EPiC Series in Computing

Volume 92, 2023, Pages 72–81

Proceedings of International Conference on Bioinformatics and Computational Biology (BICOB-2023)

The Velvet Assembler Using OpenACC Directives

Evaldo B. Costa[1] and Gabriel P. Silva[1]

Computer Institute, UFRJ, Rio de Janeiro, Brazil
{ebcosta,gabriel}@ic.ufrj.br

Abstract

There are several programs available in bioinformatics for DNA sequence assembly. This is typically an extremely time-consuming endeavor, as DNA sequences can be extensive and intricate. Velvet was created to combine short and long read sequencing data into larger genomic sequences. Using OpenMP parallel programming, the last version of Velvet was created to support multiple threads. Through OpenACC directives, we present a new version of Velvet that takes advantage of multiprocessing using graphical processing units (GPU). Our tests demonstrate that this extension of Velvet allows for faster performance and efficient memory use.

1 Introduction

The amount of time and computational resources necessary for this procedure is one of the primary obstacles associated with DNA sequence assembly. Recent advancements in computing systems have led to an increase in processing power, memory, and data storage capacity, necessitating a more efficient application of assembly programs [2].

This is a highly computational task that typically necessitates the use of parallel programs and algorithms in order to be completed with the desired precision and within acceptable time constraints. Through OpenACC directives, we present a new version of Velvet that takes advantage of multiprocessing and graphical processing units (GPU).

The Velvet de novo assembler [7] is used to construct large continuous sequences, or contigs, and gapped assemblies of contigs, or scaffolds, from short-read genomic sequencing datasets, often utilizing next-generation "short-read" sequencing. It consists of a collection of algorithms that store genomic sequencing data in de Bruijn graphs in order to effectively reduce errors and assemble the data into larger sequences. The velveth hashing method merges sequences that belong together, and then the velvetg assembler generates a graph, resolves ambiguous repeats, and separates pathways with local overlaps. Velvet was designed to function within a 64-bit Linux system using the gcc compiler.

The previous version of Velvet, 1.2.0, was designed to operate in multithreaded mode utilizing OpenMP parallel programming.

Here, we introduce a new version of Velvet that utilizes OpenACC directives. OpenACC is a paradigm for parallel programming designed to simplify parallel programming and provide excellent performance and portability across multiple architecture types, including multicore, manycore, and GPUs [1].

OpenACC enables programmers to specify which portions of code to accelerate using simple compiler instructions, without modifying the underlying code. OpenACC directives enable the compiler to move the calculation onto an accelerator by detecting parallel code parts [5].

2 Related Work

The increasing amount of data provided by DNA sequencing necessitated the development of programs to assemble these sequences. There are currently a large number of assemblers used to simplify and speed up the assembly process for consumers.

Different programming languages and paradigms are utilized by these assemblers, as well as distinct techniques. Overlap-Layout-Consensus (OLC) and De Bruijn graph are employed. The OLC method involves establishing alignments and finding overlaps between the reads, combining them into contigs, and finally producing a consensus sequence. In the De Bruijn graph approach, the readings are divided into smaller sequences of fixed length k (or k-mers) known as seeds (Tabela 1).

Table 1: Bioinformatics sequence assembler comparison

Assembler	Program	Paradigm	Algorithm	License
ABySS	C++	MPI	De Bruijn graph	open code
ALLPATHS-LG	C++	OpenMP	De Bruijn graph	open code
Edna	C++	Pthreads	OLC	open code
SOAPdenovo	C++	Pthreads	De Bruijn graph	open code
Velvet	C	OpenMP	De Bruijn graph	open code
CABOG	C	OpenMP	OLC	open code
SPAdes	C++	Pthreads	De Bruijn graph	open code

To choose which DNA sequencing assembler will be utilized, comparative and evaluative studies of de novo genome assemblers were analyzed. These research examined the criteria for the utilization of computational resources, assembly time, and the quality of the results achieved.

The evaluation of the de novo sequence assemblers ABySS, Velvet, Edena, SGA, Ray, SSAKE, and Perga revealed that, despite their ability to process prokaryotic or eukaryotic entire genomes, only Velvet and ABySS demonstrated good efficiency in terms of assembly time and use of computational resources. Despite the fact that the ABySS and Velvet assemblers provide comparable outputs, Velvet has superior scalability to manage vast amounts of data compared to other assemblers [4].

Another significant study conducted was the GAGE (Gnome Assembly Gold-standard Evaluations). Several de novo genome assemblers, including ABySS, ALLPATHS-LG, Bambus2, CABOG, MSR-CA, SGA, SOAPdenovo, and Velvet, were evaluated in this study. Some assemblers, including Velvet, obtained superior outcomes after genome assembly [6].

On the basis of the results reported in these investigations, it was chosen to employ the Velvet assembler in this study since, in addition to its good performance in genome assembly, it is a free program whose code permits implementation using an accelerator programming model.

Here, we introduce a new version of the Velvet assembler that utilizes graphics processing units (GPU) through OpenACC directives. The purpose of developing this version of the

assembler is to give the bioinformatics community with an additional data processing tool alternative.

3 Implementation

For the implementation of the new OpenACC assembler version, directives were utilized to parallelize loops and transfer data between the server and GPU. With this, all data is transferred to the GPU, which performs processing in its local memory. After processing, the data is transferred back to the server.

During the execution of the velveth program, in which the hashing method combines sequences, the GPU device was insufficient. In velvetg, the GPU device was utilized more frequently since it resolves ambiguous repeats and isolates pathways with local overlaps during this stage of graph construction.

The program Velvet has multiple codes. We only modified the routines that required the most computing resources. The pgprof tool was used to generate the Velvet profile in order to determine the regions that utilize the most processing resources (Figure 1).

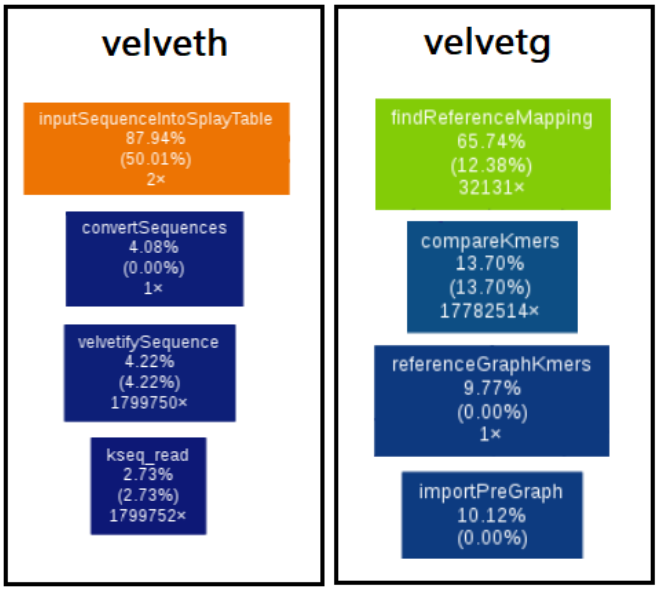

Figure 1: The outcome of the pgprof command used to generate the Velvet profile

During velveth run, the inputSequenceIntroSplayTable operation consumes about 87% of time and processing resources. In this instance, only the process's codes are modified. Similar to how the instructions referencing to this process were modified, the findReferenceMapping process takes around 65% of velvetg's total execution time. The majority of these modifications involved directives such as parallel loop, vector length, and acc data copy.

The Velvet assembler consists of multiple files that are used to generate velveth and velvetg executables. Some files employ the OpemMP programming paradigm to execute the Velvet assembler in parallel. We have modified these files to use the OpenACC programming paradigm.

Parallel directive was utilized to support OpenACC. With the Parallel directive, it is possible to achieve improved performance through the use of clauses and precise parameterizations in

order to maximize GPU utilization. Following is a list of OpenACC pragmas utilized in the files.

- #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)

This is an example of code compilation utilizing the scaffold.c file.

```
# 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)
```

The velveth and velvetg executables were compiled on a 64-bit Linux server utilizing the PGI compiler and NVIDIA GPU device. The latest version is accessible for download at https://github.com/evaldocosta/velveacc.

4 Experimental Setup

The tests were conducted on a server equipped with two Intel Xeon E5-2609 processors (1.7 GHz, 8 cores each, 20 MB cache), 128 GB of shared memory, and an NVIDIA GPU Tesla K80. The NVIDIA Tesla K80 is a dual-GPU system that employs two GK210B chipsets. This card features a total of 4992 CUDA cores clocked at 560 MHz, along with 24GB of GDDR5 vRAM, a 384-bit memory interface, and a 480 GB/s bandwidth.

All versions of Velvet were compiled with the PGI Compiler 19.10 for optimal performance. Local, high-speed SSD (Solid-State Drive) drives were utilized to store the experiment's files. The 64-bit Centos Linux distribution version 7.8 was utilized as the operating system. In both OpenMP and OpenACC implementations, the assembly was executed using the following commands.

There were three sorts of data utilized to evaluate the essays. All the raw data sets are available for download from the European Nucleotide Archive (ENA) and the National Center for

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 .

Biotechnology Information (NCBI) servers. Simulations were performed on the following organisms: Staphylococcus aureus: NCBI SRA (SRR022868, SRR022865), Rhodobacter sphaeroides: NCBI SRA (SRR081522, SRR034528), and Homo sapiens (Chromosome (DRR000546). The summary is presented in Table 2. Using the Linux time command, the total assembly time in seconds was computed for each assembly. Memory and CPU use were measured using the Linux commands smem and mpstat.

Table 2: The size of file data

Species	Genome size (Mbp)	Files
Staphylococcus aureus	2.9	NCBI SRA (SRR022868, SRR022865)
Rhodobacter sphaeroides	4.6	NCBI SRA (SRR081522, SRR034528)
Homo sapiens (chr21)	46.7	ENA (DRR000546)

5 Results

Three sets of tests were performed for the results provided in this study. After each run, the average time of the sets was determined to determine the rate of acceleration achieved in each instance.

5.1 A Reduced Total Runtime

The OpenACC version of Velvet has shorter runtimes than the OpenMP version. Using the genomes of Staphylococcus aureus and Rhodobacter sphaeroides, the average speedup was five times greater, while the average speedup for the human genome was three times greater. The overall gain acquired through genome processing can be seen in the table 3.

5.2 Improved Memory and CPU usage

When utilizing the assembler version of velveth with OpenMP support, memory utilization was less than when using the OpenACC version. This behavior was consistent across all tested genomes (Figure 2).

In addition to reducing total assembly time, OpenACC also improved memory and CPU resource use in comparison to OpenMP, indicating that local computer resources are utilized more

Table 3: The genomic timing outcomes

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

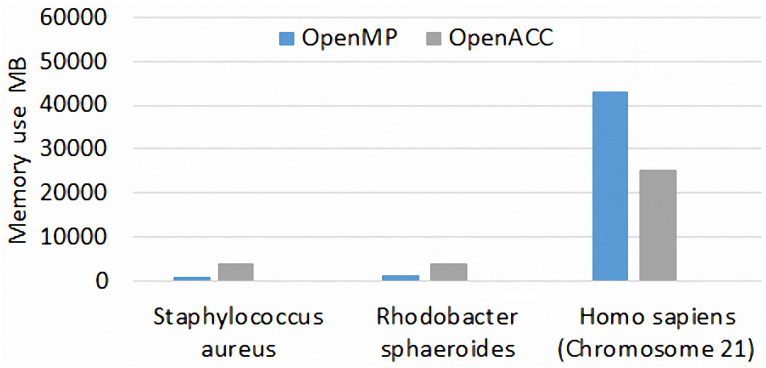

Figure 2: Memory used

efficiently when OpenACC is employed (Table 4). Obviously, this only applies to the genomes of Staphylococcus aureus and Rhodobacter sphaeroides, as the amount of data transferred to the GPU device is minimal.

Table 4: CPU use results

Species	OpenMP	OpenACC
Staphylococcus aureus	52%	86%
Rhodobacter sphaeroides	68%	85%
Homo sapiens (chr21)	72%	44%

5.3 GPU usage

The examination of GPU usage is divided into two sections: the utilization of GPU processing cores and the memory footprint.

The mean use values of the Staphylococcus aureus and Rhodobacter sphaeroides genomes were around 60% lower than those of the Homo sapiens genome (chr21) (Figure 3). This occurred because the analysis of bigger genomes required a greater number of GPU CUDA cores.

Similarly to the average use of GPU cores, the average utilization of GPU memory exhibits the similar pattern, as seen in Figure 4. In other words, larger genomes utilize more GPU memory resources, yet in no instance was the overall memory usage greater than the available GPU memory resources.

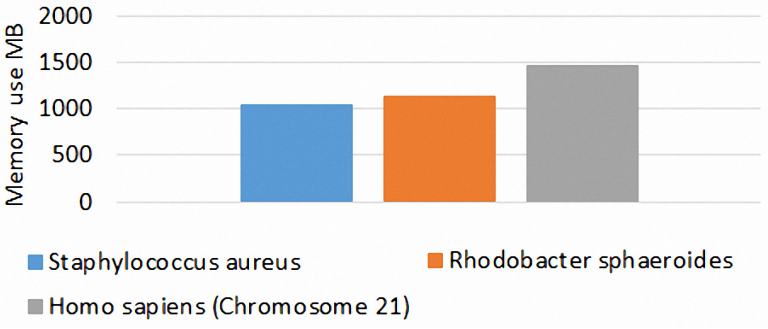

Figure 3: GPU Memory usage throughout assembly

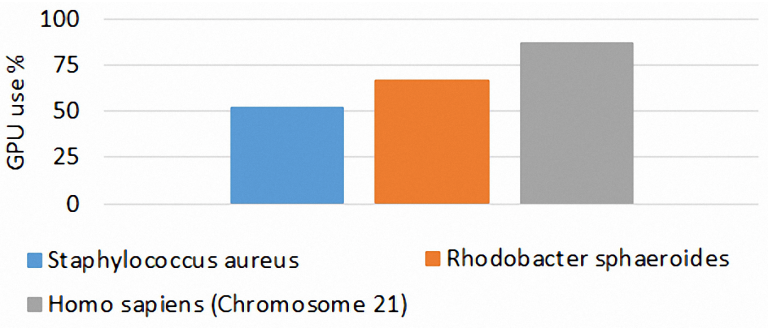

Figure 4: GPU usage throughout assembly

5.4 Data Move

During the assembly of the genome, data transfer between the server and GPU device was also considered as an important factor. This data transfer occurs in both directions, from the server to the GPU as well as from the GPU to the server. During velveth operation, there is little data transfer between the server and GPU device. As this procedure reads genomic files, GPU utilization is minimal. However, the data transfer rate during the execution of velvetg is exceptionally high. This arises because the De Brujn graph is generated during this procedure.

The results of data flow between the server and GPU device are depicted in Figure 5.

Figure 6 demonstrates that the average quantity of data transferred between the GPU and the server is bigger because, in this procedure, information is first executed on the GPU and then all data is transferred to the server.

6 Assembly Quality

After assembling the genomes, the Quast program [3] was used to verify the assembly's quality. The objective of this section is not to compare the assembly quality of the two methods, but rather to demonstrate that both methods are accurate. As demonstrated in Table 5.

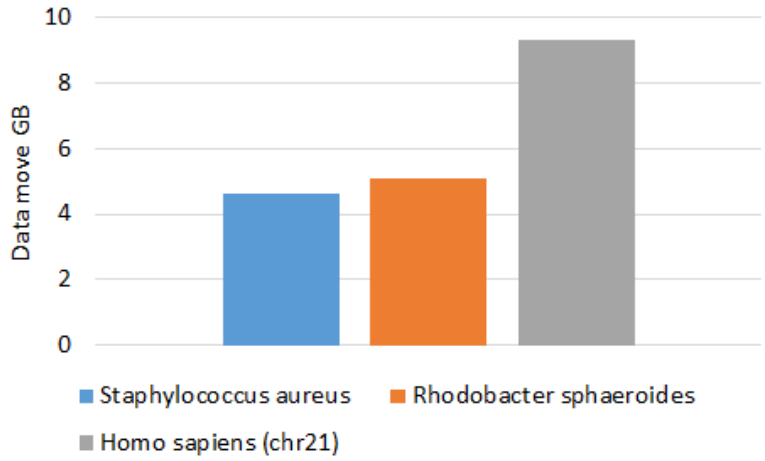

Figure 5: Average usage of data transport between the server and GPU device

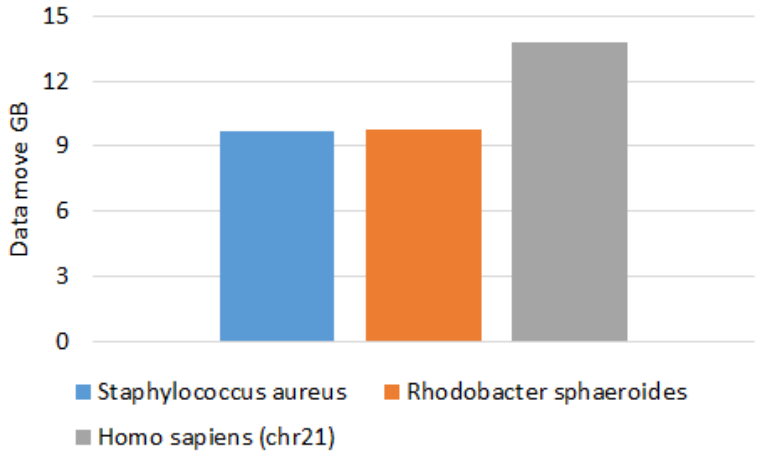

Figure 6: Average utilization of data transport between the GPU device and server

7 Conclusion

Comparing the previous version of Velvet, which utilized the OpenMP programming paradigm, to the new version in OpenACC reveals that the total gain during the entire genome assembly process was up to five times faster for Staphylococcus aureus and Rhodobacter sphaeroides genomes, and the average gain for Homo sapiens chromosome 21 was three times. The updated version of Velvet in OpenACC's source code is available on GitHub at https://github.com/evaldocosta/velvetacc.

The experiments conducted to evaluate the new OpenACC version revealed that the velveth program did not benefit significantly from the OpenACC version over the OpenMP version, necessitating additional research into optimizing its parallelization. During the execution of the velvetg program, both CPU utilization and total execution time decreased significantly,

Table 5: Staphylococcus aureus genome assembly results using the Quast program

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

demonstrating the efficiency of the transfer of computing from the CPU to the GPU.

Due to the building of the Bruijn graph that the velvetg program does, the data transfer between the server and the GPU during genome assembly was greater when executing the velvetg program. In velveth execution, this data flow is modest, as this program focuses primarily on reading the files. Regarding memory usage and GPU core occupancy, both programs exhibited the same tendency, i.e., the larger the genome, the more resources are utilized.

The final results of the evaluations indicate that the use of OpenACC directives in the new version of the Velvet assembler was effective in reducing the execution time of genome assembly by up to fivefold compared to the original version written in OpenMP, demonstrating an efficient use of GPU resources (memory and processing cores).

Acknowledgments

The authors are grateful to the Computer Institute of Rio de Janeiro Federal University for providing the computing resources utilized to conduct the experiments described in this research. Daniel R. Zerbino deserves special thanks for her assistance and suggestions.

References

[1] Shaohao Chen. Introduction to openacc. *Research Computing Services Information Services and Technology Boston University*, 2017.

[2] Evaldo B Costa, Gabriel P Silva, and Marcello G Teixeira. 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, 2015.

[3] Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 02 2013.

[4] Abdul Rafay Khan, Muhammad Tariq Pervez, Masroor Ellahi Babar, Nasir Naveed, and Muhammad Shoaib. A comprehensive study of de novo genome assemblers: current challenges and future prospective. *Evolutionary Bioinformatics*, 14:1176934318758650, 2018.

[5] Jeff Larkin. Introduction to openacc. *NVIDIA*, 2018.

[6] Steven L Salzberg, Adam M Phillippy, Aleksey Zimin, Daniela Puiu, Tanja Magoc, Sergey Koren, Todd J Treangen, Michael C Schatz, Arthur L Delcher, Michael Roberts, et al. Gage: A critical evaluation of genome assemblies and assembly algorithms. *Genome research*, 22(3):557–567, 2012.

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