramenta, cuja arquitetura foi apresentada no capítulo anterior, juntamente com os resultados obtidos. Inicialmente é apresentada a implementação geral da ferramenta, onde todas suas etapas foram implementadas de maneira sequencial, seguido das implementações paralelas e com FPGA da etapa de correção.

Para a apresentação dos resultados foram levantadas também curvas obtidas a partir de testes de escalabilidade para cada versão da ferramenta, considerando apenas a etapa de correção devido ao seu maior custo computacional. A última seção deste capítulo destina-se a uma comparação entre as diversas versões aqui desenvolvidas no aspecto de desempenho.

6.1 Implementação Geral da Ferramenta

Esta seção apresenta a implementação sequencial de todas as etapas da ferramenta, sendo ela a primeira versão desenvolvida.

6.1.1 Coleta de Dados

Nesta etapa optou-se por utilizar o banco de dados do NCBI ¹. Esta escolha foi feita em função de sua grande credibilidade, haja vista que periódicos internacionais exigem que para que uma publicação seja aceita, os arquivos contendo as informações genéticas sejam submetidas neste banco. Nele, as sequências genômicas são denominadas *raw sequences*, e podem ser localizadas no *Sequence Raw Archive* (SRA).

A coleta do genoma de referência é feita de maneira simples, acessando a aba *Genome* do banco de dados e realizando uma coleta direta da página. Para a coleta das sequências grandes e pequenas, o NCBI fornece um conjunto de ferramentas para realizar o acesso

¹<https://www.ncbi.nlm.nih.gov/>

ao 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 com o das sequências PacBio. No caso das sequências Illumina, a ferramenta permite separar as sequências provenientes de um sequenciamento do tipo *paired-ended* em dois arquivos.

6.1.2 Filtro de Qualidades das Sequências Illumina

Conforme explicado no capítulo anterior, o algoritmo desenvolvido funciona com dois parâmetros, sendo eles a média de *Q-score* da sequência e o *Q-score* individual de cada base. Estes parâmetros são apresentados a partir de comandos *define*, onde o usuário pode altera-los livremente.

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 necessário escrever no arquivo de saída utilizando a função *msync*. Além disso, após o término do algoritmo é necessário liberar a memória alocada com o *mmap*. Para isso utiliza-se a função *munmap*, caso contrário ocorrerá *memory leak*.

Desta forma, para cada sequência lida do arquivo de entrada foram feitas comparações baseadas nos valores dos parâmetros de *Q-score* individual de cada base e média de *Q-score* da sequência. Com base no resultado da comparação a sequência em questão era escrita no arquivo de saída ou não.

6.1.3 Retirada das Sequências Repetidas

Esta etapa foi implementada a partir de comandos do UNIX, utilizados no próprio *shell* do ambiente.

O primeiro passo foi extrair as linhas que não correspondem às bases nitrogenadas propriamente ditas. Conforme discutido, o arquivo fastq 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 três outras linhas de cada

bloco, gerando um arquivo intermediário contendo apenas as linhas com as bases.

A seguir, com esse arquivo intermediário foi necessário estabelecer uma abordagem para retirar as sequências repetidas deste arquivo. Foi utilizada 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 $O(n * \log(n))$. A busca por sequências repetidas foi implementada em C, onde a complexidade é de apenas $O(n)$, sendo necessário varrer o arquivo apenas uma vez para detectar as sequências repetidas. Desta forma a complexidade total do processo foi de $O(n + n * \log(n))$.

6.1.4 Tratamento de Qualidades das Sequências PacBio

A implementação aqui foi dividida em duas partes, onde a primeira parte consiste em se identificar o tipo de codificação utilizada no arquivo fastq e a segunda parte compreende em 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. Foi novamente utilizada a linguagem C. 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.

Para identificar o tipo de codificação utilizada, são observadas apenas as linhas correspondentes às qualidades das bases. As linhas são percorridas sequencialmente onde cada caractere é analisado individualmente. A análise consiste em verificar se o código ASCII do caractere se encontra nas faixas de ASCII de cada codificação. Caso seja identificado um caractere não pertencente a determinada codificação, aquela codificação é descartada. O processo ocorre até que reste apenas uma codificação.

Após a identificação do tipo de codificação utilizada, é realizada uma análise estatística no arquivo, visando descobrir um valor adequado para o *threshold* de substituição das bases pela letra N. Para tal, são utilizados os comandos *fold*, *sort* e *uniq* do UNIX para contar o número de ocorrências de cada caractere correspondente a codificação das bases no arquivo. Com base nesses valores é possível estabelecer então o valor do *threshold* com base no conjunto de 10% das bases contendo as piores qualidades.

Com o *threshold* estabelecido, o algoritmo implementado consiste em percorrer se-

quencialmente blocos de 4 linhas do arquivo das sequências grandes. As linhas das bases nitrogenadas e das qualidades das mesmas 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, o bloco é escrito no arquivo de saída e retorna-se o processo em outro bloco, até o término dos dados do arquivo de entrada.

6.1.5 Correção

Nesta subseção é apresentada a implementação base do algoritmo de correção, sendo utilizada a linguagem C sequencial apenas. Esta implementação segue o algoritmo apresentado em pseudocódigo na seção correspondente do capítulo anterior, onde para cada linha do arquivo das sequências grandes é percorrido o arquivo das sequências pequenas, realizando o deslizamento de cada sequência pequena na sequência grande.

6.1.6 Montagem

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.

6.1.7 Validação dos Resultados

A validação dos resultados foi feita utilizando o BLAST², que é uma ferramenta mantida pelo NCBI. Para utiliza-la é necessário apenas acessar a página, selecionar as opções correspondentes ao alinhamento de genomas e fornecer os dois arquivos, que são o genoma montado e o genoma de referência. O pedido então é encaminhado para uma fila de espera do *cluster* do NCBI, onde após a execução é retornado um *log* contendo as informações do alinhamento.

Como opção, o BLAST também possui uma versão executável para ser executada localmente em uma máquina ou *cluster*, porém isso não foi feito devido ao alto custo computacional da ferramenta, sendo mais viável utilizar o próprio *cluster* já fornecido.

²<https://blast.ncbi.nlm.nih.gov/Blast.cgi>

6.1.8 Análise dos Resultados

A análise da versão sequencial da ferramenta foi realizada em uma CPU com processador Intel i7-3770 com *clock* de 3,40 GHz e 4 *cores*. A memória é de 16GB de RAM e o sistema operacional é o openSUSE versão 13.2.

Para a realização dos testes na versão sequencial da ferramenta foram utilizados arquivos contendo sequências genômicas da bactéria *Escherichia coli*. Os arquivos contendo as sequências Illumina foram extraídos do banco de dados do NCBI, e possuem identificador ERR022075 ³, sendo elas provenientes de um sequenciamento do tipo *paired-ended*. O arquivo contendo as sequências PacBio foi extraído do repositório oficial da empresa ⁴.

Aqui é importante realizar uma observação sobre as sequências Illumina. Conforme citado na subseção de coleta dos dados, a ferramenta *fastq-dump* permite fazer a coleta das sequências provenientes do sequenciamento *paired-ended* em dois arquivos distintos, onde o primeiro contém a primeira metade de cada sequência e o segundo contém a segunda metade. (TAN et al., 2019) aponta a presença de uma maior quantidade de erros de sequenciamento na segunda metade das sequências. Isto posto, o segundo arquivo então foi descartado, sendo utilizado apenas o primeiro arquivo de sequências Illumina.

A escolha deste organismo em específico foi dada pela sua baixa complexidade genética, além de amplo estudo pela comunidade científica. Ambos os repositórios selecionados em questão também possuem boa credibilidade pela comunidade científica.

Devido ao baixo desempenho da versão sequencial, não é viável realizar a correção e a posterior montagem completa do genoma do organismo. Assim, foram escolhidos fragmentos de cada arquivo, visando apenas nesta situação calcular o tempo de execução e realizar uma análise de escalabilidade. Para tal, foi utilizado como base um arquivo de tamanho 1 MB contendo as sequências grandes e outro de tamanho 10 MB contendo as sequências pequenas.

O tempo médio obtido a partir da média de 10 tempos foi de 27.523 segundos e o desvio padrão foi de 24,7. Em seguida foram realizados testes de escalabilidade variando o tamanho de ambos os arquivos. No caso do arquivo das sequências grandes, foi variado o tamanho de 1 MB para 5 MB, utilizando-se passos de 1 MB e fixando o tamanho do outro arquivo. No caso do arquivo das sequências pequenas, foi variado o tamanho de 10 MB para 50 MB, utilizando-se passos de 10 MB e novamente fixando o tamanho do outro arquivo. Os resultados dos testes são apresentados nas figuras 33 e 34, onde destaca-se

³<https://www.ncbi.nlm.nih.gov/sra/?term=ERR022075>

⁴<https://github.com/PacificBiosciences/DevNet/wiki/E-coli-K12-MG1655-Hybrid-Assembly>

também a linha correspondente a escalabilidade linear.

Figura 33: Teste de escalabilidade do tamanho do arquivo PacBio para versão sequencial da ferramenta

Fonte: Autor

Figura 34: Teste de escalabilidade do tamanho do arquivo Illumina para versão sequencial da ferramenta

Fonte: Autor

Observa-se que em ambos os casos a escalabilidade é próxima à linear.

6.2 Implementação da Ferramenta *Multicore*

Esta seção apresenta a implementação *multicore* apenas da etapa de correção da ferramenta, juntamente com os resultados obtidos seguindo o mesmo método da versão sequencial.

6.2.1 Implementação da Ferramenta

A implementação da etapa de correção em um ambiente de memória compartilhada utiliza-se dos diversos núcleos presentes em uma mesma máquina para distribuir a carga de trabalho. Aqui foi utilizada a linguagem C com o OpenMP.

Após a abertura de ambos os arquivos e armazenamento do conteúdo na memória com o *mmap*, é criada uma zona paralela com *threads*. Cada *thread* possui um ponteiro público para o conteúdo do arquivo das sequências grandes, e um ponteiro privado para o conteúdo do arquivo das sequências pequenas. A leitura de uma sequência grande não corrigida e a escrita de uma sequência corrigida é realizada em seções críticas, implementadas pelo OpenMP com o auxílio de semáforos.

6.2.2 Análise dos Resultados

O ambiente computacional para a análise da versão *multicore* da ferramenta foi o mesmo utilizado na versão sequencial, sendo aproveitado dessa vez todos os seus núcleos disponíveis (4 *cores*). Os mesmos arquivos de teste da versão sequencial foram utilizados.

O tempo médio obtido a partir da média de 10 tempos foi de 7.092 segundos e o desvio padrão foi de 3,9, tendo um *speedup* de 3,88 em relação à versão sequencial. Os resultados dos testes de escalabilidade dos tamanhos dos arquivos são apresentados nas figuras 35 e 36, onde observa-se que novamente a escalabilidade é próxima à linear em ambos os casos. Isso é consequência da utilização do balanceamento de carga na distribuição das sequências para as *threads*.

Figura 35: Teste de escalabilidade do tamanho do arquivo PacBio para versão *multicore* da ferramenta

Fonte: Autor

Figura 36: Teste de escalabilidade do tamanho do arquivo Illumina para versão *multicore* da ferramenta

Fonte: Autor

6.3 Implementação da Ferramenta para *Cluster*

Esta seção apresenta a implementação para *cluster* apenas da etapa de correção da ferramenta, juntamente com os resultados obtidos seguindo o mesmo método da versão sequencial.

6.3.1 Implementação da Ferramenta

A implementação da etapa de correção em um ambiente de memória compartilhada e distribuída foi feita utilizando a linguagem C juntamente com o OpenMP e o MPI. O ambiente foi um *cluster* contendo 5 nós conectados por uma rede Gigabit Ethernet utilizando um *switch* modelo HPN V1405-8G. Cada nó possui um processador Intel i7-3770 com *clock* de 3,40 GHz e 4 *cores*. A memória é de 16GB de RAM e o sistema operacional é o openSUSE versão 13.2.

Foram criados 6 processos MPI, onde o nó mestre possui 2 processos, e os outros nós possuem apenas 1 processo. O processo excedente dentro do nó mestre é denominado processo mestre, enquanto os demais processos são denominados processos escravos. Cada processo escravo utiliza o OpenMP para criar 4 *threads*, tendo assim um total de 21 tarefas computacionais. Percebe-se então que o número de tarefas computacionais é superior ao número de recursos disponíveis no *cluster* (21 contra 20), porém, isso é justificado pelo fato que o processo mestre é responsável somente pela comunicação entre processos, ficando bloqueado na maior parte da execução da etapa de correção.

Assim, a comunicação entre os nós do *cluster* é feita através de troca de mensagens bloqueantes utilizando os pacotes do MPI. As *threads* existentes em cada processo escravo enviam sequências corrigidas e solicitam novas sequências em seções críticas, implementadas com o MPI utilizando semáforos.

6.3.2 Análise dos Resultados

A análise da versão para *cluster* foi realizada em um *cluster* contendo 5 nós conectados por uma rede Gigabit Ethernet utilizando um *switch* modelo HPN V1405-8G. Cada nó possui um processador Intel i7-3770 com *clock* de 3,40 GHz e 4 *cores*. A memória é de 16GB de RAM e o sistema operacional é o openSUSE versão 13.2.

O tempo médio obtido a partir da média de 10 tempos foi de 1.585,8 segundos e o desvio padrão foi de 3,8, tendo um *speedup* de 17,3 em relação à versão sequencial. Os

testes de escalabilidade seguiram a mesma lógica das versões anteriores, e os resultados em relação ao tamanho dos arquivos são apresentados nas figuras 37 e 38.

Figura 37: Teste de escalabilidade do tamanho do arquivo PacBio para versão para *cluster* da ferramenta

Fonte: Autor

Figura 38: Teste de escalabilidade do tamanho do arquivo Illumina para versão para *cluster* da ferramenta

Fonte: Autor

Aqui ocorre uma situação particular, onde a escalabilidade é superior à linear em ambos os casos. A justificativa para tal está na fila de espera existente entre todas as *threads* pertencentes aos nós escravos e o processo responsável pela comunicação no nó mestre. À medida que a quantidade de dados processados aumenta, existe uma dessincronização na fila de espera, de tal forma que as *threads* necessitam esperar menos tempo para receber uma nova sequência grande. O motivo desta dessincronização é que o tempo de correção para cada sequência grande é variável, dependendo principalmente do comprimento da sequência em questão. Como as sequências grandes possuem uma variação significativa em seu tamanho (de 1000 bases a até 10000 bases em média), este tempo de correção individual de cada sequência também possui uma variação relacionada. Desta forma, o tempo total de execução diminui.

6.4 Implementação da Ferramenta *Multicore* com FPGA

Esta seção apresenta a implementação em um ambiente *multicore* com FPGA apenas da etapa de correção da ferramenta, juntamente com um levantamento preliminar dos resultados experimentais realizado e publicado em congresso nacional.

6.4.1 Implementação da Ferramenta

O ambiente para a implementação da ferramenta foi um computador e uma placa FPGA. O computador em questão possui processador intel i7-720QM com *clock* de 1,9 GHz e 4 *cores*. A memória é de 6 GB de RAM e o sistema operacional é o Ubuntu LTS 16.04. A placa FPGA é a DE0-CV. Ela possui uma FPGA Cyclone V e opera com *clock* interno de 50 MHz, 49 K de elementos lógicos programáveis e 3080 Kb de memória.

Para a comunicação entre a CPU e a FPGA foi adotada uma interface serial com o protocolo RS-232. Sua escolha foi realizada em função da sua maior simplicidade de projeto, além do fato de que ele pode ser implementado em qualquer placa FPGA que tenha pinos GPIO (*General Purpose Input/Output*), tendo assim um escopo mais amplo do que outras formas de comunicação como PCI *Express*, Ethernet ou USB.

Visando minimizar os efeitos da baixa velocidade de comunicação do RS-232, foi adotado um modelo diferente de codificação para cada base nitrogenada, onde a codificação utilizada foi de 3 bits, ao invés de 8 bits no caso do ASCII estendido. Desta forma, é possível enviar um total de 8 bases a cada três pacotes (24 bits), tendo um ganho de de-

sempenho na comunicação de aproximadamente 160%. A tabela 6 apresenta a codificação utilizada para cada base.

Tabela 6: Codificação adotada para as bases nitrogenadas

Base	Codificação
A	001
C	010
T	011
G	100
N	101

Fonte: Autor

A utilização da placa DEO-CV também resultou em uma restrição de projeto pois a mesma não possui memória lógica programável o suficiente para conseguir trabalhar com sequências grandes com tamanho superior a 8000 bases. Para contornar esta restrição, foi necessário então realizar modificações pontuais no projeto do algoritmo desta versão, de tal forma que as sequências com tamanho superior a 8000 bases fossem corrigidas em *software*, já as sequências com tamanho inferior foram corrigidas em *hardware*. No contexto dos arquivos selecionados para este experimento, cerca de 10% das sequências grandes tinham tamanho superior a 8000 bases.

Também foi adotado um modelo de comunicação por duas vias, sendo uma destinada ao dados e a outra destinada a sinais de sincronização entre CPU e FPGA. A importância destes sinais está em mapear eventos importantes como o término de uma sequência pequena, o término de uma sequência grande, o término de um bloco de sequências grandes etc. A seguir são apresentados dois pseudocódigos desta implementação visando apresentar uma visão geral e completa da versão.

Algorithm 8 Algoritmo *Software*

```

1: Carrega arquivo de sequências pequenas
2: Carrega arquivo de sequências grandes
3: Abre arquivo de saída
4: Cria 2 threads

5: procedure THREAD 0
6:   while Restam sequências grandes para serem corrigidas do
7:     while Enviando bloco de sequências grandes para a FPGA do
8:       Lê uma sequência grande do arquivo
9:       if strlen(seq-grande) > SIZE) then
10:        Encaminha sequência para Thread 1
11:      else
12:        Envia sequência grande para FPGA
13:        Envia pacote de controle indicando o fim da sequência grande
14:      Envia pacote de controle indicando o fim do bloco de sequências grandes
15:      while houver sequência pequena do
16:        Envia sequência pequena para FPGA
17:        Envia pacote de controle indicando o fim da sequência pequena
18:      Envia pacote de controle indicando o fim das sequências pequenas
19:      while não receber pacote de controle indicando o fim do bloco de sequências grandes
      corrigidas do
20:        Espera pacote de controle indicando o fim da sequência grande corrigida
21:        Lê sequência grande corrigida
22:        Escreve sequência grande corrigida no arquivo de saída
23:      Envia sinal de término de correção para Thread 1 e FPGA

24: procedure THREAD 1
25:   while Não recebe sinal de fim de correção do
26:     Recebe sequência grande para corrigir da Thread 0
27:     Corrige a sequência grande
28:     Escreve sequência grande corrigida no arquivo de saída
29: Recebe último bloco de sequências grandes corrigidas
30: Fecha arquivo de saída

```

Algorithm 9 Algoritmo *Hardware*

```

1: while não recebe pacote de controle indicando o fim da correção do
2:   while não recebe pacote de controle indicando o fim do bloco de sequência grandes do
3:     while não recebe pacote de controle indicando o fim de uma sequência grande do
4:       Recebe pacote de dados de um sequência grande
5:       Encaminha sequência grande para a memória da FPGA
6:   while não recebe pacote de controle indicando o fim das sequências pequenas do
7:     while não recebe pacote de controle indicando o fim de uma sequência pequena do
8:       Recebe pacote de dados de um sequência pequena
9:       Realiza correção das sequências grandes com a pequenas
10:  while Existe sequência grande corrigida para enviar à CPU do
11:    Envia sequência grande corrigida para a CPU
12:    Envia pacote de controle indicando o fim da sequência grande corrigida
13:  Envia pacote de controle indicando o fim do bloco de sequências grandes corrigidas
14: Recebe último bloco de sequências grandes corrigidas
15: Fecha arquivo de saída

```

6.4.1.1 Implementação do *Software*

Foi desenvolvido um código em linguagem C com o OpenMP para ser executado na CPU, sendo este código responsável pela transmissão e a recepção das sequências para a FPGA além da correção de maneira semelhante a versão *multicore*. Na transmissão ele realiza a leitura do arquivo das sequências grandes e pequenas, codifica-as conforme a codificação apresentada, empacotando-as e escrevendo-as na porta serial utilizando para tal a biblioteca *termios.h*. Na recepção os dados recebidos são decodificados e as sequências corrigidas são escritas no arquivo de saída. O processo de codificação e decodificação é aqui realizado através de operações lógicas de deslocamento de bits. Este procedimento é ilustrado como exemplo na figura 39.

Figura 39: Processo de empacotamento das bases nitrogenadas

A – 001 – 0x01	0000 0001
C – 010 – 0x02	0000 0010
T – 011 – 0x03	0000 0011
G – 100 – 0x04	0000 0100

ex: ATTGCAGT (8x3 = 24 bits = 3 pacotes)

1º pacote	A << 5 or T << 2 or T >> 1
2º pacote	T << 7 or G << 4 or C << 1 or A >> 2
3º pacote	A << 6 or G << 3 or T

A	T	T	G	C	A	G
001	011	01	1	100	01	100
1º pacote				2º pacote		3º pacote

Fonte: Autor

Os códigos correspondentes aos processos de codificação e decodificação são apresentados nos apêndices B e C respectivamente.

6.4.1.2 Implementação do *Hardware*

Devido à codificação adotada e aos recursos limitados da placa FPGA, o circuito de correção implementado nesta placa em VHDL possui algumas especificidades desta implementação.

O circuito receptor realiza um tratamento diferenciado nas sequências, de forma que se a sequência em questão for grande, é necessário armazená-la na memória da placa, visando à economia da lógica programável.

A comparação de cada par de bases nitrogenadas no circuito corretor é realizada com um trio de portas XNOR, devido à codificação adotada, conforme apresentado pela figura 40. O codificador completo, considerando o caractere N codificado é apresentado na figura 41.

Foi necessário que o circuito transmissor enviasse um caractere de quebra de linha no final da transmissão da sequência corrigida, para realizar a leitura da sequência na porta

serial da CPU.

Figura 40: Unidade de comparação entre bases nitrogenadas

Fonte: Autor

Figura 41: Unidade de comparação completa

Fonte: Autor

6.4.2 Estimativa dos Resultados Experimentais

Anterior a etapa de implementação desta versão foi realizado um levantamento teórico dos dados de desempenho da mesma. Análise semelhante foi apresentada em (ALMEIDA; SATO; MIDORIKAWA, 2019). Aqui, o tempo total de correção de uma sequência grande é dado pelo tempo de comunicação entre a CPU e a FPGA para transmitir as sequências somado com o tempo de correção propriamente dito.

Para o cálculo do tempo de comunicação, foi considerado o protocolo RS-232 com o

modo 801 e velocidade de 115.200 bits por segundo. Considerando a codificação de 3 bits por base nitrogenada, são obtidas as seguintes equações para os tempos de recepção e transmissão respectivamente:

$$tempo_{rx} = \frac{n_{bases} * 3}{8} * 11 * \frac{1}{115200} \quad (6.1)$$

$$tempo_{tx} = \frac{n_{bases} * 3}{8} * 23 * \frac{1}{115200} \quad (6.2)$$

Os coeficientes 11 e 23 possuem relação direta com o número de bits a ser recebido/transmitido respectivamente. No caso da transmissão, foi necessário adicionar alguns estados adicionais após o envio de cada pacote, gerando assim um atraso adicional em relação à recepção.

O tempo total necessário para a correção é composto por duas parcelas, que são o tempo de deslizamento da sequência grande na sequência pequena e o tempo da correção (substituição dos bits referentes às bases nitrogenadas).

Como o cálculo da similaridade é realizado em 1 ciclo de *clock*, o tempo de deslizamento é dado pela diferença entre o tamanho da sequência grande e o tamanho da sequência pequena vezes o período do *clock*. Já o tempo de correção é variável, a depender da região em que foi obtida a maior similaridade. Considerando-se o pior caso, seria necessário realizar novamente todo o deslizamento da sequência pequena na grande, para em seguida realizar a correção na região determinada. Considerando que a operação de correção é composta por uma substituição de bits, necessitando de apenas 1 ciclo de *clock* para ser realizada, o cálculo do tempo total de correção na placa FPGA DE0-CV é dado por:

$$tempo_{total} = (tam_{seqgrande} - tam_{seqpequena}) * 20ns * 2 + 20ns \quad (6.3)$$

Visando a comparação entre os três tempos em questão (transmissão, correção e recepção), será estabelecido um caso exemplo para a correção de uma sequência grande de 1000 bases nitrogenadas por uma sequência pequena de 100 bases. Neste caso, substituindo estes valores nas equações anteriores, são obtidos os seguintes tempos:

- Transmissão (seq. grande): 75 ms
- Recepção (seq. grande): 36 ms
- Recepção (seq. pequena): 4 ms

- Correção: 36 us

Observa-se então que o tempo de correção é cerca de 110 vezes inferior ao tempo de transmissão de uma sequência pequena, de maneira que ele pode ser descartado no cálculo do tempo total de execução da ferramenta.

Assim, o cálculo do tempo de execução é dado pela soma dos tempos de correção de um bloco de sequências grandes. Desta forma, o tempo de correção de um bloco pode ser obtido pela seguinte equação:

$$t_{bloco} = t_{rx_{bloco}} + t_{rx_{seqpeq}} * n_{seqpeq} + t_{tx_{bloco}} \quad (6.4)$$

E o tempo total é dado pela equação:

$$t = t_{bloco} * \frac{tam_{arquivo}}{tam_{bloco}} \quad (6.5)$$

Com o intuito de confirmar as equações 6.4 e 6.5, foi realizado um estudo experimental reduzido, limitando-se neste momento a executar apenas uma parte da ferramenta. Este estudo consistiu na transmissão de uma sequência grande (1000 bases) e uma pequena (100 bases) para a FPGA, seguido da correção e posterior transmissão da sequência grande corrigida para a CPU. O tempo total de execução neste caso foi capturado no osciloscópio Tektronix TBS 1052B, e é apresentado na figura 42.

Figura 42: Tempo total do estudo experimental

Fonte: Autor

Observa-se que o tempo experimental obtido aqui foi de 121 ms. Substituindo os dados deste estudo nas equações 6.4 e 6.5, obtém-se o tempo de 115 ms. Os tempos em questão possuem um desvio percentual de apenas 5,2%, indicando que a análise teórica da ferramenta está razoável.

Assim, foram levantadas curvas de escalabilidade teóricas, seguindo o mesmo modelo das outras versões. Os resultados são apresentados nas figuras 43 e 44.

Figura 43: Teste de escalabilidade do tamanho do arquivo PacBio para versão para *multicore* com FPGA da ferramenta

Fonte: Autor

Figura 44: Teste de escalabilidade do tamanho do arquivo Illumina para versão para *multicore* com FPGA da ferramenta

Fonte: Autor

6.4.3 Análise dos Resultados

O tempo médio obtido a partir da média de 10 tempos foi de 649 segundos e o desvio padrão foi de 1,2, tendo um *speedup* de 42,4 em relação à versão sequencial. Foram identificados alguns problemas relacionados a implementação na placa FPGA DE0-CV que serão tratados em seção posterior.

6.5 Implementação da Ferramenta para *Cluster* com FPGAs

A última implementação da ferramenta é uma união das implementações anteriores (*multicore* com FPGA e *cluster*), e não foi possível realizar a coleta dos seus resultados experimentais. Ela consiste em realizar a distribuição das sequências grandes para os nós da rede utilizando o MPI, análogo ao que é realizado na versão para *cluster* da ferramenta, porém, ao invés da correção ser feita por *threads* criadas no nó, ela é realizada principalmente dentro da FPGA. A comunicação entre a CPU e a FPGA ocorre exatamente da mesma forma que a versão *multicore* com FPGA da ferramenta assim como o circuito responsável pelo processamento dentro da FPGA.

Aqui é possível realizar uma aproximação grosseira dos resultados experimentais. Levando-se em consideração que os nós dos *cluster* são representações uniformes de um mesmo ambiente computacional, estando conectados todos com uma mesma placa FPGA, haverá uma redução no tempo total de execução por um fator n , sendo este o número de nós do *cluster*.

Assim:

$$T_{clusterFPGA} = T_{multicoreFPGA}/n \quad (6.6)$$

6.6 Comparação dos Resultados

A tabela 7 apresenta um resumo dos resultados de todas as versões das ferramentas.

Tabela 7: Comparação dos resultados de teste das versões da ferramenta

Versão	Tempo (s)	<i>Speedup</i>
sequencial	27.532,4	1
<i>multicore</i> (4 cores)	7.082	3,88
<i>cluster</i> (20 cores)	1.587,8	17,3
<i>multicore</i> com FPGA	649	42,42

Fonte: Autor

Observa-se que a versão *multicore* com FPGA apresenta um desempenho consideravelmente superior que a versão para *cluster*, mostrando a importância da aplicação dos aceleradores na otimização de algoritmos.

Baseado no menor tempo de execução da versão *multicore* com FPGA, neste cenário de testes, foi realizada a correção completa nesta versão, ou seja, considerando o tamanho total de ambos os arquivos de sequências genômicas.

Foi realizado um corte no arquivo das sequências PacBio, retirando toda as sequências com tamanho inferior a 1000 bases, pois elas são úteis para o processo de autocorreção, porém não são utilizadas na montagem devido ao tamanho reduzido. Além disso, foi utilizada uma cobertura de 20x das sequências Illumina, conforme apontado por (FU; WANG; AU, 2019) como sendo o valor de saturação para a correção híbrida.

Após o processo completo de correção, foi realizada a montagem do genoma utilizando as ferramentas de montagem descritas anteriormente. O genoma montado foi então alinhado com o genoma de referência e foi verificada sua semelhança com base em relatório emitido pelo BLAST. Todo o processo foi realizado numa máquina com as seguintes configurações: processador Intel Xeon E5620 com *clock* de 2,4 GHz e 8 *cores*. A memória é de 24 GB de RAM e o sistema operacional é o CentOS 7.2.1511. Cabe ressaltar que no caso da ferramenta aqui desenvolvida a correção foi feita no ambiente *multicore* com FPGA enquanto a montagem foi feita neste ambiente descrito.

A tabela 8 sumariza estes resultados, apresentado os tempos totais de execução (correção + montagem) juntamente com a similaridade entre o genoma montado e o genoma de referência. O tempo total da ferramenta Lordec foi calculado a partir do tempo de correção da mesma, retirado do artigo correspondente somado com o tempo de montagem utilizando a ferramenta Canu. Não foi possível encontrar o tempo de execução da ferramenta MARVEL em nenhum artigo científico, nem executá-la no ambiente com-

putacional em questão, por isso o mesmo não é apresentado na tabela.

Tabela 8: Tempo total de execução para montagem da E. coli

Método	Tempo Total (min)	Similaridade
Canu (completo)	28	99,95%
Canu (montagem) + Ferramenta	3312	99,99%
Smartdenovo	2	89%
Smartdenovo + Canu (correção)	50	99,93%
Smartdenovo + Ferramenta	3324	99,98%
LoRDEC	32	99,99%

Fonte: Autor

Como é possível observar, o algoritmo aqui utilizado possui o maior tempo de execução entre os casos apresentados, devido ao seu alto custo computacional e por se tratar de um algoritmo não otimizado para este propósito. Porém, ele resulta em um genoma mais próximo do genoma de referência dentre os casos apresentados, indicando que existe margem para melhoria das ferramentas computacionais já existentes. Além disso, é importante destacar as limitações resultantes da FPGA utilizada no cenário experimental, onde foi utilizada uma FPGA de propósito acadêmico, ao invés de uma para projetos de alto desempenho. Desta forma, faz-se necessário destacar três das limitações que possuem maior impacto no tempo de execução da versão híbrida da ferramenta:

- **Modo de comunicação:** O modo de comunicação utilizado neste projeto foi o RS-232, devido a limitações da placa FPGA utilizada. Este modo possui velocidades de comunicação mais baixas que outros modos elencados anteriormente, fazendo com que o custo da comunicação prevaleça no custo computacional do algoritmo.
- **Número de mensagens trocadas:** Devido a pequena quantidade da memória de bits da FPGA utilizada, foi necessário reenviar as sequências pequenas após a correção de cada bloco de sequências grandes, aumentando o custo computacional do algoritmo. Uma maneira de contornar o problema da baixa quantidade de memória é utilizar a memória externa periférica que as placas FPGAs possuem. Esta memória costuma ter tamanho superior à memória de bits da placa FPGA, porém é necessário o projeto de um controlador para utilizá-la.
- **Número de circuitos de correção:** O número de circuitos de correção possui relação direta com a lógica programável da FPGA. No contexto deste projeto não foi possível

utilizar mais de um circuito, pois a lógica programável da FPGA foi utilizada majoritariamente no circuito de comunicação.

Havendo a possibilidade de contornar estas três limitações citadas é possível obter ganhos significativos no desempenho da ferramenta. Assim, é apresentada a seguir uma estimativa de resultados futuros, em um cenário onde as limitações foram removidas.

6.7 Estimativas de Resultados Futuros

Visando apresentar um horizonte de melhoria imediata para o tempo de execução da ferramenta, são realizadas aqui estimativas de resultados futuros. Esta seção é dividida de tal forma a apresentar soluções de projeto e estimativas para cada limitação citada anteriormente. Por fim, é feita uma análise do resultado previsto, considerando a implementação de todas as soluções. As estimativas aqui realizadas serão feitas com base em uma placa FPGA voltada para projetos de alto desempenho, ou seja, com uma quantidade significativa de cursos internos para uso do projetista.

6.7.1 Mudança no Modo de Comunicação

Conforme discutido anteriormente, a comunicação via PCI *Express* permite taxas de comunicação da ordem de GB/s, porém, essa velocidade acaba sendo limitada pelo próprio clock da placa FPGA em questão. Placas FPGA voltadas para projetos de alto desempenho costumam possuir clocks com frequências da ordem de 150 MHz, desta forma, a velocidade de comunicação pode ser limitada para 150 Mb/s. Com base nesta velocidade, é possível estimar os novos tempos de transmissão e recepção a partir das seguintes equações:

$$tempo_{rx} = \frac{n_{bases} * 3}{8} * 11 * \frac{1}{150000000} \quad (6.7)$$

$$tempo_{tx} = \frac{n_{bases} * 3}{8} * 23 * \frac{1}{150000000} \quad (6.8)$$

Destaca-se aqui que a estimativa em questão é grosseira, pois não leva em consideração características específicas da implementação do circuito transmissor/receptor utilizando o PCI *Express*.

6.7.2 Redução no Número de Mensagens Trocadas

Conforme dito anteriormente, o número de mensagens trocadas possui relação direta e inversamente proporcional com a memória da FPGA. Aqui é proposto o uso da memória periférica presente na placa FPGA, sendo possível então armazenar blocos de sequências pequenas, de maneira análoga com o que é feito com os blocos de sequências grandes. A diferença neste caso está na necessidade do uso de um controlador para acessar tal memória periférica e no tamanho fixo de palavra, ou seja, cada sequência pequena deverá ser quebrada e armazenada em mais de um endereço na memória.

Propõe-se aqui um modelo de uso de *flags*. A *flag* é um bit que tem como propósito indicar se determinado endereço de memória apresenta o fim de uma sequência pequena ou não. Para tal, pode-se utilizar o valor 1 na *flag* para indicar o término da sequência ou o valor 0 caso contrário.

A figura 45 apresenta uma situação onde a memória possui 16 bits de tamanho de palavra. Considerando a codificação de três bits adotada neste projeto, é possível armazenar cinco bases e o bit correspondente à *flag* em cada endereço. Na figura em questão estão armazenadas duas sequências pequenas na memória.

Figura 45: Armazenamento das sequências pequenas na memória periférica

A	C	A	C	T	0
...					0
T	G	G	A	T	1
G	T	A	A	A	0
...					0
C	C				1

Fonte: Autor

Como as sequências pequenas são quebradas para serem armazenadas na memória é necessário realizar a união das partes antes de seu uso na correção. Essa união é feita a partir de acessos sucessivos a endereços contínuos da memória periférica, por meio do controlador. Neste cenário, o tempo de preparação de uma sequência pequena (união de suas partes) é dado por:

$$tempo_{preparacao} = T_{clock} * \frac{tamanho_{sequencia}}{5} \quad (6.9)$$

6.7.3 Aumento do Número de Circuitos de Correção

Com o uso de uma FPGA voltada para projetos de alto desempenho, é possível criar diversos circuitos responsáveis pela correção, realizando assim o processo de correção de múltiplas sequências grandes em paralelo. Na equação 6.3 foi apresentado o pior caso para o tempo de correção de uma sequência grande utilizando-se uma sequência pequena. Expandindo o conceito tratado, é possível se estimar o tempo total de correção de uma sequência grande como sendo:

$$t_{correcao1seq} = ((tam_{seqgrande} - tam_{seqpequena}) * 20ns * 2 + 20ns) * n_{seqspequenas} \quad (6.10)$$

E o tempo total de correção considerando apenas um circuito de correção é dado por:

$$t = t_{correcao1seq} * n_{seqsgrandes} \quad (6.11)$$

Por fim é estimado o tempo de correção paralela, considerando um número n de circuitos de correção:

$$t_{par} = \frac{t}{n} \quad (6.12)$$

6.7.4 Análise de Resultados Estimados

Para finalizar esta seção, é apresentada uma análise de resultados estimados com base em placas FPGA para projetos de alto desempenho, utilizando-se as equações apresentadas anteriormente no cenário experimental deste projeto (correção do genoma da *E. coli*). Neste cenário é possível enviar todas as sequências pequenas em um bloco apenas para a FPGA, pois as mesmas cabem na memória periférica da placa. Assim, tanto as sequências pequenas quanto as grandes só precisarão ser enviadas uma vez para a placa FPGA. Portanto, é feito o cálculo dos tempos de recepção das sequências grandes, recepção das sequências pequenas e recepção das sequências grandes levando em consideração este contexto. Também é utilizada aqui a comunicação via PCI *Express*. A seguir são apresentados os tempos de comunicação calculados:

- Tempo de recepção das sequências grandes: 2,75 s
- Tempo de recepção das sequências pequenas: 2,45 s
- Tempo de transmissão das sequências grandes: 5,75 s

Observa-se aqui que os tempos de comunicação possuem reduções significativas implementando-se as mudanças em questão, principalmente devido ao uso da comunicação via PCI *Express*, que neste caso é da ordem de 1000x mais rápida que o RS-232.

O tempo de correção de uma sequência grande é feito por aproximação, considerando-se uma sequência grande de tamanho 1000, 900.000 sequências pequenas de tamanho 100 e o clock de 150 MHz da FPGA. O tempo obtido é de 10,8s. Este tempo não leva em consideração o tempo para se montar uma sequência pequena na memória, pois o mesmo é desprezível neste caso.

É identificado que nesta situação o tempo de correção de uma sequência grande é superior à soma de todos os tempos de comunicação. Assim, é possível atribuir o tempo total de execução da ferramenta como sendo o tempo necessário para corrigir todas as sequências grandes, podendo desconsiderar o tempo de comunicação. Este já foi estimado pelas equações 6.11 e 6.12; dessa forma, é realizada novamente uma aproximação, considerando 100.000 sequências grandes no arquivo e mantendo-se o número n de circuitos de correção. O tempo total de correção é então dado por:

$$t_{par} = \frac{1.080.000}{n} \quad (6.13)$$

Considerando o valor $n = 1000$, o que é possível em placas FPGA voltadas para projetos de alto desempenho, obtém-se o valor de 1080 segundos, ou seja, 18 minutos para a correção das sequências.

Por fim, é apresentada na tabela 9 a mesma tabela da seção 6.6, porém agora com os valores estimados do tempo de execução. Novamente aqui é apresentado o tempo total, ou seja, o tempo da correção somado com o tempo da montagem do genoma.

Tabela 9: Tempo total de execução para montagem da E. coli considerando resultados estimados

Método	Tempo Total (min)	Similaridade
Canu (completo)	28	99,95%
Canu (montagem) + Ferramenta (estimativa)	28	99,99%
Smartdenovo	2	89%
Smartdenovo + Canu (correção)	50	99,93%
Smartdenovo + Ferramenta (estimativa)	40	99,98%
LoRDEC	32	99,99%

Fonte: Autor

Observa-se que neste cenário a ferramenta aqui desenvolvida possui melhor qualidade dentre as demais e um tempo de execução da mesma ordem. Ela também ganha em tempo de execução da ferramenta LoRDEC, quando ambas são executadas em conjunto com a ferramenta Canu, sendo a LoRDEC uma ferramenta do estado-da-arte utilizada pela comunidade científica, além de necessitar do genoma de referência do ser vivo a ser montado.

As estimativas aqui realizadas foram baseadas em um ambiente com uma placa FPGA apenas. Em um ambiente de *cluster* com FPGAs é possível aplicar novamente estimativa apresentada na equação 6.6, onde o tempo de execução é dividido por um fator n , sendo este o número de nós do *cluster*.

7 CONCLUSÕES

O custo computacional de um algoritmo está diretamente relacionado ao volume de dados utilizados por ele e à complexidade de suas operações. No caso da montagem de genoma, ambos os fatores estão presentes em demasia, sendo esta uma área propícia para o emprego de técnicas de computação de alto desempenho.

Este trabalho apresentou um primeiro passo na aplicação de aceleradores implementados em *hardware* utilizando FPGAs, em um contexto tão grande quanto a correção de sequências genômicas. Os resultados aqui apresentados apontam que a adição de um acelerador implementado em *hardware* promove ganhos significativos no processo de correção das sequências genômicas, atendendo à hipótese levantada durante a etapa inicial deste trabalho. Entende-se aqui este trabalho como um primeiro passo para um emprego mais robusto e eficaz do *hardware* neste contexto.

Também foi identificada a possibilidade de melhoria das ferramentas já existentes no estado da arte, melhorias estas possíveis com o uso de aceleradores. Uma redução significativa no tempo de execução das ferramentas viabilizaria uma melhoria no algoritmo base de correção adotado, utilizando-se operações de custo computacional mais elevado, ou então o uso de um maior volume de dados. Ambos os fatores por sua vez contribuem para um genoma montado com maior fidelidade ao original. Por mais que o tempo de execução da versão da ferramenta aqui apresentada seja consideravelmente superior aos das ferramentas já utilizadas pela comunidade científica, as estimativas feitas baseadas em uma infraestrutura computacional com maior quantidade de recursos apontam um horizonte favorável no quesito de se assemelhar o tempo de execução.

Porém, entende-se que as contribuições deste trabalho não se resumem apenas ao contexto da bioinformática. Foi possível aqui explorar o aspecto do desempenho em diferentes arquiteturas, sendo elas de memória compartilhada, de memória distribuída, de memória compartilhada e distribuída e por fim uma arquitetura heterogênea. Cada situação apresenta um desafio diferente na implementação de uma versão paralela do algoritmo, visando explorar ao máximo os recursos fornecidos pelo ambiente computacional.

Além disso, foram apresentados os desafios da implementação de um algoritmo em *hardware*, onde o aspecto da comunicação entre a CPU e o acelerador pode facilmente se tornar o gargalo de tal algoritmo. Os recursos de projeto aqui levantados, para viabilizar a implementação deste algoritmo, o qual possui um grande volume de dados na FPGA (que por sua vez possui limitação em sua memória), podem ser facilmente aproveitados em outros projetos correlatos.

Também foram publicados dois artigos relacionados aos dados obtidos neste trabalho. O primeiro deles no 18º WPerformance e o segundo no XX WSCAD, ambos congressos de relevância nacional para a área de alto desempenho. Com base nas oportunidades de trabalhos futuros apresentadas a seguir, espera-se incrementar a atual pesquisa, visando à elaboração futura de um artigo para periódico internacional.

Por fim, ressalta-se que trabalhos envolvendo a computação de alto desempenho podem evoluir continuamente. A necessidade constante de se repensar os algoritmos já existentes e de se explorar ao máximo os recursos disponíveis, ligada ao advento da computação heterogênea, que introduz os aceleradores em um cenário que antes era predominantemente composto por CPUs, resultam em inúmeras possibilidades de melhorias, o que será apresentado a seguir como potencialidades em trabalhos futuros.

7.1 Trabalhos Futuros

Dentre as diversas oportunidades de trabalhos futuros, as que se apresentam como mais imediatas são:

1. Implementação do algoritmo no *cluster* com FPGAs: Devido a limitações tanto no quesito tempo, quanto no contexto social corrente (pandemia COVID-19), não foi possível realizar a integração das FPGAs no *cluster*. Esta implementação é de suma importância para se confirmar as estimativas de resultados aqui levantados.
2. Modificação do algoritmo de correção: O algoritmo de correção aqui apresentado não cobre as três possíveis operações de correção (substituição, adição e remoção de bases). No contexto do trabalho aqui apresentado, foi utilizado um caso de teste onde apenas a operação de substituição de bases era necessária para realizar a correção das sequências genômicas. A modificação deste algoritmo, através de um incremento de operações, é necessária para se generalizar a ferramenta aqui apresentada, de tal forma que ela possa montar qualquer genoma, de qualquer ser vivo.

3. Montagem do genoma de outros seres vivos: Neste trabalho foi feita a montagem do genoma da *E. coli*, seguindo o modelo dos artigos científicos publicados na área. A montagem do genoma de outros seres vivos com maior complexidade genética segue a linha de se apresentar a ferramenta aqui desenvolvida como genérica.
4. Comparação de desempenho com GPUs: Observa-se ainda, na comunidade científica de alto desempenho, uma predominância do uso das GPUs como aceleradores. Uma comparação entre o mesmo algoritmo base implementado em FPGA e em GPU forneceria subsídios práticos para se justificar o tipo de escolha aqui adotada, visando a não se limitar apenas ao fundamento teórico por trás de ambos os aceleradores.
5. Utilização de outras FPGAs: A placa FPGA utilizada no contexto deste trabalho é de propósito acadêmico, possuindo uma quantidade consideravelmente inferior de recursos quando comparada a uma mesma placa para projetos de alto desempenho. A utilização de outras placas FPGAs, visando comprovar as estimativas realizadas na seção 6.7, faz-se necessária para valorizar a ferramenta aqui desenvolvida.
6. Mudança do algoritmo de correção: Neste trabalho foi implementado um algoritmo básico de correção por janela deslizante, devido a uma menor curva de dificuldade para sua implementação em *hardware*. O uso de algoritmos mais sofisticados, envolvendo grafos e sua respectiva implementação em *hardware*, poderiam melhorar ainda mais o desempenho da ferramenta.

REFERÊNCIAS

- ALBERTS, B. et al. *Molecular biology of the cell*. [S.l.]: Garland Science, Taylor and Francis Group,, 2018.
- ALMEIDA, F.; SATO, L.; MIDORIKAWA, E. Análise de viabilidade de ferramenta para correção híbrida de sequências genômicas em ambiente de memória compartilhada com fpga. In: *SIMPÓSIO EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO, 20. Anais...* Porto Alegre, RS, Brasil: SBC, 2019. p. 430–437. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/wscad/article/view/8688>.
- ALMEIDA, F. et al. Análise de desempenho de algoritmos para correção híbrida de sequências genômicas em ambiente de memória compartilhada e distribuída. In: *WORKSHOP EM DESEMPENHO DE SISTEMAS COMPUTACIONAIS E DE COMUNICAÇÃO, 18. Anais...* Porto Alegre, Brasil: SBC, 2019. ISSN 2595-6167. Disponível em: <https://sol.sbc.org.br/index.php/wperformance/article/view/6470>.
- 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.
- BESTA, M. et al. Graph processing on fpgas: Taxonomy, survey, challenges. *arXiv preprint arXiv:1903.06697*, 2019.
- BLAST. 2020. <https://blast.ncbi.nlm.nih.gov/Blast.cgi>. Acesso: 02/10/2020.
- 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.

- 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.
- FROST, R. Mending wall. *North of Boston*, v. 12, 1914.
- FU, S.; WANG, A.; AU, K. F. A comparative evaluation of hybrid error correction methods for error-prone long reads. *Genome biology*, Springer, v. 20, n. 1, p. 26, 2019.
- 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.
- HENNESSY, J. L.; PATTERSON, D. A. *Computer architecture: a quantitative approach*. 5th. ed. [S.l.]: Morgan Kaufmann, 2012.
- Hill, K. et al. Comparative analysis of opencl vs. hdl with image-processing kernels on stratix-v fpga. In: *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. [S.l.: s.n.], 2015. p. 189–193.
- HU, Y.; GEORGIOU, P. A real-time de novo dna sequencing assembly platform based on an fpga implementation. *IEEE/ACM transactions on computational biology and bioinformatics*, IEEE, v. 13, n. 2, p. 291–300, 2015.
- HUFFSTETLER, J. *Intel Processors and FPGAs—Better Together*. 2018. (<https://itpeernetwork.intel.com/intel-processors-fpga-better-together/>). Acesso: 05/09/2020.
- INTEL. Cyclone V device overview. *San Jose, CA, USA*, 2018.
- 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.
- MAHRAM, A.; HERBORDT, M. C. FMSA: FPGA-accelerated clustalw-based multiple sequence alignment through pipelined prefiltering. In: *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. [S.l.: s.n.], 2012. p. 177–183.
- 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.

- NELSON, D. L.; COX, M. M. *Princípios de Bioquímica de Lehninger-7*. [S.l.]: Artmed Editora, 2018.
- NOWOSHILOW, S. et al. The axolotl genome and the evolution of key tissue formation regulators. *Nature*, v. 554, n. 7690, p. 50, 2018.
- PACBIO, I. Understanding accuracy in dna sequencing. *Pacific Biosciences*, 2020.
- PAREEK, C. S.; SMOCZYNSKI, R.; TRETYN, A. Sequencing technologies and genome sequencing. *Journal of applied genetics*, Springer, v. 52, n. 4, p. 413–435, 2011.
- PEDRONI, V. A. *Circuit design with VHDL*. [S.l.]: MIT press, 2004.
- QASAIMAH, M. et al. Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels. In: IEEE. *2019 IEEE International Conference on Embedded Software and Systems (ICESSE)*. [S.l.], 2019. p. 1–8.
- RAMACHANDRAN, A. et al. FPGA accelerated dna error correction. In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. [S.l.: s.n.], 2015. p. 1371–1376. ISSN 1530-1591.
- RÉGNIER, M. Knuth-morris-pratt algorithm: an analysis. In: SPRINGER. *INTERNATIONAL SYMPOSIUM ON MATHEMATICAL FOUNDATIONS OF COMPUTER SCIENCE. Proceedings...* [S.l.], 1989. p. 431–444.
- 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*. 2015. <https://github.com/ruanjue/smartdenovo>. Acesso: 17/06/2020.
- SALAMAT, S.; ROSING, T. Fpga acceleration of sequence alignment: A survey. *arXiv preprint arXiv:2002.02394*, 2020.
- 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.
- SNIR, M. et al. *MPI-The Complete Reference, Volume 1: The MPI Core*. 2nd.). ed. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262692155.
- STONE, J. E.; GOHARA, D.; SHI, G. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, IEEE Computer Society, v. 12, n. 3, p. 66–73, 2010.
- TAN, G. et al. Long fragments achieve lower base quality in illumina paired-end sequencing. *Scientific reports*, Nature Publishing Group, v. 9, n. 1, p. 1–7, 2019.

TEXAS, I. Max3232 3-v to 5.5-v multichannel rs-232 line driver/receiver with ± 15 -kv esd protection. *Dallas, Texas: Texas Instruments*, 2017.

VARMA, B. S. C. et al. FAssem: FPGA based acceleration of de novo genome assembly. In: *IEEE ANNUAL INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES*, 21. *Proceedings*. [S.l.: s.n.], 2013. p. 173–176.

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.

VOET, D.; VOET, J. G. *Biochemistry*. [S.l.]: John Wiley & Sons, Inc, 2011.

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.

APÊNDICE A – CONFIGURAÇÃO DA PORTA SERIAL VIA TERMIO.S.H

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <termios.h>
4 #include <unistd.h>
5 #include <errno.h>
6
7 int main(int argc, char *argv[])
8 {
9     int fd;
10
11     fd = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY);
12
13     if(fd == -1){
14         printf("Erro na abertura da porta");
15         return 0;
16     }
17     else
18         printf("Porta aberta com sucesso");
19
20
21     struct termios SerialPort;
22     tcgetattr(fd, &SerialPort);
23
24     cfsetispeed(&SerialPort, B115200); // BaudRate de leitura como 115200
25     cfsetospeed(&SerialPort, B115200); // BaudRate de escrita como 115200
```

```
26
27 // Configuracao para modo 801
28 SerialPort.c_cflag |= PARENB;
29 SerialPort.c_cflag |= PARODD;
30 SerialPort.c_cflag &= ~CSTOPB;
31 SerialPort.c_cflag &= ~CSIZE;
32 SerialPort.c_cflag |= CS8;
33
34 SerialPort.c_cflag &= ~CRTSCTS;
35 SerialPort.c_cflag |= CREAD | CLOCAL;
36
37 SerialPort.c_iflag &= ~(IXON | IXOFF | IXANY);
38 SerialPort.c_iflag &= ~(ICANON | ECHO | ECHOE | ISIG);
39
40 SerialPort.c_oflag &= ~OPOST;
41
42 SerialPort.c_cc[VMIN] = 10;
43 SerialPort.c_cc[VTIME] = 0;
44
45 // Seta os atributos na porta serial
46 if((tcsetattr(fd, TCSANOW, &SerialPort)) != 0){
47     printf("Erro na configuracao dos atributos");
48     return 0
49 }
50
51 tcflush(fd, TCIFLUSH); // Limpa o buffer de recepcao
52
53 //Codigo utilizando a porta serial
54
55 close(fd);
56 return 0;
57 }
```

APÊNDICE B – CODIFICAÇÃO DAS SEQUÊNCIAS GENÔMICAS

```
1  #define A 0x01
2  #define C 0x02
3  #define T 0x03
4  #define G 0x04
5  #define N 0x05
6
7  void codifica(unsigned char seq[], unsigned char codificada[])
8  {
9      int i = 0;
10     int j = 0;
11     int resto;
12     unsigned char pivochar[8];
13
14     // Primeira parte do algoritmo trabalha com blocos de 8 caracteres
15     for(j = 0; (j+1)*8 <= strlen(seq); j++){
16         for(i = 0; i < 8; i++){
17             if (seq[j*8+i] == 'A')
18                 pivochar[i] = A;
19             if (seq[j*8+i] == 'C')
20                 pivochar[i] = C;
21             if (seq[j*8+i] == 'T')
22                 pivochar[i] = T;
23             if (seq[j*8+i] == 'G')
24                 pivochar[i] = G;
25             if (seq[j*8+i] == 'N')
```

```

26         pivochar[i] = N;
27     }
28
29     codificada[3*j+0] = pivochar[0] << 5 | pivochar[1] << 2 | pivochar[2]
        >> 1;
30     codificada[3*j+1] = pivochar[2] << 7 | pivochar[3] << 4 | pivochar[4]
        << 1 | pivochar[5] >> 2;
31     codificada[3*j+2] = pivochar[5] << 6 | pivochar[6] << 3 | pivochar[7];
32 }
33
34 // Segunda parte do algoritmo trabalha com os caracteres restantes
35 memset(pivochar, 0x00, 8);
36 resto = strlen(seq) % 8;
37
38 if (resto != 0){
39     for(i = 0; i < resto; i++){
40         if (seq[j*8+i] == 'A')
41             pivochar[i] = A;
42         if (seq[j*8+i] == 'C')
43             pivochar[i] = C;
44         if (seq[j*8+i] == 'T')
45             pivochar[i] = T;
46         if (seq[j*8+i] == 'G')
47             pivochar[i] = G;
48         if (seq[j*8+i] == 'N')
49             pivochar[i] = N;
50     }
51
52     codificada[3*j+0] = pivochar[0] << 5 | pivochar[1] << 2 | pivochar[2]
        >> 1;
53     codificada[3*j+1] = pivochar[2] << 7 | pivochar[3] << 4 | pivochar[4]
        << 1 | pivochar[5] >> 2;
54     codificada[3*j+2] = pivochar[5] << 6 | pivochar[6] << 3 | pivochar[7];
55 }
56
57 codificada[strlen(codificada)] = '\n';
58 }

```

APÊNDICE C – DECODIFICAÇÃO DAS SEQUÊNCIAS GENÔMICAS

```
1  #define A 0x01
2  #define C 0x02
3  #define T 0x03
4  #define G 0x04
5  #define N 0x05
6
7  void decodifica(unsigned char codificada[], unsigned char seq[])
8  {
9      int j = 0;
10     int resto;
11
12     for(j = 0; (j+1)*3 <= strlen(codificada); j++){
13         seq[8*j+0] = codificada[3*j+0] >> 5;
14         seq[8*j+1] = (codificada[3*j+0] >> 2) & 0x07;
15         seq[8*j+2] = (codificada[3*j+0] << 1 | codificada[3*j+1] >> 7) & 0x07;
16         seq[8*j+3] = (codificada[3*j+1] >> 4) & 0x07;
17         seq[8*j+4] = (codificada[3*j+1] >> 1) & 0x07;
18         seq[8*j+5] = (codificada[3*j+1] << 2 | codificada[3*j+2] >> 6) & 0x07;
19         seq[8*j+6] = (codificada[3*j+2] >> 3) & 0x07;
20         seq[8*j+7] = codificada[3*j+2] & 0x07;
21     }
22
23     resto = strlen(codificada) % 3;
24
25     if (resto != 0){
```

```
26     seq[8*j+0] = codificada[3*j+0] >> 5;
27     seq[8*j+1] = (codificada[3*j+0] >> 2) & 0x07;
28     seq[8*j+2] = (codificada[3*j+0] << 1 | codificada[3*j+1] >> 7) & 0x07;
29     seq[8*j+3] = (codificada[3*j+1] >> 4) & 0x07;
30     seq[8*j+4] = (codificada[3*j+1] >> 1) & 0x07;
31     seq[8*j+5] = (codificada[3*j+1] << 2 | codificada[3*j+2] >> 6) & 0x07;
32     seq[8*j+6] = (codificada[3*j+2] >> 3) & 0x07;
33     seq[8*j+7] = codificada[3*j+2] & 0x07;
34 }
35
36 }
```