Análise de desempenho de algoritmos para correção híbrida de sequências genômicas em ambiente de memória compartilhada e distribuída

Felipe V. de Almeida[1], Liria M. Sato[1], Edson T. Midorikawa[1], Tatiana T. Torres[2]

[1]Escola Politécnica – Universidade de São Paulo (USP)

[2]Instituto de Biociências – Universidade de São Paulo (USP)

{felipe.valencia.almeida,liria.sato,emidorik}@usp.br

{tttorres}@ib.usp.br

Abstract. *Genome analysis is an area with extensive research because it allows the study of diseases and the development of new treatments. To do this, researchers use the genome, assembled with computational tools to perform their analysis. This work presents a performance analysis of a hybrid correction algorithm for genome sequences, this being a necessary stage for the assembly of the genome. Seven versions of the algorithm were implemented to compare their performance. The results obtained from the tests show that it is possible to obtain performance gains of up to about 17 times in relation to the sequential version, and that the best version of the algorithm has scalability higher than linear.*

Resumo. *A análise do genoma é uma área com amplas pesquisas que permitem o estudo de doenças e o desenvolvimento de novos tratamentos. Para isso, pesquisadores utilizam-se do genoma montado através de ferramentas computacionais para realizar sua análise. Este trabalho apresenta uma análise de desempenho acerca de um algoritmo de correção híbrida de sequências genômicas, sendo esta uma etapa necessária para a montagem do genoma. Foram implementadas sete versões do algoritmo visando comparar seus desempenhos. Os resultados obtidos a partir dos testes revelam que é possível obter ganhos de desempenho de até cerca de 17 vezes em relação à versão sequencial, e que a melhor versão do algoritmo possui escalabilidade superior à linear.*

1. Introdução

A bioinformática é uma área interdisciplinar que tem como propósito a aplicação de algoritmos computacionais para a solução de problemas biológicos. Suas pesquisas em geral são voltadas à Genômica, como a identificação de proteínas, identificação de padrões de aminoácidos, elaboração de árvores filogenéticas e análise de DNA, sendo esta última o escopo deste trabalho.

O genoma é a composição de todo o DNA presente em um ser vivo. O DNA é condensado nos cromossomos, presentes nas células do ser vivo. Ele é composto pelas bases nitrogenadas, que podem ser Adenina (A), Citosina (C), Guanina (G) ou Timina (T). O genoma se manifesta pelo genótipo, ou seja, a composição genética do ser vivo, que

através da interação com o ambiente resulta em seu fenótipo, que são suas características observáveis.

O estudo do genoma é necessário para o entendimento de um ser vivo. Através dele é possível observar padrões entre as espécies, identificar doenças e, por consequência, seus tratamentos, além de melhorar o entendimento do organismo. Este processo, que antes era dificultado pelo alto custo financeiro e computacional, vem sendo viabilizado cada vez mais através de esforços conjuntos entre pesquisadores de diversas áreas [Goodwin et al. 2016].

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

Trabalhos como este só são possíveis através da montagem de múltiplos genomas de uma mesma espécie. Ou seja, um processo que já é custoso computacionalmente precisa ser executado múltiplas vezes, visando fornecer subsídios para os pesquisadores da área. Considerando que a montagem de um genoma é um processo que possui tempo variável de minutos a até meses, dependendo da complexidade do ser vivo e dos recursos computacionais disponíveis [Del Angel et al. 2018], faz-se necessário um estudo de desempenho, visando à criação de algoritmos otimizados para este propósito.

No processo de obtenção do genoma de um ser vivo inicialmente ocorre o sequenciamento do DNA, onde o material genético do ser vivo é clonado e introduzido em uma máquina denominada sequenciador. O sequenciador utiliza-se de processos bioquímicos para quebrar o DNA em regiões aleatórias, gerando fragmentos, processo este conhecido como sequenciamento *shotgun*, e armazenar as informações correspondentes em um arquivo de saída. O arquivo de saída aqui possui um tamanho (quantidades de bases nitrogenadas) superior ao genoma do ser vivo, devido à etapa de clonagem do DNA. A razão entre a quantidade de bases nitrogenadas presentes no arquivo e o genoma do ser vivo é denominada cobertura (*coverage*).

Estes fragmentos então são montados utilizando-se um algoritmo de montagem de DNA que é composto principalmente por técnicas de alinhamento de sequências, visando obter a similaridade entre os fragmentos ou parte deles. A montagem possui um alto custo computacional, pois dependendo da complexidade do ser vivo, o número de fragmentos varia de centenas de milhares a até bilhões, no caso do ser humano [Wheeler et al. 2008].

As duas maiores fabricantes de sequenciadores são a Illumina e a Pacific Biosciences (PacBio), que criaram tecnologias diferentes para o sequenciamento, sendo a tecnologia PacBio mais nova que a Illumina.

O sequenciamento Illumina resulta em pequenos fragmentos contendo em média

100 a 250 bases nitrogenadas e com boa qualidade [Illumina 2010]. A montagem de seus fragmentos é realizada através de grafos De Brujin [Compeau et al. 2011]. Esses fragmentos são denominados na literatura como sequências pequenas (*short reads*).

O sequenciamento PacBio resulta em fragmentos grandes com uma média de 10.000 bases nitrogenadas, porém com baixa qualidade [Rhoads and Au 2015]. O fato dos fragmentos serem maiores quando comparados aos Illumina reduz o custo computacional da montagem, devido ao menor número de fragmentos resultantes; porém, a baixa qualidade pode resultar em montagens errôneas, necessitando então uma etapa de correção anterior à montagem. Esses fragmentos são chamados na literatura de sequências grandes/longas (*long reads*).

A correção das sequências PacBio pode ser de dois tipos. A correção do tipo auto (*self*) utiliza os próprios fragmentos PacBio para realizar a correção neles mesmos, através do consenso entre diferentes fragmentos alinhados. A correção do tipo híbrida (*hybrid*) utiliza fragmentos de outra tecnologia com maior qualidade (em geral Illumina) para corrigir os fragmentos PacBio, também através do alinhamento.

Neste trabalho é proposto um estudo de um algoritmo de correção híbrida de autoria própria, utilizando um algoritmo de janela deslizante. A janela deslizante é um algoritmo trivial que possui resultado ótimo, porém, em determinadas situações sua aplicação não ocorre devido ao alto custo computacional.

2. Trabalhos Relacionados

Os trabalhos relacionados encontrados na literatura estão divididos conforme a tecnologia e a etapa da montagem de DNA. Estes trabalhos resultam em ferramentas utilizadas para propósitos específicos.

[Bolger et al. 2014] desenvolveu uma ferramenta chamada de Trimmomatic, com o propósito de melhorar a qualidade geral das sequências Illumina, realizando uma poda nas sequências. Por mais que a tecnologia Illumina resulte em fragmentos com boa qualidade, alguns fragmentos podem ser sequenciados erroneamente, sendo realizada então a poda destes fragmentos. Além disso, pedaços de fragmentos que tenham um número considerável de bases nitrogenadas com baixa qualidade também podem ser podados. O resultado da utilização desta ferramenta é um arquivo de sequenciamento Illumina com qualidade resultante superior.

A empresa Illumina possui uma ferramenta oficial para a montagem de fragmentos com a sua tecnologia, que é a SPAdes [Bankevich et al. 2012]. Essa ferramenta utiliza grafos De Brujin, conforme já mencionado, e tem recebido diversas atualizações desde sua criação, estando atualmente na versão 3.9.0 (2019). Contudo, essa ferramenta não é adequada para montagem de seres vivos com grandes quantidades de DNA (arquivos de entrada muito grandes), ou que possuem alta taxa de replicação em seu DNA. No primeiro caso, o custo computacional torna-se inviável, devido ao grande número de sequências existentes para montar o grafo. No segundo caso, a dificuldade está em distinguir regiões de ambiguidade no DNA, ou seja, duas ou mais regiões distintas porém possuindo sequências de bases nitrogenadas semelhantes. Aqui, como os fragmentos Illumina são pequenos, a distinção da ambiguidade é um empecilho para a montagem.

[Koren et al. 2017] desenvolveu o Canu, uma ferramenta com o propósito de mon-

tar sequências PacBio, realizando a correção do tipo auto. Seu *pipeline* é dividido em três etapas, sendo elas a correção, o corte e a montagem. Na etapa de correção ocorre o alinhamento das sequências PacBio presentes no arquivo de entrada, para a obtenção do consenso. Na etapa de corte, ocorre a retirada das sequências que são mais destoantes das demais. Por último, na etapa de montagem, ocorre a montagem propriamente dita. A ferramenta permite que cada uma das etapas possa ser executada individualmente, de forma que outras ferramentas possam ser acopladas junto a esta.

Smartdenovo [Ruan 2015] é uma ferramenta também desenvolvida para a montagem de sequências PacBio, porém esta não realiza a etapa da correção. Desta forma, é necessário que o arquivo contendo as sequências PacBio já esteja corrigido, caso contrário, o genoma resultante da montagem terá alta taxa de erro.

Em [Salmela and Rivals 2014] foi desenvolvido o LorDEC, que é uma ferramenta que realiza a correção de sequências PacBio. Esta ferramenta não realiza a montagem por janela deslizante, diferentemente do trabalho aqui proposto, devido ao alto custo computacional. Ao invés disso, ela realiza a montagem de um grafo utilizando as sequências com boa qualidade e compara então os fragmentos PacBio com este grafo. Sua correção não é perfeita, pois as comparações são realizadas apenas com o grafo montado a partir das sequências pequenas, e não com as sequências em si; porém, isso viabiliza seu custo computacional.

Ressalta-se aqui que os trabalhos comparativos entre diferentes métodos de correção/montagem de genoma encontrados na literatura possuem foco na comparação dos tempos globais (correção + montagem) e a qualidade do genoma resultante, como por exemplo o trabalho de [Khan et al. 2018], ou então os próprios artigos lançados juntamente com as ferramentas já descritas. Neste caso predomina-se principalmente o fator biológico (amostra, método de clonagem, componentes utilizados) como por exemplo em [Grohme et al. 2018]. Um outro aspecto também explorado na literatura com maior enfoque computacional é a comparação de diferentes técnicas de alinhamento, como por exemplo os trabalhos de [Purbarani et al. 2016] e [Aluru and Jammula 2014]. Observou-se então uma carência na literatura de estudos voltados para uma análise computacional de técnicas de correção híbrida de genoma.

3. Descrição do Problema

O algoritmo para a correção híbrida consiste em se trabalhar com dois arquivos de sequências genômicas de tecnologias diferentes, sendo um deles Illumina e o outro PacBio. Os arquivos são compostos por uma série de sequências de DNA, onde cada sequência é representada por 4 linhas. A primeira linha indica o cabeçalho da sequência, sendo iniciada pelo caractere "@", enquanto a segunda linha é a própria sequência de bases nitrogenadas. A terceira linha indica o cabeçalho da qualidade, utilizando o caractere "+"e a quarta linha são as qualidades de cada base. A seguir é apresentado um exemplo de arquivo.

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

Conforme observa-se no exemplo, cada base nitrogenada possui um identificador de qualidade (*Q-score*), representado por um caractere ASCII. A função do identificador é indicar qual a probabilidade de determinada base nitrogenada ter sido sequenciada corretamente. Diferentes codificações podem ser utilizadas em um arquivo de sequências genômicas, refletindo diferentes caracteres ASCII adotados [Cock et al. 2009], por isso, é necessário inicialmente identificar o tipo de codificação utilizada.

O arquivo de sequências grandes deve ser percorrido de forma que cada sequência grande seja lida e comparada com todas as sequências pequenas, visando realizar a correção. A comparação ocorre através do alinhamento por janela deslizante, onde cada alinhamento pontua uma medida de similaridade. O alinhamento considera que duas bases iguais pontuam 1 enquanto que duas bases distintas pontuam 0. Um *threshold* então é estabelecido, de forma que caso a similaridade obtida no alinhamento seja superior ao *threshold*, a correção é realizada, substituindo aquela região previamente alinhada da sequência grande pela sequência pequena. A Figura 1 ilustra um exemplo de alinhamento.

4. Estudo Experimental

O ambiente utilizado no estudo 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. Foi utilizado o compilador gcc e o mpicc, ambos com versão 4.8.3 e flag de otimização -O3 para gerar os executáveis.

Foram desenvolvidas 7 versões para o algoritmo de correção híbrida, visando comparar o desempenho de cada uma delas. São elas:

a) Algoritmo sequencial
b) Algoritmo *multicore*
c) Algoritmo multitarefa
d) Algoritmo distribuído
e) Algoritmo distribuído multitarefa
f) Algoritmo distribuído *multicore*
g) Algoritmo distribuído *multicore* com *broadcast*

A seguir será descrita cada versão.

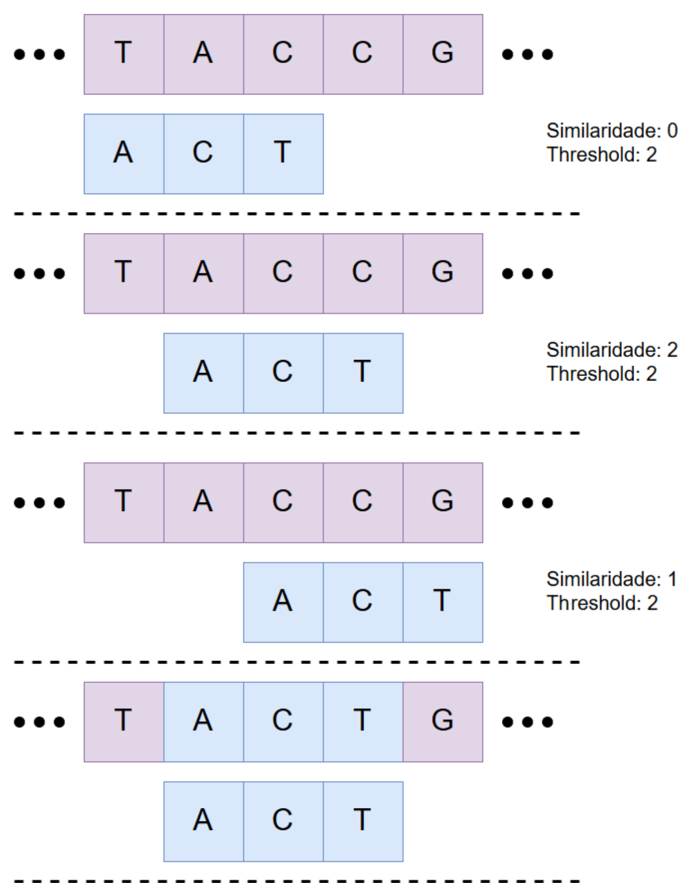

Figura 1. Exemplo de alinhamento

4.1. Algoritmo Sequencial

O algoritmo sequencial foi implementado em C, visando sua execução em um único *core* de um nó do *cluster*, com a criação de apenas uma tarefa computacional.

Inicialmente uma sequência grande é lida do arquivo de sequências grandes. Então, é lida uma sequência pequena do arquivo de sequências pequenas, ocorrendo a seguir o seu deslizamento pela sequência grande. O maior valor de similaridade é armazenado juntamente com o *offset* de deslizamento, de forma que, após o término do deslizamento, caso o valor da maior similaridade seja superior ao *threshold* estabelecido, a correção ocorre. Este processo ocorre para todas as sequências pequenas do arquivo. Após o término das sequências pequenas, a sequência grande corrigida é escrita no arquivo de saída. Uma nova sequência grande então é lida, e isso se repete até o término das sequências grandes.

Para melhorar o desempenho do algoritmo, uma lógica adicional foi projetada com o intuito de se reduzir o tempo total gasto com as sequências pequenas. Ela consiste em se comparar o valor máximo de similaridade obtida com o deslizamento de determinada sequência pequena em uma grande com o *threshold* necessário para a correção. Caso o valor de similaridade seja menor que o *threshold* é possível fazer uma dedução se a próxima sequência pequena terá a possibilidade de corrigir a sequência grande. Esta dedução baseia-se na diferença entre a sequência pequena já deslizada e a próxima sequência do arquivo. O cálculo feito então é que a diferença de caracteres entre a sequência pequena

atual e a posterior deve ser maior que a diferença entre a similaridade obtida e o *threshold*; caso contrário, a próxima sequência não conseguirá corrigir a sequência grande, sendo possível então já descartá-la. Esse processo pode ser repetido para um número qualquer de sequências pequenas do arquivo, desde que o comprimento da sequência atual e da posterior seja o mesmo.

Algorithm 1 Algoritmo Sequencial

```
Carrega arquivo de sequências pequenas
Carrega arquivo de sequências grandes
Abre arquivo de saída
while Restam sequências grandes para serem corrigidas do
    Lê uma sequência grande do arquivo
    while Restam sequências pequenas para corrigir do
        offset ← 0.
        similaridademax ← 0.
        i ← 0.
        Lê uma sequência pequena do arquivo
        while Deslizamento de sequencia pequena na sequência grande do
            Compara localmente sequência grande com sequência pequena para obter
pontuação
            similaridade ← pontuação.
            if similaridade > similaridademax then
                similaridademax ← similaridade.
                offset ← i.
        if similaridademax > threshold then
            Corrige sequência grande com pequena
        else
            while Sequência pequena não corrige sequência grande do
                Lê nova sequência pequena
                if Comprimento da sequência pequena atual é diferente da nova then
                    break
                caracteredif ← 0.
                while Comprimento da sequência do
                    if Caractere da sequência pequena atual é diferente da nova then
                        caracteredif ← caracteredif + 1.
                if caracteredif + similaridademax >= threshold then
                    break;
    Escreve sequência grande corrigida no arquivo de saída
```

4.2. Algoritmo *Multicore*

A versão *multicore* do algoritmo foi implementado em C com o OpenMP, com o propósito de ser executada em um nó do *cluster* utilizando todos os seus 4 *cores*, criando assim 4 tarefas computacionais.

Ela trabalha com a distribuição das sequências grandes para as *threads*. Aqui ocorre um balanceamento de carga, onde ao invés de se dividir o arquivo das sequências

grandes pelo número de *threads*, o arquivo possui um ponteiro público, e cada *thread* pode, após corrigir uma sequência grande, ler uma nova. Seções críticas foram criadas nos momentos de leitura do arquivo de sequências grandes e também de escrita do arquivo de saída, para evitar situações de corrida crítica. No arquivo das sequências pequenas foram utilizados ponteiros privados para cada *thread*, de forma que o processo de correção de uma sequência por uma *thread* seja independente dos demais.

4.3. Algoritmo Multitarefa

O algoritmo multitarefa foi implementado em C com o MPI para ser executado apenas em 1 nó do *cluster*, onde são criadas 4 tarefas computacionais MPI (ranks) sendo então realizado um paralelismo de processos no nó. Inicialmente, a comunicação se faz através da distribuição do arquivo das sequências pequenas para todos os ranks. Logo após, o rank 0 envia uma sequência grande para os outros ranks, sendo eles os responsáveis pelo processo de correção. Quando um rank finaliza o processo de correção, ele envia a sequência grande corrigida para o rank 0, que envia de volta uma nova sequência grande. Este processo se repete até o momento em que todas as sequências grandes foram enviadas pelo rank 0. Nesse momento ele envia uma mensagem com uma *tag* de término, para receber a última sequência grande de cada nó e finalizar o processo de correção.

4.4. Algoritmo Distribuído

O algoritmo distribuído assemelha-se ao multitarefa, porém este ao invés de criar as tarefas em um mesmo nó, ele as distribui pelo *cluster*, de forma que cada nó executa apenas 1 tarefa. Desta forma, são utilizados os 5 nós do *cluster*, criando 5 tarefas computacionais MPI.

4.5. Algoritmo Distribuído Multitarefa

A versão distribuída multitarefa utiliza todos os recursos disponíveis no *cluster*, ou seja, os 5 nós com 4 *cores*. Optou-se pela criação de 21 tarefas computacionais MPI ao invés de 20, pois a tarefa responsável pela comunicação com as demais fica suspensa durante uma parcela considerável do tempo de execução, devido à troca de mensagens.

Para não ocorrer a distribuição redundante do arquivo de sequências pequenas em cada nó, uma área de memória compartilhada é criada no primeiro processo do nó. Os demais processos de cada nó recebem o identificador da área de memória, de forma que após o arquivo de sequências grandes ser enviado para os primeiros processos, os outros têm acesso, evitando assim a replicação desnecessária do arquivo.

O processamento então ocorre de maneira semelhante à versão anterior do algoritmo, onde o processo de rank 0 é responsável por enviar as sequências grandes para os outros processos e receber as sequências corrigidas.

4.6. Algoritmo Distribuído *Multicore*

A versão distribuída *multicore* utiliza o MPI juntamente com o OpenMP, criando assim tarefas MPI e *threads* dentro das tarefas. Nela, o nó mestre (rank 0) possui uma *thread* alocada especificamente para a comunicação com os outros nós da rede, enquanto as outras *threads* também realizam a correção.

O nó mestre possui uma seção crítica, responsável por controlar a leitura do arquivo das sequências grandes, que pode ser acessada pela *thread* responsável pela comunicação ou pelas *threads* responsáveis pela correção. Os outros nós também possuem seções críticas, onde cada *thread* pode enviar ao nó mestre uma sequência corrigida e solicitar uma nova sequência grande.

4.7. Algoritmo Distribuído *Multicore* com *broadcast*

A distinção desta versão para a versão anterior está na utilização da comunicação tipo *broadcast* para o envio do arquivo das sequências pequenas. O objetivo aqui foi observar o ganho de desempenho da comunicação *broadcast* quando comparada com a comunicação *unicast*.

5. Resultados e Análise

Para a realização dos testes foram utilizados arquivos contendo sequências genômicas da bactéria *Escherichia coli*. O arquivo contendo as sequências Illumina foi extraído do banco de dados do *National Center for Biotechnology Information* (NCBI), e possui identificador ERR022075 [1]. O arquivo contendo as sequências PacBio foi extraído do repositório oficial da empresa [2].

Um pre-processamento dos dados foi realizado em ambos os arquivos, visando melhorar o processo da correção.

No arquivo Illumina inicialmente foi utilizada a ferramenta Trimmomatic com configuração padrão para melhorar a qualidade das sequências; em seguida foram retiradas as sequências repetidas, consequentes da quebra aleatória de DNA. Também foi retirada toda a informação presente no arquivo que não correspondesse às bases nitrogenadas, ou seja, considerando o modelo de 4 linhas no arquivo para cada sequência, apenas a segunda linha foi mantida. Por último, reduziu-se a cobertura do arquivo, através da truncagem, sendo a cobertura original por volta de 1000x.

No arquivo PacBio foi realizada uma substituição das bases nitrogenadas, visando remover a necessidade das qualidades deste arquivo na correção. Foi estabelecido um *threshold* de 5 para o *Q-score*, de forma que, caso determinada base possua um *Q-score* inferior a 5, ela é substituída pela letra N, que necessariamente pontua 1 sempre para o alinhamento com a sequência pequena. Desta forma, o arquivo utilizado na correção possui apenas duas linhas para cada sequência de bases nitrogenadas, sendo elas o cabeçalho e a própria sequência.

5.1. Teste de Versão

Foram realizados testes para cada uma das versões desenvolvidas, tendo o arquivo Illumina tamanho de 10MB e o arquivo PacBio tamanho de 1MB. A Tabela 1 apresenta o desempenho através da média de 10 tempos de cada versão, enquanto a Figura 2 ilustra graficamente os resultados.

[1]https://www.ncbi.nlm.nih.gov/sra/?term=ERR022075
[2]https://github.com/PacificBiosciences/DevNet/wiki/E-coli-K12-MG1655-Hybrid-Assembly

Tabela 1. Resultados obtidos com as diferentes versões

Versão	Tempo de Execução (s)	Desvio Padrão (s)
sequencial	27532,4	24,7
multicore	7082	3,9
multitarefa	7730	16,1
distribuído	6697	14,3
distribuído multitarefa	1620,8	25,5
distribuído *multicore* (5 processos)	1665	39,1
distribuído *multicore* (6 processos)	1587,8	3
distribuído *multicore* com *broadcast*	1585,8	3,8

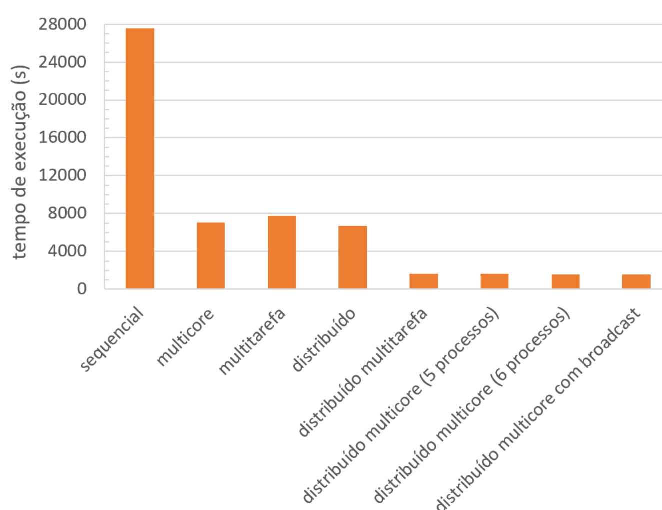

Figura 2. Tempos de execução obtidos no teste de versão do algoritmo

Comparando a versão sequencial com a versão *multicore* observa-se que o *speedup*[3] obtido através da paralelização do código foi de aproximadamente 3,88. A diferença entre o *speedup* obtido e o número de *cores* da máquina é justificado pela necessidade de se utilizar seções críticas no código, de maneira que apenas uma *thread* lê e escreve de cada vez.

Comparando a versão multitarefa com a versão *multicore*, observa-se que mesmo ambas utilizando todos os recursos disponíveis na máquina, a versão *multicore* possui desempenho superior à versão multitarefa. Isso pode ser justificado pelo fato que o *overhead* do MPI para gerar todos os processos na mesma máquina, além do custo da troca de mensagens, ser superior ao *overhead* para gerar as *threads* no OpenMP.

A comparação entre as versões multitarefa e distribuída revelam que a versão distribuída possui desempenho superior. Uma justificativa para isso está em uma melhor utilização da memória e dos caches do processador, já que na versão distribuída a carga é dividida entre os nós do *cluster*.

[3] Aqui o *speedup* é tratado como o ganho de desempenho da versão paralela em relação a versão sequencial do algoritmo

O *speedup* da versão distribuída multitarefa quando comparada com a versão sequencial foi de 17 enquanto o *speedup* em relação à versão *multicore* foi de 4,4. A justificativa aqui está novamente no *overhead* na criação dos múltiplos processos e no custo de troca de mensagens. Deve-se ressaltar que como apenas 1 processo é responsável pela troca de mensagens, existe uma fila dos processos restantes para enviar/receber sequências grandes, reduzindo assim o desempenho do algoritmo.

A versão distribuída *multicore* foi executada utilizando 5 processos, onde cada processo possui 4 *threads* e com 6 processos, onde 5 processos possuem 4 *theads* e 1 processo é sequencial e responsável pela troca de mensagens. A versão com 5 processos possui desempenho inferior quando comparado à versão com 6 processos, pois a *thread* responsável pela troca de mensagens fica suspensa na maior parte do tempo, esperando para enviar/receber novas sequências grandes. Desta forma, acaba-se utilizando 19 cores, ao invés de 20. Na versão com 6 processos isto é contornado, resultando em um aumento do desempenho.

Comparando a versão distribuída multitarefa que se utiliza apenas do MPI com a versão distribuída *multicore* que utiliza MPI + OpenMP, a segunda possui desempenho superior no caso com 6 processos. Isso é fundado no fato de que um número menor de mensagens do MPI é necessário na versão distribuída *multicore*, pois são utilizadas apenas 5 tarefas MPI frente às 21 utilizadas na versão distribuída multitarefa.

Por último, a comparação entre a versão distribuída *multicore* com e sem *broadcast* indicam baixa contribuição no desempenho ao se trocar o método de distribuição das sequências pequenas. A justificativa para isso está no fato que o *cluster* possui apenas 5 máquinas, então a utilização do *broadcast* para enviar as sequências pequenas economizou poucas mensagens.

A Figura 3 ilustra graficamente o *speedup* das versões quando comparadas com a versão sequencial.

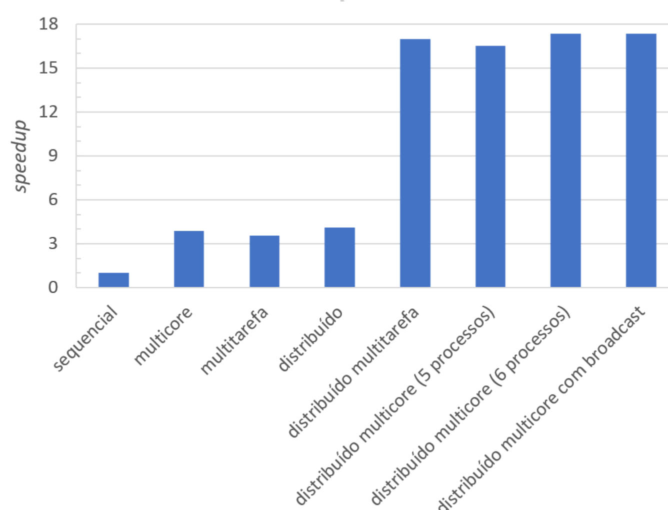

Figura 3. *Speedups* obtidos nos testes de versão do algoritmo

5.2. Teste de Escalabilidade

Após obter-se a melhor versão, um teste de escalabilidade foi realizado, com o intuito de se observar a escalabilidade do algoritmo proposto. Para tal, o tamanho do problema em questão foi variado, ou seja, o algoritmo foi executado em tamanhos distintos de dados, variando-se o tamanho do arquivo das sequências grandes e das sequências pequenas.

Variou-se o tamanho do arquivo de sequências grandes de 1MB até 5MB, utilizando-se passos de 1MB. O arquivo de sequências pequenas foi variado de 10MB até 50MB, utilizando-se passos de 10MB. Os gráficos apresentados nas Figuras 4 e 5 ilustram os resultados obtidos.

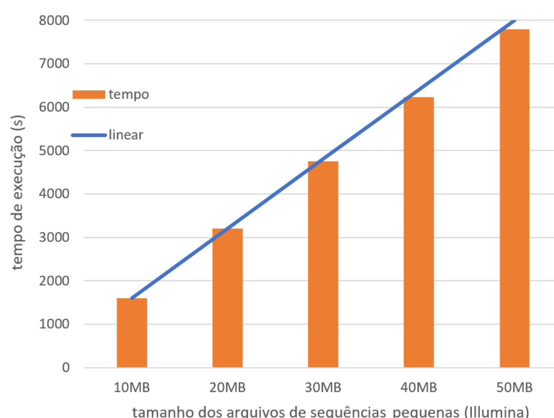

Figura 4. Teste de escalabilidade para variação no tamanho do arquivo de sequências pequenas

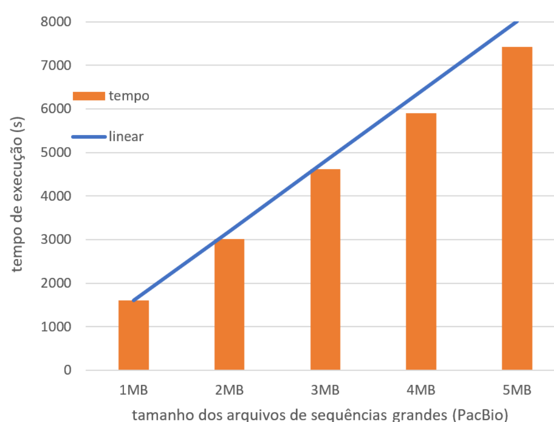

Figura 5. Teste de escalabilidade para variação no tamanho do arquivo de sequências grandes

Os resultados do teste de escalabilidade para a variação do tamanho do arquivo de sequências pequenas apontam que a escalabilidade obtida é superior à linear. A justificativa para isto está na fila de espera das *threads* para realizar a comunicação com o processo de rank 0, que aumenta o tempo total de correção de uma sequência grande. Com um maior número de sequências pequenas, utilizando-se a lógica adicional descrita na versão sequencial do algoritmo, um maior número de sequências pequenas pode ser

descartada em determinados casos, resultando em uma maior amplitude de tempos para a correção. Esta amplitude de valores gera uma dessincronização entre as *threads*, fazendo com que elas requisitem a comunicação com o processo de rank 0 em momentos distintos, diminuindo assim o tempo de espera na fila.

Os resultados do teste de escalabilidade para a variação do arquivo de sequências grandes também apontam uma escalabilidade superior à linear. Novamente a justificativa está na fila de espera, porém o motivo aqui é que com um número maior de sequências grandes, a dessincronização entre as *threads* ocorre naturalmente.

6. Conclusão e Trabalhos Futuros

O artigo teve como objetivo explorar uma área com carência na literatura que é a análise de algoritmos de correção híbrida. Foi realizada aqui uma análise de desempenho em um algoritmo desenvolvido com 7 versões distintas, onde observou-se que a versão utilizando MPI com OpenMP e comunicação do tipo *broadcast* apresentou melhor desempenho. Além disso, esta versão possui boa escalabilidade, obtendo resultados superiores à escalabilidade linear em relação ao número de sequências e apontando como gargalo do algoritmo a fila de comunicação com o nó principal.

O grande motivador deste trabalho foi a possibilidade da implementação posterior em *hardware* para o algoritmo de correção de sequências genômicas, visando melhorar o desempenho aqui obtido. Desta forma, tendo sido estabelecida a melhor versão em *software*, define-se o trabalho futuro como sua implementação em *hardware* utilizando-se uma Field Programmable Gate Array (FPGA) juntamente com uma linguagem de descrição de *hardware*.

Referências

Aluru, S. and Jammula, N. (2014). A review of hardware acceleration for computational genomics. *IEEE Design Test*, 31(1):19–30.

Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., et al. (2012). Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477.

Bolger, A. M., Lohse, M., and Usadel, B. (2014). Trimmomatic: a flexible trimmer for illumina sequence data. *Bioinformatics*, 30(15):2114–2120.

Cock, P. J., Fields, C. J., Goto, N., Heuer, M. L., and Rice, P. M. (2009). The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants. *Nucleic acids research*, 38(6):1767–1771.

Compeau, P., Pevzner, P., and Tesler, G. (2011). How to apply de bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987–991.

Consortium, . G. P. et al. (2010). A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061.

Consortium, . G. P. et al. (2015). A global reference for human genetic variation. *Nature*, 526(7571):68.

Del Angel, V. D., Hjerde, E., Sterck, L., Capella-Gutierrez, S., Notredame, C., Pettersson, O. V., Amselem, J., Bouri, L., Bocs, S., Klopp, C., et al. (2018). Ten steps to get started in genome assembly and annotation. *F1000Research*, 7.

Goodwin, S., McPherson, J. D., and McCombie, W. R. (2016). Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333.

Grohme, M. A., Schloissnig, S., Rozanski, A., Pippel, M., Young, G. R., Winkler, S., Brandl, H., Henry, I., Dahl, A., Powell, S., et al. (2018). The genome of schmidtea mediterranea and the evolution of core cellular mechanisms. *Nature*, 554(7690):56.

Illumina (2010). De novo assembly using illumina reads.

Khan, A. R., Pervez, M. T., Babar, M. E., Naveed, N., and Shoaib, M. (2018). A comprehensive study of de novo genome assemblers: current challenges and future prospective. *Evolutionary Bioinformatics*, 14:1176934318758650.

Koren, S., Walenz, B. P., Berlin, K., Miller, J. R., Bergman, N. H., and Phillippy, A. M. (2017). Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research*, 27(5):722–736.

Purbarani, S. C., Sanabila, H. R., Bowolaksono, A., and Wiweko, B. (2016). A survey of whole genome alignment tools and frameworks based on hadoop's mapreduce. In *2016 International Workshop on Big Data and Information Security (IWBIS)*, pages 65–70.

Rhoads, A. and Au, K. F. (2015). Pacbio sequencing and its applications. *Genomics, Proteomics Bioinformatics*, 13(5):278 – 289. SI: Metagenomics of Marine Environments.

Ruan, J. (2015). Smartdenovo. `https://github.com/ruanjue/smartdenovo`. Acesso: 02/03/2019.

Salmela, L. and Rivals, E. (2014). Lordec: accurate and efficient long read error correction. *Bioinformatics*, 30(24):3506–3514.

Wheeler, D. A., Srinivasan, M., Egholm, M., Shen, Y., Chen, L., McGuire, A., He, W., Chen, Y.-J., Makhijani, V., Roth, G. T., et al. (2008). The complete genome of an individual by massively parallel dna sequencing. *Nature*, 452(7189):872.