Journal of Applied Computing Research, 3(2):118-128
July-December 2013

Task scheduling in genetic sequencing tool

Jéfer Benedett Dörr, Guilherme Galante, Luis Carlos E. De Bona
Universidade Federal do Paraná. Rua Cel. Francisco H. dos Santos, 100, 81531-980, Curitiba, Paraná, Brazil
jefer@ufpr.br, ggalante@inf.ufpr.br, bona@inf.ufpr.br

Abstract. This paper proposes a task scheduler to control the demand of sending gaps encountered during the process of genome sequencing processing considering computational resources available. Gaps are spaces without representation in the genome sequencing process. This activity generates many competing tasks that consume a lot of computational resources, mainly memory. The goal of the scheduler is to prevent more required computational resources besides those which can be alive supplied, because in this case, a performance degradation of the system will occur and it may cause a delay in the processing time of the tasks. The motivation for this work is to improve the efficiency of the implementation of the closure of gaps in genome sequencing. For the evaluation of the proposal, a scheduler for gaps with scheduling policies based on monitoring of computing resources has been implemented.

Keywords: bioinformatics, scheduling tasks, genetic sequencing.

Introduction

The information contained in the genome can be of great value to many studies. The genome is hereditary information encoded in deoxyribonucleic acid (DNA) that is passed to the descendants of an organism being formed of monomers called nucleotides. Each of these nucleotides has a molecule termed base, which can be Adenine, Cytosine, Guanine and Thymine.

In order to know the DNA of an organism it is necessary to extract this data using a genetic sequencer (Shendure *et al.*, 2005) and then string them together. The genetic sequencing of the target organism extracts a large amount of small fragments of DNA, repetitive and disordered (Chaisson *et al.*, 2004; Pandey *et al.*, 2008; Sundquist *et al.*, 2007) that must be assembled to obtain a consensus sequence of the bases which represent DNA (Mardis, 2008; Metzker, 2009). The assembly process requires a lot of computational power, and a set of specific and efficient algorithms to perform the sequence (Chaisson *et al.*, 2009; Pevzner *et al.*, 2001).

The pipeline denovo2, illustrated in Figure 1 (Applied Biosystems, 2010; Life Technologies, 2012) is a set of programs widely used for assembling this large volume of data obtained from these data sequencers and to generate one consensus DNA sequence. The assembly is performed by Velvet (Zerbino, 2008; Zerbino and Birney, 2008), an assembler which uses de Bruijn graphs, an approach capable of handling a large amount of data with good coverage (Compeau *et al.*, 2011). After assembly is held, the closing of gaps occurs with the intention of finding bases for the representation of empty positions. In this case, an instance of Velvet assembler is used for each gap, resulting in a large amount of tasks.

These multiple Velvet instances run simultaneously and in some cases may require more memory resources than the computing environment can provide. The assembly tasks are fairly heterogeneous, varying from a few to hundreds of Gigabytes (GB). In order to keep all the tasks running, all the available memory is consumed and, as a consequence, the system enters a state of performance degradation, causing delays in the tasks processing. This situation can be avoided by checking the available computational resources before starting new instances of the assembler. With this objective, a scheduler task has been proposed.

In the scheduler investigated, instead of processing all tasks at the same time and with no criteria as it was originally made, the tasks are triggered by observing the resource avail-

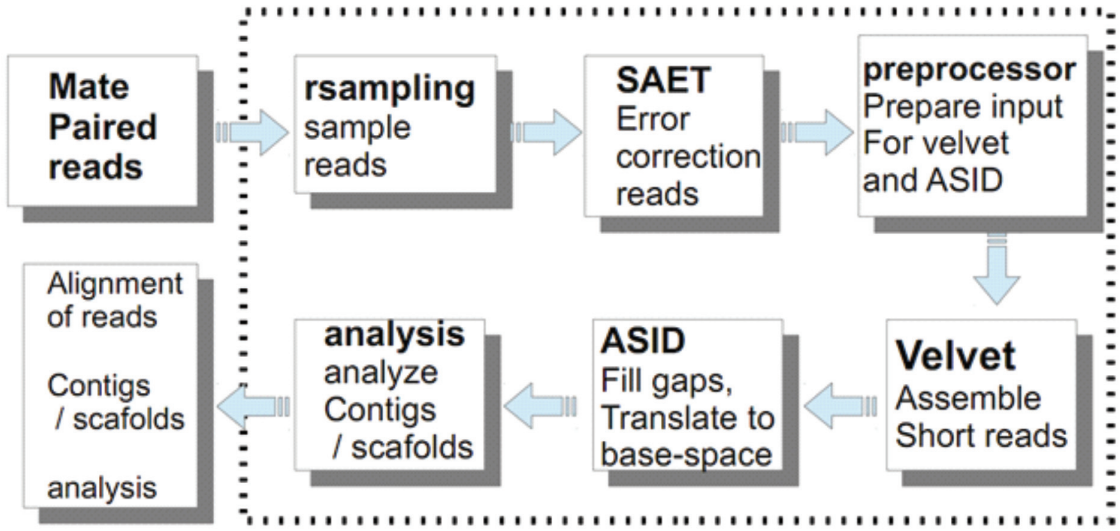

Figure 1. The pipeline denovo2.

ability needed for processing them. Thus, different tasks can be executed consuming resources in parallel, in a controlled manner, causing no performance degradation and resulting in execution time reduction. By observing the availability of resources before submitting a task for execution, this implementation can keep the task within the limits of available computational resources and avoids performance degradation.

The proposed approach proved to be very efficient. In a first version of the scheduler, it was possible to reduce running time in 55% when compared to original implementation. With an enhanced version the time was reduced by 73% compared to the original time.

The rest of the paper is organized as follows: In the section "Related works", related works are presented. The section "Pipeline denovo2: problems found" presents the details of the problem encountered during the execution of the pipeline denovo2 genetic sequencing. The section "Tasks scheduler for denovo2" presents the proposed scheduler. In the section "Forecast of memory usage" the forecast memory usage is presented. In the section "Experimental results" the results of the experiments are presented. Finally, the section "Conclusion and future work" presents the conclusions and suggests future work.

Related works

A task scheduler is considered a component that runs resource management. It is extremely important for parallel and distributed systems, and can be considered one of the most challenging problems in this area. Scheduling is to determine in what order the tasks are executed. This work proposes task scheduling and paging preventing trashing on a parallel supercomputer using expected memory usage of each task, having the following related works.

The problem of resource sharing for many simultaneous tasks in a parallel system (Olivier *et al.*, 2012), which may result in problems making paging not possible to guarantee quality of service for the execution of all tasks is presented in Batat and Dror (2000). Pagination affects the synchronization between tasks. An alternative is to impose access controls and only admit new tasks that fit into available memory. Despite suffering from delayed execution, this leads to a better overall performance by preventing the harmful effects of paging and trashing. In Setia *et al.* (1999) the impact of memory usage is evaluated for task scheduling considering the long-term problems to avoid paging and trashing super parallel computers using the characteristics of memory usage of each task. In the work of Arras *et al.* (2013) the processing subject to strong resource constraints, particularly in terms of memory was approached with extensions to list-scheduling algorithms for taking into account memory requirements. The suggested approach was a parallel scheduling with a new model featuring memory tasks and priority adjustment of the tasks, achieved gain and preventing deadlocks with priority adjustment.

Finally, in Nikolopoulos and Polychronopoulos (2003) the prevention of paging and trashing is exploited for the efficient scheduling of parallel tasks. The challenge is to do as Ghodsi *et al.* (2011) consider the problem of fair resource in a system where each task has different demands.

Pipeline denovo2: problems found

The pipeline denovo2 is a set of programs implemented in different languages that are used together in order to receive fragments from genetic sequencers of new generation, capable of generating a large data volume and arrange them in order to assemble a consensus contiguous genome sequence, which is the result of multiple sequence bases alignment, and each base occurs more often in a given position (Meidanis and Setubal, 1997).

Depending on the data set being processed, the execution pipeline denovo2 can consume all available memory, causing slowdowns in the computer system as a whole and making the task prolonged indefinitely.

To identify the source of the problem, it was necessary to monitor the work performed by each pipeline stage denovo2. For this, an Altix UV 100 machine was used. It offers, in its current configuration, a total of 256 GB of random access memory (RAM) and 64 processing cores. Experiments were executed with the dataset named training1 set research provided by the Department of Biochemistry, Federal University of Paraná. No more information about the data set due to confidentiality of the survey was available.

The training1 set has a size of 111.6 million base pairs, contains 4,756 gaps that occupy 8 GB of disk space, and results in an output of 300 GB of data. Using this data set, the execution of the tasks occurred normally until the assembly phase, however, during the closing gaps phase, we were unable to verify the consumption of the entire memory available impacting application performance.

At this stage, demonstrated in Figure 2, an instance of Velvet (Zerbino, 2009) is executed for each gap found to try to assemble the parts that were left without representation bases. This procedure is repeated four times with different parameters for each gap. Because all instances are executed concurrently, it is possible to estimate approximately 280.000 tasks running concurrently for training1 dataset.

This large amount of concurrent tasks requires memory allocation to keep the gaps in execution. At this stage, the 512 GB of RAM are not enough and 8 GB of swap are also used. By using the virtual memory swap, the speed of access to data drops dramatically (Tanenbaum, 2008). Excessive consumption of RAM by lots of running tasks generate a flow to write and read a lot of data from the hard disk while running, caused by the paging mechanism that makes tasks exchanged between disk and swap. This situation makes the remaining processes wait for a long time in queue to use the central unit processing (CPU). This problem is known as thrashing, which usually causes

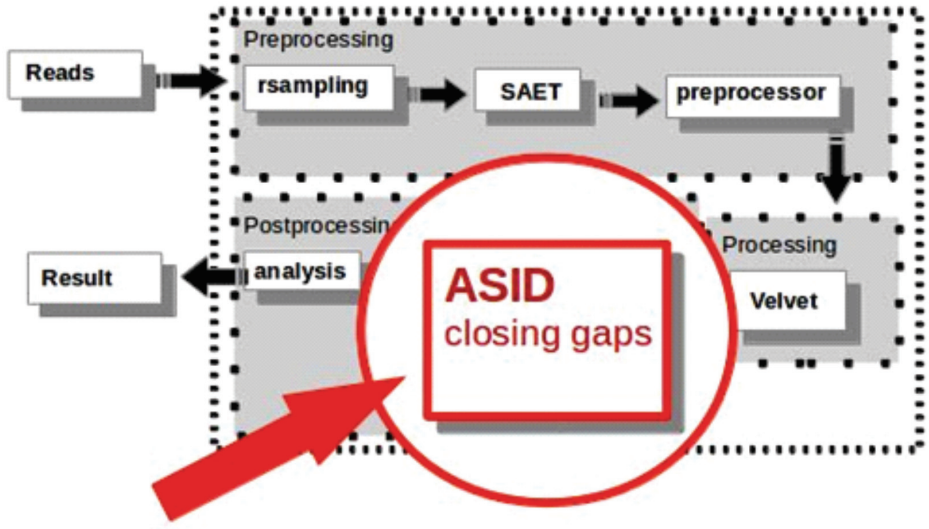

Figure 2. The pipeline denovo2.

serious performance problems making the system unusable (Denning, 2008).

Trashing is a phenomenon that occurs when excessive paging is performed by reducing CPU utilization and throughput. As an aggravating factor trashing the operating system detects that the CPU is idle (during closing gaps CPU usage is below 1%) and admits more processes increasing the level of multiprogramming, and consequently the rate of pages failure, worsening performance (Denning, 1968; Denning, 2008). The graph in Figure 3 adapted from Denning (2008) demonstrates how trashing affects performance; the horizontal axis shows the increase in the level of multiprogramming, while the vertical axis shows the efficiency of the implementation. While an increase of performance is expected by increasing the level of multiprogramming, the yield falls suddenly after a critical load (Denning, 2008).

This is exactly the behavior during the execution of the tests. The use of parallelism aims to improve performance, but when there is an excessive use of parallelism there may come a critical moment when the performance improvement expected from the way to a performance decreases. This performance decrease is caused by the lack of resources which causes the phenomenon called trashing, as reported by Denning (1968) and reassessed by Denning (2008) as shown in Figure 3.

With the experiments, we determined that the problem is the large amount of jobs being executed in parallel. One way to solve this problem is to organize the instantiation of these tasks taking into consideration the availability of computational resources.

Tasks scheduler for denovo2

Scheduling is the action of determining in what order tasks are to be executed. A scheduler is a tool that allows the tasks control in accordance with the policies and the restrictions presented (Casavant and Kuhl, 1988; El-Rewini *et al.*, 1994). The basic problem of the scheduler is to satisfy goals, how to get fast response time, maximize system output (flow), maximize processor utilization, avoid indefinite hold, combine tasks of high and low priority and minimize execution time, (Kunz, 1991; Setia *et al.*, 1999; Tanenbaum, 2008; Wu and Sun, 2004) according to the criteria defined for scheduling.

The scheduling of tasks involves three main components: consumers, politics and resources. Tasks waiting to be executed are consumers. Resources are available computational resources, such as memory, disk and CPU. Finally, the scheduling policy is the set of rules used to determine when and which task should be executed.

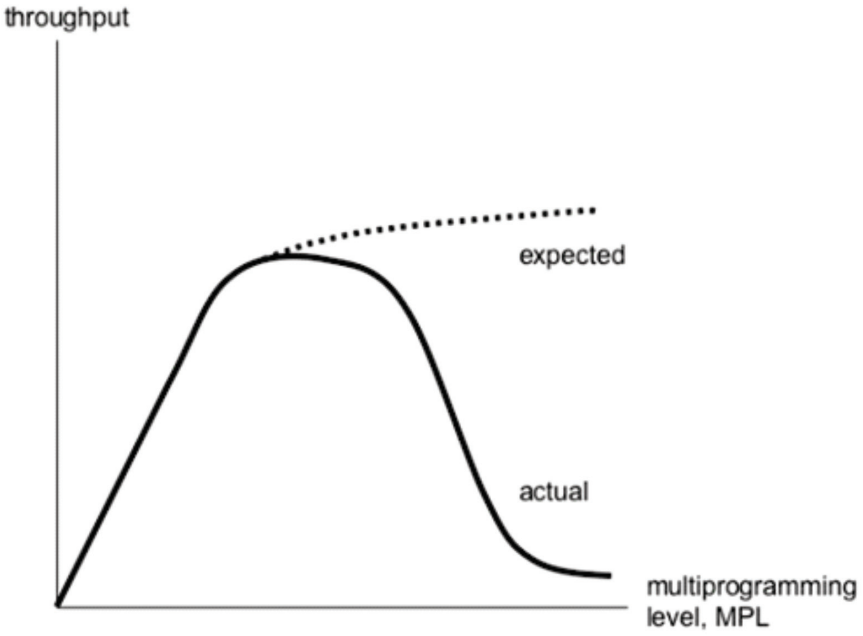

Figure 3. The expected performance being affected by trashing.

The policy defined by the scheduler directly impacts performance of the application that is controlled. The proposed scheduler uses system information such as scheduling policies. When the system receives a new gap to be processed, the scheduler already knows the expected maximum usage of RAM for that gap. Then the free memory of the machine at the time is asked to verify the possibility to run the gap not transcending the limits of available memory, as shown in Section "Experimental results". After performing this query if the condition is satisfied, the scheduler can submit a new gap to run. Then the scheduler receives another gap and performs the same verification process before submitting tasks, as demonstrated by the algorithm of Figure 4. Even in case of suffering some delay in the submission of tasks, execution is guaranteed, all within the machine's capacity limits and thus allowing a more effective implementation resulting in performance gain.

The proposed policy for the scheduler is the use of the prediction of memory usage, considering the free memory given by an internal control. The scheduler triggers a task when the available memory is greater than the amount of memory that was required to perform the task.

Forecast of memory usage

It is possible that different data sets occupy the same space than other sets. Then, two fields with the same space occupied on disk can have a big difference in execution time and memory consumed.

To predict the memory usage for the implementation gaps, 4 sample gaps for 5 different classes of intervals sizes were selected. The samples were classified according to the size occupied on disk and classes: mini (up to 3 Megabytes (MB)), small (3.1 to 17 MB), medium (17.1 to 37 MB), large (37.1 to 77 MB) and extra (above 77.1 MB). For each class the average usage of RAM was calculated and the standard deviation was added, so that these data were used to design expectation maximum memory usage, as shown in Table 1.
With these values, a database of estimated values using each class gap was created during processing. These values are used in the decision of the scheduler policy to evaluate whether to submit new gaps, or wait for the required amount of free memory.

The free memory in the system is stored in a variable, from this moment it is this variable that will answer the expectation of free memory on the machine. As the peak memory

```
for gap[n] do vet_gaps
    gap[n] prev_mem <- size(gap.class)
    queue_gaps.ins(gaps[n], CloseGapX)
end for
while queue_gaps != queue_gaps.empty do
    memory_available <- system.free_memory
    if (memory_available < gap[n].mem_prev) then
       wait
    else
       run(CloseGapX, gap[n])
    end if
end while
```

Figure 4. Implementation of an algorithm for scheduling gaps.

Table 1. Expected usage of RAM by classes gaps.

	mini	**small**	**medium**	**large**	**extra**
Size on disk (*MB*)	3	13	37	77	112
Average memory usage (*GB*)	0,6	3,5	10	32	80
Standard deviation	0,4	0,5	3	3	5
Forecast of memory *RAM* usage (*GB*)	1	4	13	35	85

usage of tasks is close to its end, the scheduler must consider the long term, so for this reason when launching a new task is queried, the free memory is expected in this variable and not directly from the system. The expected maximum memory usage for the gap that will be run is subtracted at the end of this variable and this value is returned. Thus it is a parallel control of the amount of available memory and that can be allocated without overloading the system.

Experimental results

To evaluate the results, a comparative analysis of the behaviour of computational resources was made during the original scheduler execution with the original implementation of the proposed scheduler, which implements the policies scaling the points raised in this work that could result in performance improvement. The tools used in the monitoring of resources were GNU/Linux operating system SUSE Enterprise Server 11 SP1 x64 tools. The results were obtained by observing the logs generated during the execution of each of the two alternatives for closing the gaps.

As a consequence of resource monitoring and management tasks using training1 set, the graph in Figure 5 shows the difference of the times obtained in implementing the two approaches using the same set of gaps. On the horizontal axis the time in hours is shown and on the vertical axis the original scheduler and the proposed scheduler are represented. The original scheduler is represented in blue and the proposed scheduler is represented in red.

The graph in Figure 6 shows, in red, the memory usage while running the original scheduler; and, in blue, while implementing the alternative proposed scheduler with training1 set. The vertical axis shows, in GB, the amount of RAM used in runs while horizontal axis represents the time in hours. The green line indicates the limit in GB of available memory, and the amount that should never be extrapolated.

As it can be seen in the graph of Figure 6, the original scheduler, in red, required the entire available RAM and consumed the entire swap while maintaining maximum use practically throughout all the execution. The use of multiprogramming with consumption of the entire memory, represented by the green line, and swapping all trashing caused the phenomenon of extending the time required to complete the process and obtain the result. On the other hand, the proposed scheduler in blue never extrapolates the available memory and maintains a safety margin to leave memory available to the system, represented by the green line. When using RAM comes close to the memory limit set to the minimum necessary to keep the system up and running, the proposed scheduler waits for free memory to instantiate new gaps. For this reason, curves can be observed in Figure 6 with the saw-tooth like behaviour. When enough memory is released, another gap is launched for execution. The memory will be released and the scheduler will dynamically perform the query until the free amount is sufficient for all processing of the next gap, according to the prediction for memory usage of the scheduling policy.

The graph in Figure 7 shows the CPU usage during the execution of the original scheduler and the proposed scheduler to compare the two approaches with training1 set. In red, the CPU usage is represented during the execution of the original scheduler, which is due to the problem of waiting for input and output operations which is due to the problem of waiting for input

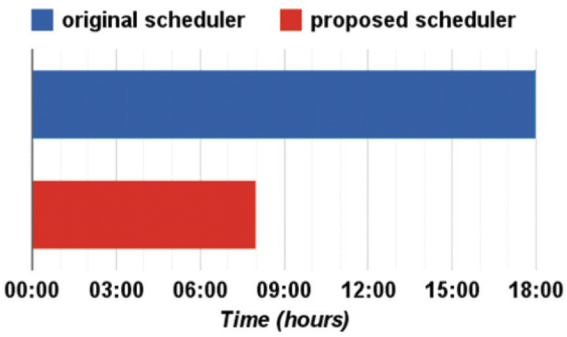

Figure 5. Comparison of execution times.

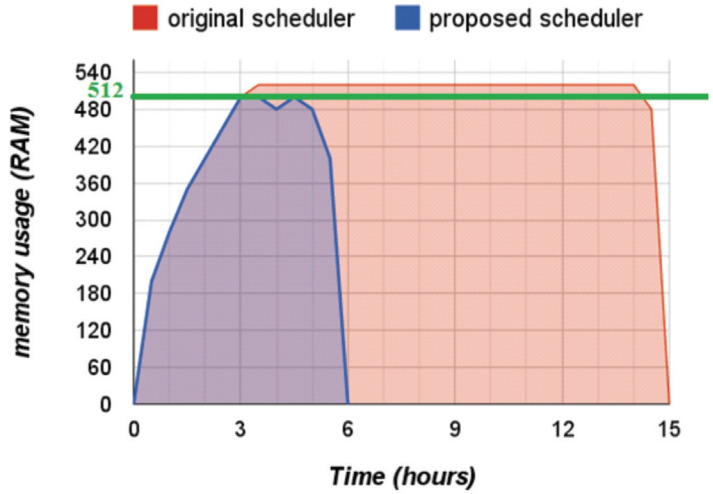

Figure 6. Implementation of an algorithm for scheduling gaps.

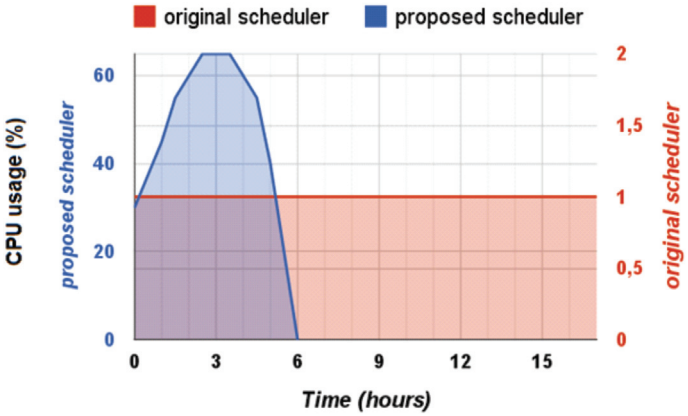

Figure 7. Statement of CPU usage during the execution of alternatives for closing gaps.

and output operations caused by trashing the reduced CPU utilization and remains low, below 1%. In blue, it is possible to observe a better usage of the CPU time by the proposed scheduler, achieved through better management of available resources and showing that avoiding bottlenecks could improve the use of CPU. By monitoring the resources in order not to overload the system and launching new tasks only when the required resource is available for execution, it was possible to avoid the problem of trashing. Not suffering from degradation caused by trashing can better benefit the processing power and with it the runtime decreases.

Disk usage was monitored by the result of iostat command of GNU/Linux operating system, which reports statistics of input and output of the system. In the extended mode, by using the -x parameter display you can view the result of the %util, which indicates the percentage of utilization, the flow, and disk access.

This field indicates how busy your disk capacity is to serve new requests. The %util field shows 100% in disk access when disk access is saturated and the device becomes a bottleneck, reducing system performance. Values above 100% indicate that the disk system is overloaded, resulting in degradation of disk access performance. Reading and writing operations are a bottleneck and slow the progress of the execution of the closure of gaps. With control of memory, a result from using the scheduler, this index remained low and contributed to the reduction in runtime.

Figure 8 compares the results of the iostat command demonstrating the peak percentage

of disk utilization during the execution of the original scheduler and of the proposed scheduler with training1 set. As it can be seen in the graph of Figure 8, the value of the original variable %util scheduler, in red, reached peak values of 5000%. In the case of the proposed scheduler, in blue, this value remained below 10% throughout the run time, not causing any delay.

As shown in Figures 6 and 8 memory resources and disk access were required in addition to their availability, which caused performance bottleneck and would not allow use of processing, as shown in Figure 7, and consequently caused delay in the execution of tasks. Managing the use of these resources enabled to improve performance by avoiding the bottleneck problem and trashing, being able to reduce the execution time as shown in Figure 5.

All work undertaken so far in this article used the sequential version of the assembler (each task uses only one CPU core). This decision was taken because the parallel execution generated a high amount of tasks and problems that did not allow a complete execution of the task. Analyzing the sequential version the results proved the possibility of improving results, then, tests using the parallel version were resumed. In this first approach parallelism was totally eliminated because it was being used incorrectly without getting any performance gain. The problem was to try to parallelize several small tasks, as well as to deplete the resources available for the high amount of tasks to be processed. The time for the management of parallelism was greater than the time it would gain the goal. Therefore, the first approach eliminated the parallel executions. But as parallelism is advantageous for cases where the processing takes longer, we decided to test a conditional parallelism, which for the larger task parallelism was used and the smaller tasks were performed sequentially. This is approach v2.

Parallelism is an effective technique for performance gain on large tasks. The size of the task was the reason for the parallel version to take longer than the sequential version, since most of the tasks to be performed were classified as mini or small group. With parallelism an overhead of managing threads occurs, running in case of small tasks. The usage of parallelism ends up generating more overhead than gain time. The larger task parallelism is advantageous. Therefore, its use for the global event for all tasks was not advantageous due to the high number of small tasks. But differentiating the use for large jobs using the parallel version of the assembler and a small sequential version seems a good pick and will be the next approach in trying to improve the runtime.

To evaluate this second version of the proposed scheduler, a second set of data has been used. The sampling set is the first set containing representatives of all sizes of tasks but in smaller quantities. In this second set there are 300 gaps and the results of processing times can be seen in Figure 9, where the horizontal axis indicates the time in hours and the vertical axis shows 3 versions used in the test: the original version in blue, the version that was the first proposed scheduler in red, and in yellow the proposal to improve the scheduler, called scheduler v2. In the proposed scheduler v2, differentiation of sequential and parallel use of the assembler used for closing gaps was performed.

With this set of data, the observed reduction of the original round to round using the proposed scheduler followed the same behav-

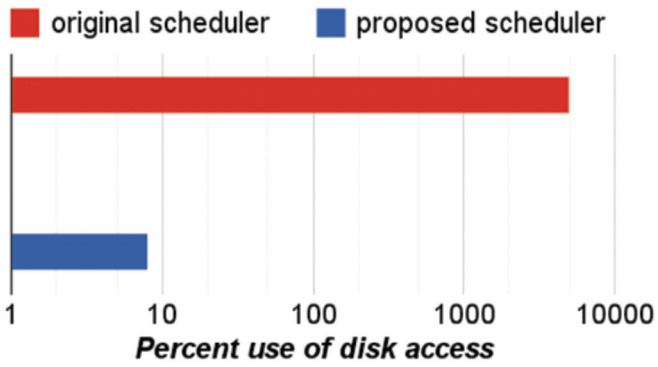

Figure 8. Statement waiting for disk access.

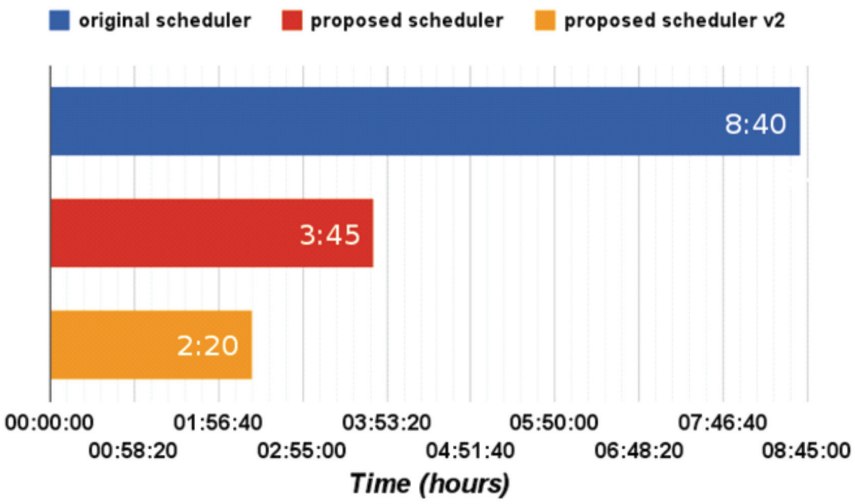

Figure 9. Comparing the three proposals.

ior as the first round of training set, reducing by approximately 55% the time necessary to complete execution. The proposed differentiation using size as a criterion task to decide the use of a parallel or a sequential version of the assembler scheduler, termed as proposed scheduler version 2, achieved a 73% reduction in the time required for a full implementation.

Conclusion and future work

By using the scheduler it was possible to keep track of tasks that used resources efficiently and to be able to run multiple tasks in parallel. With this approach, the execution time in the experiments was reduced by 55% in relation to the time obtained from the original scheduler and reduced implementation in 73% compared to the original time using the scheduler with differentiate use of sequential or parallel implementation assembler, according to the size of the task to be processed.

This work discussed only part of the pipeline denovo2 where the attempted closing of the gaps found is done, but it is possible to seek improvements and more efficient implementations also in other stages of this widely used tool in the quest for knowledge of DNA. Although this work is punctual at one of many phases that comprise the genetic sequencing pipeline denovo2 we identified and provided an efficient solution to a problem that prevents its execution thus impeding the work of genetic sequencing performed at the Department of Biochemistry and Molecular Biology of the Federal University of Parana and others that use the same tool for genetic sequencing.

As future work, it is possible to improve this scheduler updating field forecast to keep memory usage in parallel execution as much as possible with the task. The scheduler can be improved by using the best-fit scale to choose the best task and not wait for the resource to the next queue. As the processing of each task is independent, it can be proposed as a distributed processing. As there is still a large flow of reading and writing to disk all the time during a processing, execution in memory could be more efficient. It eliminates the bottleneck of reading and writing.

Acknowledgements

I would like to thank the Department of Biochemistry and Molecular Biology of the Federal University of Paraná, which provided the data set used, monitored and analyzed the results of this work, especially Professor Leonardo Magalhães Cruz and his PhD student Vinicius Weiss. I would also like to thank the department of Computer Science of the Federal University of Paraná for the staff, the algorithmic and computational support to the Altix UV 100 supplied by Professors Marcos Alexandre Castilho and Fabiano Silva.

References

APPLIED BIOSYSTEMS. 2010. SOLiD de novo accessory tools 2.0 1. Available at: http://gsaf.cssb.ute-

xas.edu/wiki/images/7/71/DeNovo_Assembly_Pipeline_2.0.pdf. Accessed on: February 02, 2012.

ARRAS, P.; FUIN, D.; JEANNOT, E.; STOUCHININ, A.; THIBAULT, S. 2013. List Scheduling in Embedded Systems under Memory Constraints. *In*: International Symposium on Computer Architecture and High Performance Computing, 25th, Porto de Galinhas, 2003. *Proceedings…* IEEE, p. 152-159.

http://dx.doi.org/10.1109/SBAC-PAD.2013.22

BATAT, A.; DROR, F. 2000. Gang scheduling with memory considerations. *In:* International Parallel and Distributed Processing Symposium, 14th, Cancun, 2000. *Proceedings…* Cancun, p. 109-114.

CASAVANT, T.L.; KUHL, J.G. 1988. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions Software Engineering* **14**(2):141–154.

http://dx.doi.org/10.1109/32.4634

CHAISSON, M.J.; BRINZA, D.; PEVZNER, P. 2009. De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome research*, **19**(2):336-46.

http://dx.doi.org/10.1101/gr.079053.108

CHAISSON, M.; PEVZNER, P.; TANG, H. 2004. Fragment assembly with short reads. *Bioinformatics*, **20**(13):2067–74.

http://dx.doi.org/10.1093/bioinformatics/bth205

COMPEAU, P.; PEVZNER, P.; TESLER, G. 2011. How to apply de Bruijn graphs to genome assembly. *Nature biotechnology*, **29**(11):987–991. http://dx.doi.org/10.1038/nbt.2023

DENNING, P.J. 1968. Thrashing: its causes and prevention. *In:* Fall joint computer conference, Fall, part I, New York, 1968. *Proceedings…* ACM, p. 915-922.

DENNING, P.J. 2008. Thrashing. *In:* Wiley Encyclopedia of Computer Science and Engineering. [s.l.], Hoboken John Wiley & Sons cop.

http://dx.doi.org/10.1002/9780470050118.ecse967

EL-REWINI, H.; LEWIS, T.G.; ALI, H. 1994. *Task scheduling in parallel and distributed systems.* Englewood Cliffs, Prentice-Hall, 290 p.

http://dx.doi.org/10.1109/2.476197

GHODSI, A.; ZAHARIA, M.; HINDMAN; B. KONWINSKI, A.; SHENKER, S.; STOICA, I. 2011. Dominant resource fairness: fair allocation of multiple resource types. *In:* USENIX conference on Networked systems design and implementation, 8th, Berkeley, 2011. *Proceedings...* Berkeley, p. 24.

KUNZ, T. 1991. The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions Software Engineering*, **17**(7):725–730.

http://dx.doi.org/10.1109/32.83908

LIFE TECHNOLOGIES. 2012. Project denovo. Available at: http://solidsoftwaretools.com/gf/project/denovo. Accessed on: July 18, 2012.

MARDIS, E. 2008. The impact of next-generation sequencing technology on genetics. Trends in genetics: *TIG*, **24**(3):133–41.

http://dx.doi.org/10.1016/j.tig.2007.12.007

MEIDANIS, J.; SETUBAL, J.C. 1997. *Introduction to Computational Molecular Biology*. Boston, PWS Publishing Company, 296 p.

METZKER, M.L. 2009. Sequencing technologies, the next generation. *Nature Reviews Genetics*, **11**(1):31–46. http://dx.doi.org/10.1038/nrg2626

NIKOLOPOULOS, D.S.; POLYCHRONOPOULOS, C.D. 2003. Adaptive scheduling under memory constraints on non-dedicated computational farms. *Future Generation Computer Systems*, **19**(4):505-519.

http://dx.doi.org/10.1016/S0167-739X(03)00031-1

OLIVIER, S.L.; PORTERFIELD, A.K.; WHEELER, K.B.; SPIEGEL, M.; PRINS, J.F. 2012. OpenMP task scheduling strategies for multicore NUMA systems. *International Journal of High Performance Computing Applications*, **26**(2):110-124.

http://dx.doi.org/10.1177/1094342011434065

PANDEY, V.; NUTTER, R.C.; PREDIGER, E. 2008. *Next-generation genome sequencing: Towards personalized medicine*. Weinheim, Wiley-VCH verlag GmbH Co, 260 p.

PEVZNER, P.; TANG, H.; WATERMAN, M.S. 2001. An Eulerian path approach to DNA fragment assembly. *National Academy of Sciences of the United States of America*, **98**(17):9748–9753.

http://dx.doi.org/10.1073/pnas.171285098

SETIA, S.; SQUILLANTE, M.; NAIK, V. 1999. The impact of job memory requirements on gang-scheduling performance. *SIGMETRICS Performance Evaluation Review*, **26**(4):30–39. http://dx.doi.org/10.1145/309746.309751

SHENDURE, J.; PORRECA, G.J.; REPPAS, N.B.; LIN X.; MCCUTCHEON, J.P.; ROSEMBAUM, A.M.; WANG, M.D.; ZANG, K.; MITRA, R.D.; CHURCH, G.M.; 2005. Accurate multiplex polony sequencing of an evolved bacterial genome. *Science*, **309**(5741):1728–32.

SUNDQUIST, A.; RONAGHI, M.; TANG, H.; PEVZNER, P.; BATZOGLOU, S. 2007. Whole-Genome Sequencing and Assembly with High-Throughput, Short-Read Technologies. *PloS one*, **2**(5):484.

http://dx.doi.org/10.1371/journal.pone.0000484

TANENBAUM, A.S. 2008. *Modern Operating Systems*. Upper Saddle River, Prentice-Hall, 1072 p.

WU, M.; SUN, X. 2004. Memory conscious task partition and scheduling in grid environments. *In:* IEEE/ACM International Workshop on Grid Computing, 5th, Washington, 2004. *Proceedings…* Washington, p. 138-145.

http://dx.doi.org/10.1109/GRID.2004.43

ZERBINO, D.R. 2008. Velvet Manual - version 1.1. Available at: http://helix.nih.gov/Applications/velvet_manual.pdf. Accessed on: December 19, 2012.

ZERBINO, D.R. 2009. *Genome assembly and comparison using de Bruijn graphs*. Hinxton, UK. Ph.D. Thesis. University of Cambridge, 149 p. Available at: http://www.ebi.ac.uk/sites/ebi.ac.uk/files/shared/documents/phdtheses/daniel_zerbino.pdf. Accessed on: December 16, 2012.

ZERBINO, D.R.; BIRNEY, E. 2008. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome research*, **18**(5):821–829. http://dx.doi.org/10.1101/gr.074492.107

Submitted on: January 06, 2014
Accepted on: August 08, 2014