Desenvolvimento do Montador Velvet Usando OpenACC

Evaldo B. Costa[1], **Gabriel P. Silva**[1]

[1]Instituto de Computação – Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro – RJ – Brazil

{ebcosta,gabriel}@ic.ufrj.br

Resumo. *Em bioinformática, existem vários programas disponíveis para análise de sequências de DNA. Esta é geralmente uma tarefa muito demorada, uma vez que essas sequências de DNA podem ser muito longas e complexas. O montador Velvet foi projetado para montar dados de sequenciamento de leitura curta e longa em sequências genômicas mais longas. A última versão do Velvet foi desenvolvida para funcionar com várias threads usando programação paralela com OpenMP. Aqui apresentamos uma nova versão do Velvet que explora multiprocessamento e unidades de processamento gráfico (GPU) por meio de diretivas OpenACC. Nossos testes demonstram que essa extensão do Velvet permite um desempenho mais rápido e uso de memória mais eficiente.*

1. Introdução

Um dos principais desafios associados à montagem de sequências de DNA é a quantidade de tempo e recursos computacionais necessários para esse processamento. Avanços recentes em sistemas de computação resultaram em mais poder de processamento, maior capacidade de memória e armazenamento de dados, de modo que os programas de montagem precisam ser usados de maneira mais eficiente [Costa et al. 2015].

A montagem de sequências de DNA é uma tarefa altamente computacional, que geralmente requer o uso de programas e algoritmos paralelos, para que possa ser realizada com a precisão desejável e dentro de prazos adequados. Aqui apresentamos uma nova versão do Velvet que explora multiprocessamento e unidades de processamento gráfico (GPU) por meio de diretivas OpenACC.

O montador Velvet *de novo* [Zerbino and Birney 2008] é usado para construir longas sequências contínuas, ou *contigs*, bem como montagens com *gap* de *contigs*, ou *scaffolds*, a partir de conjuntos de dados de sequenciamento genômico de leitura curta, normalmente obtidos com sequenciadores de DNA chamados de próxima geração (*Next-Generation DNA sequecing*). O montador é composto por um conjunto de algoritmos que armazenam dados de sequenciamento genômico em grafos *de Bruijn* para eliminar erros de forma eficiente e montá-los em sequências mais longas. Essas duas tarefas são executadas por dois programas executáveis, velveth e velvetg: primeiro, o algoritmo de *hash* do velveth mescla sequências que pertencem umas às outras, depois o montador do velvetg constrói um grafo, resolve repetições ambíguas e separa caminhos compartilhando sobreposições locais. O Velvet foi desenvolvido em linguagem "C" para funcionar em um ambiente Linux de 64 bits.

A última versão do Velvet 1.2.0 foi desenvolvida para trabalhar em múltiplas *threads* usando programação paralela com OpenMP. Neste trabalho apresentamos uma nova

versão do Velvet usando as diretivas OpenACC, um paradigma de programação paralela desenvolvido com o objetivo de simplificar a programação paralela e oferecer alto desempenho e portabilidade entre diferentes tipos de arquiteturas: *multicore*, *manycore* e GPUs [Chen 2017].

O OpenACC permite que os programadores usem simples diretivas de compilador para identificar quais áreas de código devem ser aceleradas, sem exigir modificação no próprio código subjacente. Ao identificar segmentos de código paralelos, as diretivas OpenACC permitem que o compilador faça o trabalho de mapear a computação no acelerador [Costa and Silva 2019] [Larkin 2018].

2. Trabalhos Relacionados

Com a crescente quantidade de dados gerados pelo sequenciamento de DNA, foi necessário desenvolver programas para realizar com mais rapidez e eficência a montagem dessas sequências. Atualmente, existe um grande número de montadores que são utilizados para tornar o processo de montagem mais simples e rápido para os usuários.

Esses montadores possuem abordagens distintas tanto no uso das estratégias quanto na linguagem de programação e no paradigma de programação. As estratégias usadas são: *Overlap-Layout-Consensus* (OLC) e o grafo *de Bruijn*. O uso da abordagem OLC consiste em produzir alinhamentos entre as leituras e identificar sobreposições, agrupando-as em *contigs*, e então produzir uma sequência de consenso. Já na estratégia de uso do grafo *de Bruijn*, as leituras são fragmentadas em sequências menores de tamanho fixo k (ou *k-mers*) denominadas *seeds* (Tabela 1).

Tabela 1. Comparação de montadores de sequências em bioinformática

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

Para definir qual montador de sequenciamento de DNA utilizado neste trabalho, foi realizada uma análise dos estudos realizados que compararam e avaliaram montadores de genoma *de novo*. Esses estudos avaliaram os critérios de utilização de recursos computacionais, tempo de montagem e qualidade dos resultados obtidos.

A avaliação dos montadores de sequência *de novo*: ABySS, Velvet, Edena, SGA, Ray, SSAKE e Perga mostrou que, apesar de todos serem capazes de processar genomas inteiros procarióticos ou eucarióticos, apenas os montadores Velvet e ABySS mostraram boa eficiência em termos de tempo de montagem e uso de recursos computacionais (Figura 1). Apesar de resultados semelhantes entre os montadores ABySS e Velvet, este último também possui alta escalabilidade para lidar com uma grande quantidade de dados em comparação com outros montadores [Khan et al. 2018].

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

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

Figura 1. Montadores avaliados quanto ao desempenho e confiabilidade na montagem de genomas. Observa-se o percentual de bases dos contigs montados e mapeados ao respectivo genoma de referência [Khan et al. 2018]

Outro importante estudo realizado foi o GAGE (Gnome Assembly Gold-standard Evaluations). Neste estudo, foi realizada uma avaliação de alguns montadores de genoma *de novo* como ABySS, ALLPATHS-LG, Bambus2, CABOG, MSR-CA, SGA, SOAPdenovo e Velvet. Alguns montadores, entre eles o Velvet, obtiveram os melhores resultados após a montagem dos genomas de referência [Salzberg et al. 2012]. Apesar do Velvet estar novamente entre os melhores montadores, foi verificado que ele tem um alto consumo de recursos computacionais, principalmente o uso de memória.

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

Aqui apresentamos uma nova versão do montador Velvet que faz uso de unidades de processamento gráfico (GPU) por meio de diretivas OpenACC. O desenvolvimento desta versão do montador tem por objetivo disponibilizar para a comunidade mais uma opção de ferramenta para o processamento de dados em bioinformática.

3. Implementação

Para o desenvolvimento da versão do montador em OpenACC foram usadas diretivas para paralelização de laços, assim como a movimentação de dados entre o servidor e a GPU. Com isto, todos os dados são movidos para a GPU onde é executado o processamento em sua memória local. Após o término do processamento, os dados são movidos de volta para o servidor.

Durante a execução do programa velveth, onde o algoritmo de *hash* mescla sequências, o dispositivo GPU teve baixa utilização. No velvetg, o dispositivo GPU foi mais utilizado, pois nesta etapa de construção de um grafo, repetições ambíguas são resolvidas e os caminhos compartilhando sobreposições locais são resolvidos.

O programa Velvet possui um código muito extenso. Modificamos apenas os trechos de código que consomem a maior quantidade de recursos computacionais. Para identificar essas regiões que consomem maiores quantidades de recursos computacionais, utilizamos o comando *pgprof* para criar o perfil de execução do Velvet visto na Figura 2.

Durante a execução do velveth, o processo chamado **inputSequenceIntroSplayTable**

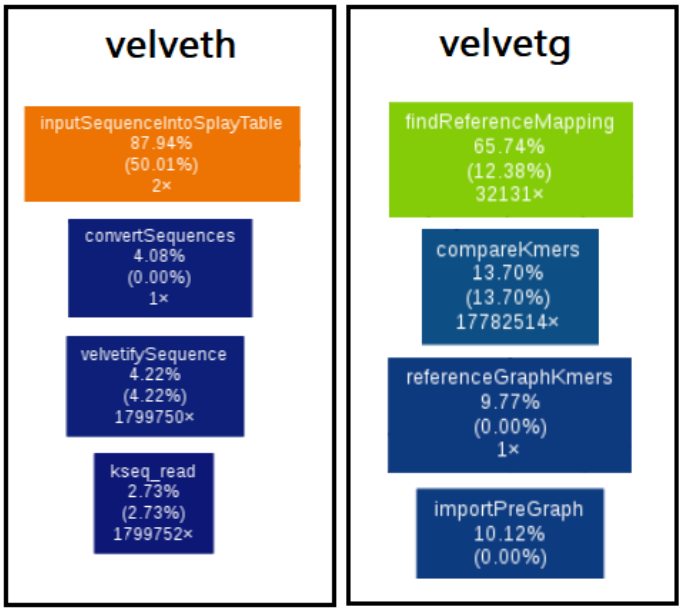

Figura 2. Resumo de utilização dos executaveis velveth e velvetg

consome aproximadamente 87% do tempo e dos recursos computacionais. Neste caso, alteramos apenas os trechos de código utilizados neste processo. No velvetg, o processo **findReferenceMapping** consome aproximadamente 65% do tempo de execução, sendo que da mesma forma foram alterados os trechos de código referentes a este processo. Essas mudanças foram feitas principalmente usando diretivas como parallel loop, **vector length** e **data copy**.

O montador Velvet é composto por vários arquivos que são usados para criar os executáveis velveth e velvetg. Existem alguns arquivos que usam o paradigma de programação OpenMP para executar o montador Velvet em paralelo. Alguns desses arquivos não tiveram um melhor desempenho usando OpenACC, por este motivo permaneceram utilizando OpenMP. Esses aquivos trabalham com rotinas que são exclusivas do paradigma de programação OpenMP. Os demais arquivos foram alterados para usar o paradigma de programação no OpenACC.

Para usar o OpenACC, foi utilizada a diretiva **parallel**. Com a diretiva **parallel**, é possível obter melhores resultados de desempenho com o uso de cláusulas, para obter a maior utilização possível através de parametrizações específicas do dispositivo GPU. A seguir está uma lista dos pragmas OpenACC que foram usados nesses arquivos.

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

Este é um exemplo de código compilado, que foi encontrado no arquivo scaffold.c.

Os executáveis velveth e velvetg foram compilados em um servidor com sistema operacional Linux de 64 bits, utilizando o compilador PGI e um dispositivo GPU NVIDIA. O

```
# pgcc -acc -ta=tesla -Minfo=acc,par -mp -fast -c src/scaffold.c -o obj/scaffold.o

countCoOccurences:
565, Generating copyout(coOccurencesCount[:5])
566, Generating Tesla code
567, #pragma acc loop gang, vector(256)
measureCoOccurences:
629, Generating copyout(coOccurencesIndex[:])
630, Generating Tesla code
631, #pragma acc loop gang, vector(256)
estimateLibraryInsertLength:
700, Generating copy(counter,variance,coOccurences[:coOccurencesCount])
701, Generating Tesla code
702, #pragma acc loop gang, vector(128)
Generating reduction(+:variance,counter)
```

código fonte da nova versão do Velvet em OpenACC está disponível no GitHub através do link: https://github.com/evaldocosta/velvetacc.

4. Ambiente de Testes

Os testes foram realizados em um servidor com dois processadores Intel Xeon E5-2609 (1,7 GHz, 8 núcleos cada, 20 MB de cache), com 128 GB de memória compartilhada e uma GPU NVIDIA Tesla K80. O NVIDIA Tesla K80 é uma unidade de GPU dupla que utiliza dois chipsets GK210B. Como unidade, esta placa oferece um total de 4992 núcleos CUDA com clock de 560 MHz acoplados a 24 GB de VRAM GDDR5 com interface de memória de 384 bits e largura de banda de 480 GB/s.

Todas as versões do Velvet foram compiladas usando o PGI Compiler 19.10 para fornecer o melhor desempenho. Os arquivos usados no experimento foram armazenados em discos locais SSD (Solid-State Drive) de alta velocidade. O sistema operacional utilizado foi a versão 7.8 da distribuição Centos Linux de 64 bits. Nas implementações do padrão OpenMP e OpenACC, foram utilizados os comandos abaixo para executar a montagem.

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

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

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

Para a execução dos ensaios, foram utilizados três tipos de dados. Todos os conjun-

tos de dados brutos podem ser baixados dos servidores do European Nucleotide Archive (ENA) e do National Center for Biotechnology Information (NCBI), as simulações foram executadas com os genomas: Staphylococcus aureus — NCBI SRA (SRR022868, SRR022865), Rhodobacter sphaeroides — NCBI SRA (SRR081522, SRR034528) e Homo sapiens (Cromossomo 21) — ENA (DRR000546). Um resumo pode ser visto na Tabela 2. O tempo total de montagem em segundos foi calculado usando o comando *time* do Linux para cada montagem. Os comandos do Linux *smem* e *mpstat* foram usados para medir o uso de memória e CPU, respectivamente.

Tabela 2. Tamanho dos dados utilizados

Espécies	Tamanho Genoma (Mbp)	Arquivos
Staphylococcus aureus	2.9	NCBI SRA (SRR022868, SRR022865)
Rhodobacter sphaeroides	4.6	NCBI SRA (SRR081522, SRR034528)
Homo sapiens (chr21)	46.7	ENA (DRR000546)

5. Resultados

Para os resultados apresentados neste estudo, foram executadas três séries de testes. Após cada execução, o tempo médio das séries foi calculado para definir o speedup e a eficiência obtida em cada caso.

5.1. Tempo de execução

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

Tabela 3. Comparação de ganho dos tempos de execução da versão OpenACC em relação à versão OpenMP

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

Pode ser observado na Tabela 3 o ganho total obtido no processamento dos genomas.

5.2. Utilização de memória

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

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

A Figura 3 e Figura 4 mostram o consumo de memória durante a execução dos processos velveth e velvetg nas versões em OpenMP e OpenACC.

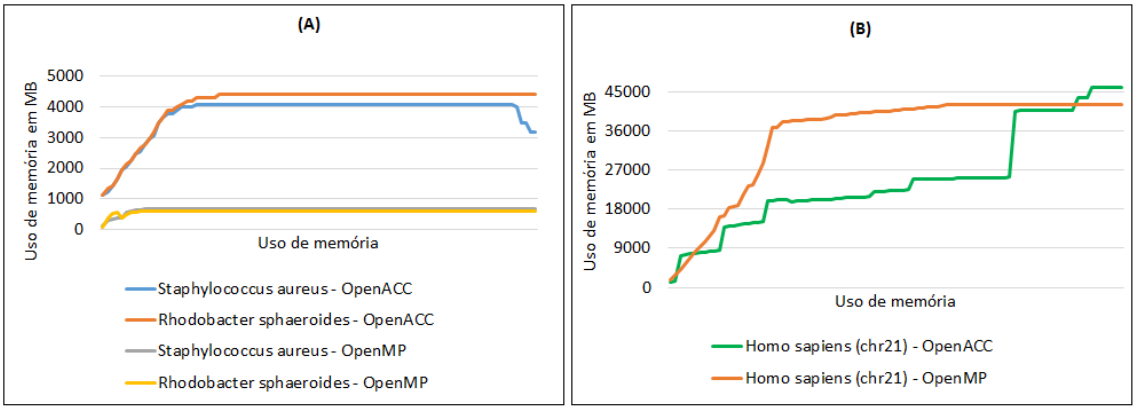

Figura 3. utilização de memória do velveth. (A) utilização de memória dos genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides*. (B) utilização de memória do genoma *Homo sapiens*

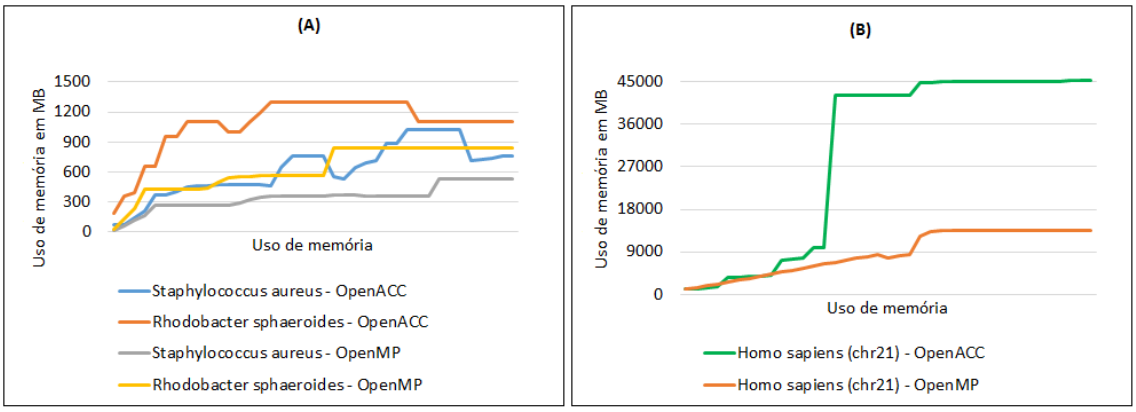

Figura 4. utilização de memória do velvetg. (A) utilização de memória dos genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides*. (B) utilização de memória do genoma *Homo sapiens*

5.3. Utilização de CPU

O uso de CPU na execução do processo velveth, na versão com OpenACC, teve um aumento em relação à versão em OpenMP, assim como também houve um aumento no uso de memória (Figura 5).

Porém quando executado o processo velvetg usando OpenACC, o consumo de CPU foi menor, principalmente usando o cromossomo 21 de *Homo sapiens*, como visto na Figura 6.

5.4. Utilização de GPU

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

- **Utilização dos núcleos de processamento**

Na Figura 7 são apresentados os resultados da média de utilização dos núcleos de processamento de GPU (*cuda cores*) durante a fase de montagem dos genomas.

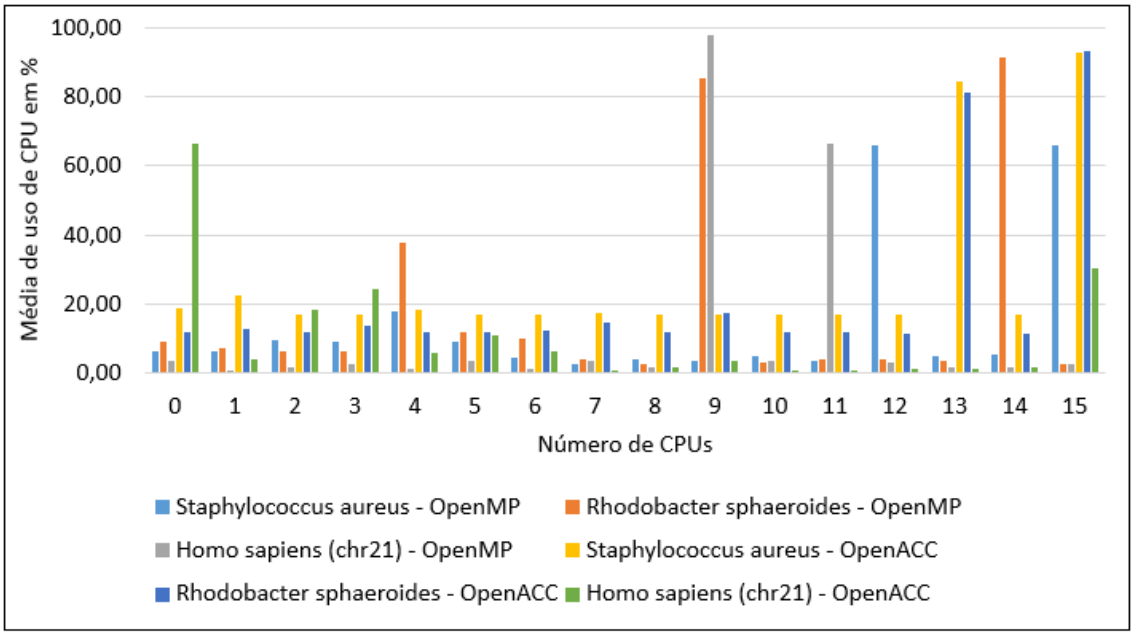

Figura 5. Média de utilização de CPU durante a leitura dos arquivos do genoma

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

- **Quantidade de memória utilizada**

Assim como na média de utilização dos núcleos da GPU, o mesmo comportamento pode ser visto em relação à utilização média de memória da GPU, conforme apresentado na Figura 8. Ou seja, genomas maiores fazem uso de mais recursos de memória da GPU, mas em nenhum caso foi consumido, em média, um total de memória maior do que os recursos disponíveis na GPU. Ainda, como no caso anterior, o consumo de memória também não é diretamente proporcional ao tamanho dos genomas.

- **Movimentação de dados**

Outro ponto avaliado durante o processamento dos genomas foi a movimentação de dados entre o servidor e a GPU, que ocorre em ambos os sentidos, do servidor para a GPU e no sentido oposto, da GPU para o servidor. Durante a execução do programa velveth não existe uma grande movimentação de dados entre o servidor e a GPU, porque este processo faz a leitura dos arquivos, quando a GPU é pouco usada.

Porém, a movimentação de dados na execução do programa velvetg é bem maior. Isso ocorre porque nesse processo é feita a construção do grafo *de Bruijn*, cuja construção é delegada para a GPU. Na Figura 9 são apresentados os resultados médios da movimentação de dados entre o servidor e a GPU, durante a execução do programa velvetg.

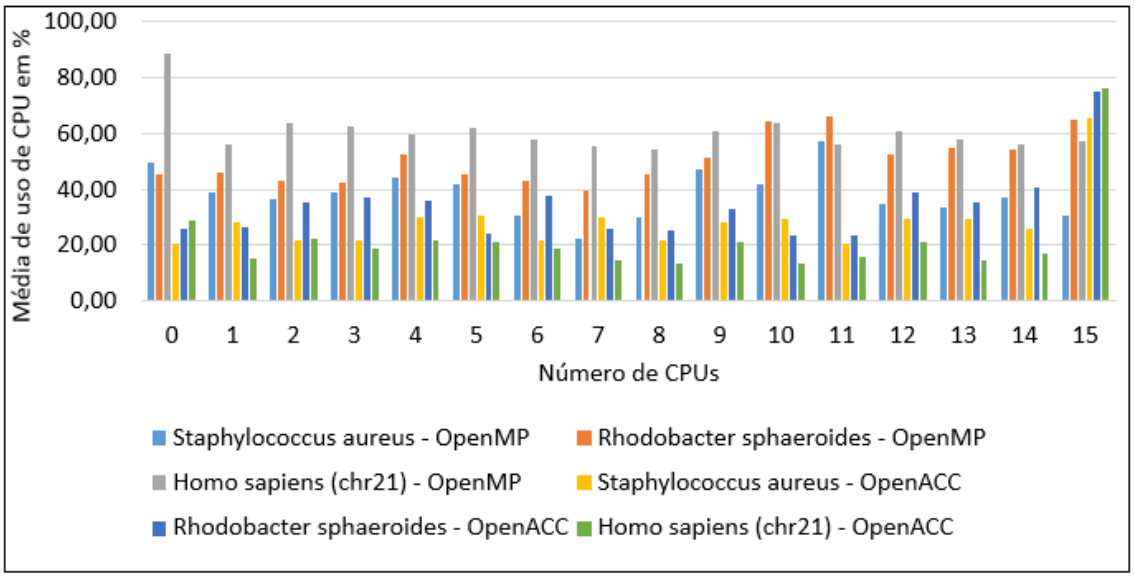

Figura 6. Média de utilização de CPU durante a montagem do genoma

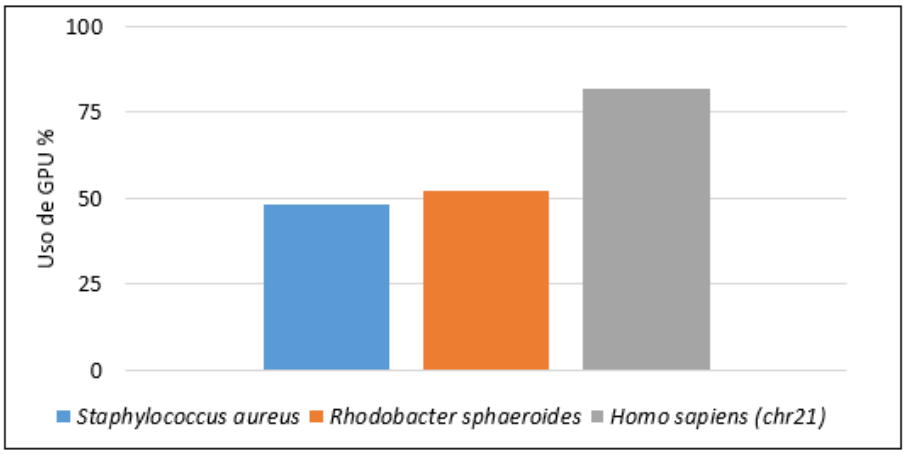

Figura 7. Média de utilização dos núcleos de processamento de GPU

Verifica-se que, na Figura 10, em que é apresentada a média de movimentação de dados entre a GPU e o servidor, a quantidade de dados é maior. Isto ocorre pelo processamento das informações que são executadas na GPU e depois enviadas para o servidor.

5.5. Qualidade da Montagem

Após a montagem dos genomas, a qualidade da montagem foi verificada usando o programa Quast [Mikheenko et al. 2018]. Como pode ser visto na Tabela 4, a qualidade da montagem tanto no OpenMP quanto no OpenACC foram próximas. O proposito do trabalho não é avaliar a qualidade da montagem, e sim melhorar o tempo de execução da montagem atraves do uso do OpenACC,mas queríamos assegurar que não houve perda de qualidade dos resultados da montagem.

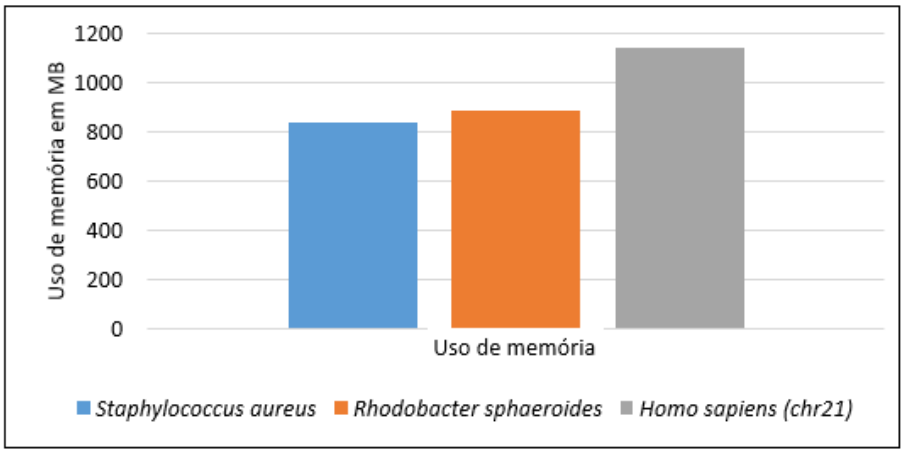

Figura 8. Média de utilização de memória da GPU

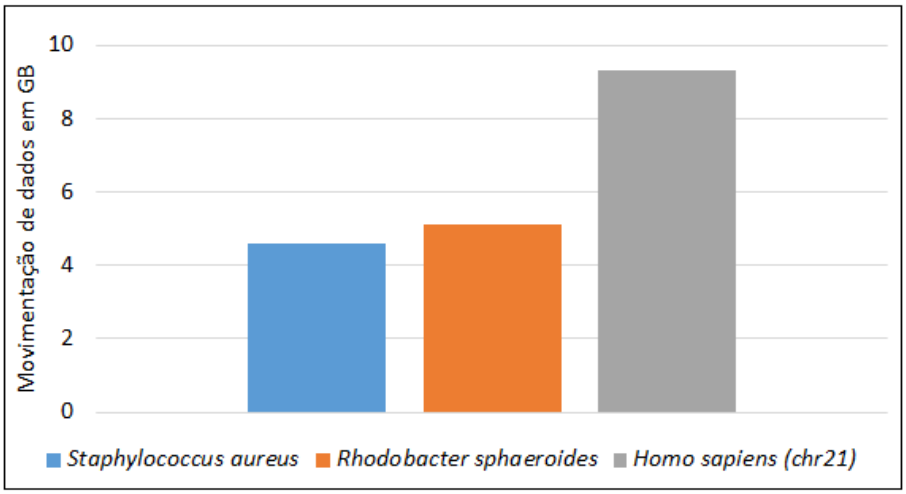

Figura 9. Média de movimentação de dados do servidor para a GPU durante a execução do programa velvetg

6. Conclusão

Uma comparação da versão anterior do Velvet, que usa o paradigma de programação OpenMP, com a nova versão em OpenACC, mostra que o ganho total durante todo o processo de montagem dos genomas chegou a ser de até 5 vezes mais rápido para os genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides*, e o cromossomo 21 de *Homo sapiens* alcançou um ganho médio de 3 vezes. O código fonte da nova versão do Velvet em OpenACC está disponível no GitHub através do link: https://github.com/evaldocosta/velvetacc.

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

A movimentação de dados entre o servidor e a GPU durante a montagem do genoma foi

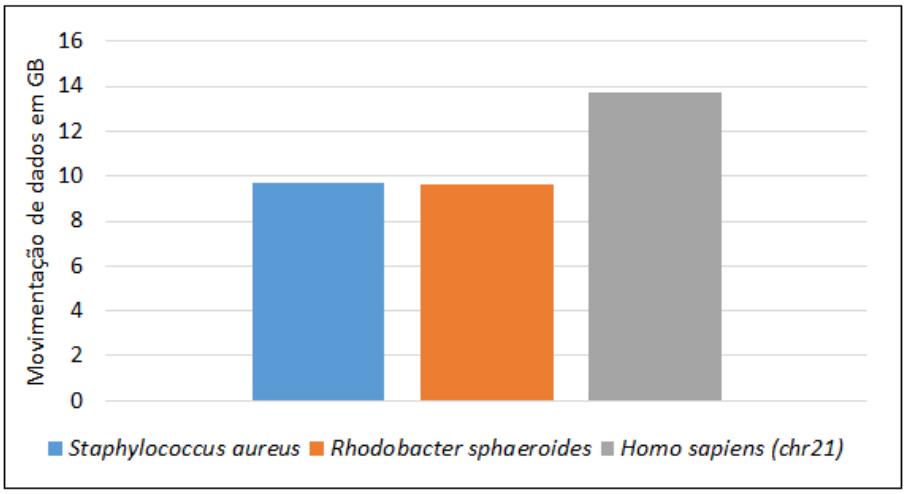

Figura 10. Média de movimentação de dados da GPU para o servidor

Tabela 4. Resultados da montagem do genoma Staphylococcus aureus usando o programa Quast

OpenMP Results		OpenACC Results	
Assembly	contigs	Assembly	contigs
# contigs (>= 0 bp)	713	# contigs (>= 0 bp)	710
# contigs (>= 1000 bp)	171	# contigs (>= 1000 bp)	170
# contigs (>= 5000 bp)	119	# contigs (>= 5000 bp)	118
# contigs (>= 10000 bp)	78	# contigs (>= 10000 bp)	77
# contigs (>= 25000 bp)	40	# contigs (>= 25000 bp)	41
# contigs (>= 50000 bp)	9	# contigs (>= 50000 bp)	9
Total length (>= 0 bp)	2863724	Total length (>= 0 bp)	2863603
Total length (>= 1000 bp)	2760566	Total length (>= 1000 bp)	2760567
Total length (>= 5000 bp)	2629876	Total length (>= 5000 bp)	2629877
Total length (>= 10000 bp)	2346511	Total length (>= 10000 bp)	2346512
Total length (>= 25000 bp)	1744129	Total length (>= 25000 bp)	1771272
Total length (>= 50000 bp)	651405	Total length (>= 50000 bp)	651405
# contigs	220	# contigs	219
Largest contig	95737	Largest contig	95737
Total length	2796223	Total length	2796224
GC (%)	32.57	GC (%)	32.57
N50	31818	N50	31818
N75	15853	N75	15853
L50	29	L50	29
L75	58	L75	58
# N's per 100 kbp	0.00	# N's per 100 kbp	0.00

maior na execução do programa velvetg, por conta da construção do grafo *de Bruijn* que este programa realiza. Na execução do velveth essa movimentação de dados é mínima, porque a tarefa de leitura dos arquivos é predominante neste programa. Em relação ao uso de memória e ocupação dos núcleos da GPU ambos programas apresentaram o mesmo

comportamento, ou seja, quanto maior o genoma, mais recursos são utilizados.

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

Agradecimentos

Os autores agradecem ao Instituto de Computação da Universidade da Federal do Rio de Janeiro por fornecer os recursos computacionais utilizados para a realização dos experimentos apresentados neste trabalho. Agradecemos também ao Dr. Daniel R. Zerbino por suas contribuições.

Referências

Chen, S. (2017). Introduction to openacc. *Research Computing Services Information Services and Technology Boston University*.

Costa, E. B. and Silva, G. P. (2019). Introdução à programação com openacc. In *WSCAD 2019 - Minicursos ()*.

Costa, E. B., Silva, G. P., and Teixeira, M. G. (2015). Performance evaluation of parallel genome assemblers. In *Proc. of the 7th Int. Conf. on Bioinformatics and Computational Biology (BICOB 2015)*, volume 1, pages 31–38.

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.

Larkin, J. (2018). Introduction to openacc. *online] http://ondemand. gputechconf. com/gtc/2015/presentation/S5192-Jeff-Larkin. pdf [accessed 15 August 2017]*.

Mikheenko, A., Prjibelski, A., Saveliev, V., Antipov, D., and Gurevich, A. (2018). Versatile genome assembly evaluation with quast-lg. *Bioinformatics*, 34(13):i142–i150.

Salzberg, S. L., Phillippy, A. M., Zimin, A., Puiu, D., Magoc, T., Koren, S., Treangen, T. J., Schatz, M. C., Delcher, A. L., Roberts, M., et al. (2012). Gage: A critical evaluation of genome assemblies and assembly algorithms. *Genome research*, 22(3):557–567.

Zerbino, D. R. and Birney, E. (2008). Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829.