Velvet performance in the computing service of the UPV/EHU

June 10, 2012

General Service of Informatics Applied to the Research
(Scientific Computing)
f IZO-SGI

Contents

1 Introduction

Velvet is a set of algorithms designed as read sequences assembler. In this report we present a summary of the results derived from a extensive series of benchmarks of Velvet. This report will mainly help the researchers to predict the computational resources they will need and plan their work.

2 Environment

This benchmark has been run in the machines of the Computing Service of the University of the Basque Country (http://www.ehu.es/sgi). Most of the benchmarks have been run in a 6 core E5645 Xeon processor at 2.4 GHz with hyperthreading deactivated and X86_64 architecture. Each computing node has 2 processors, i.e., 12 cores. Reads were obtained from the genome sequencing through Illumina HiSeq2000 massive sequencing technology.

In the shown benchmarks Velvet has been compiled with OpenMP and GNU compilers. The serial version show similar results to the OpenMP one when using 1 thread.

The data files are real files of sequences with 100 bp per sequence. We have used uncompressed files to remove from the benchmark the decompress ion time. We have found that approximately 1 GB of compressed data (.fasq.gz files) becomes in 2.7 GB of uncompressed data and has 11.2 millions of sequences. This can be useful to obtain prediction based on the size of the data files, which are more accessible than the number of sequences inside the files.

The used commands have been

```
velveth DATA_1 31 -fastq -shortPaired2 data/*.fastq
velvetg DATA_1 -min_contig_lgth 200 -read_trkg yes -amos_file yes
```

3 Preliminary facts

This benchmark have been realized by the technicians of the Computing Service, with a limited knowledge in bioinformatics.

The data comes in pairs of files and each pair corresponds to one sample. The program has been compiled with a $CATEGORY$ equal to the number of samples. The -sortPaired$X$ option, where $X = 2, 3, \ldots$, works for 2,3,... samples or $CATEGORY$ value, otherwise the program seems to work bat the `Roadmap` file is not generated. This is not clearly shown in the documentation of the program.

If $KMER$ is smaller than the default value 31 velvet must be run with the `-accel_bits 15` option, not documented in the manual.

We have compiled the program using the Intel as well as the GNU compiler. The Intel compilation has been discarded because it uses a lot of memory in the parallelized version.

It is remarkable that when asking n threads with the standard variable $OMP\_NUM\_THREADS$ *velveth* uses $n + 1$ in the parallelized part, but the speed up corresponds to n. I.e., when $OMP\_NUM\_THREADS = 1$ *velveth* uses two threads but the time is equal to the serial version. When speaking about cores or threads we will refer to the value assigned to $OMP\_NUM\_THREADS$.

The RAM memory used has not been be very accurately measured, we can estimate that the error is not longer than 10%. This is due to the fact that the memory usage is not constant, varies along the calculation and the system checks the memory usage at intervals and not continuously. *velvetg* at some point seems to use more RAM during a short period of time, compared to the complete calculation time.

Not all the benchmarks are shown, other ones have been made to check that the trends described in the document can be extrapolated in a wider range of situations.

Tabla 1: Time in seconds used by *velveth* as a function of the number of threads.

	1 core	4 cores	8 cores	12 cores
Time (s)	554	197	122	104
Speed up	1	2.8	4.5	5.3

Tabla 2: Time in seconds and RAM memory in GB used by *velveth* as a function of the number of sequences in millions.

Million of sequences	1.84	3.68	7.36	14.7	29.4
Time (s)	35	81	197	474	1098
Memory (GB)	5	11	14	26	

4 Results

We have benchmarked both *velveth* and *velvetg*. In this section we show the results for this two programs

4.1 *velveth*

First we show in the table 1 the behavior of *velveth* when increasing the number of used cores or threads. We see that *velveth* does not scale very well when increasing the number of cores. With 4 cores we obtain a speed up of 2.8 what means an 70% of efficiency in the use of the 4 cores. Above 4 threads the performance is degraded and it does not worth to use more.

We recommend to submit *velveth* in parallel and asking four cores.

Later we have measured the increase in time as a function of the number of sequences in the data file. In the table 2 we show the time used by *velveth* using 4 cores. We see a linear behavior and the used time is fitted very accurately with the following formula:
where N is the number of millions of sequences. For long simulations we can obtain the execution time in hours as a function of the compressed data size in GB S as:

$$hours = 0.12S \tag{1}$$

More important in Velvet than the time is the memory used by the program. In the table 2 we show the evolution of the used RAM memory as a function of the millions of sequences. We observe again a linear trend that is described by

$$mem = 1.7N + 2.7 \tag{2}$$

where mem is the memory in GB and N is the number of millions of sequences. For long simulations, we can obtain the needed RAM memory in GB as a function of the size in GB S of the compressed data as:

$$mem = 19S \tag{3}$$

To compare with the ABySS assembler, in other benchmarks, we have fitted also to the logarithms of the data. The obtained equation is

$$mem = \frac{n^{0.80}}{19500} \tag{4}$$

where mem is the memory in GB and n is the number of sequences (do not confuse with the million of sequences N). The results of these fitting are qualitatively similar to the regression line.

For long simulations, we can obtain the needed RAM memory in GB as a function of the size in GB S of the compressed data as:

$$mem = 22.3S^{0.80} \tag{5}$$

4.2 *velvetg*

First we show in the table 3 the behavior of *velveth* when increasing the number of used cores or threads. We see that *velvetg* does not scale well at all when increasing the number of cores. With 4 cores we obtain a speed up of 1.5 what means an 38% of efficiency in the use of the 4 cores. We observe during the execution that the most of the time is using one core, i.e., only a small part of the code runs in parallel. For more than 4 cores the performance is very bad and it does not worth to use more.

Tabla 3: Time in seconds used by *velvetg* as a function of the number of threads.

	1 core	4 cores	8 cores
Time (s)	1358	886	728
Speed up	1	1.5	1.9

We recommend to submit *velvetg* asking one or four cores as much.

Later we have measured the increase in time as a function of the number of sequences in the data file. In the table 4 we show the time used by *velvetg* using 4 cores, same number of cores as previously for *velveth*. We see a linear behavior and the used time is fitted very accurately with the following formula:

$$time = 193N - 398 \tag{6}$$

where N is the number of millions of sequences. The time need when using one thread instead of 4 is approximately 1.5 times larger. *velvetg* needs 4.7 times more time that *velveth*. For long simulations we can obtain the execution time in hours as a function of the compressed data size in GB S as:

$$hours = 0.60S \tag{7}$$

More important in Velvet than the time is the memory used by the program. In the table 4 we show the evolution of the used RAM memory as a function of the millions of sequences. We observe again a linear trend that is described by

$$mem = 2.2N - 4.0 \tag{8}$$

where mem is the memory in GB and N is the number of millions of sequences. For long simulations, we can obtain the needed RAM memory in GB as a function of the size in GB S of the compressed data as:

Tabla 4: Time in seconds and RAM memory in GB used by *velvetg* as a function of the number of sequences in millions.

Million of sequences	1.84	3.68	7.36	14.7	29.4
Time (s)	148	343	886	2225	5399
Memory (GB)	5	11	14	26	63

Tabla 5: Time in seconds and RAM memory in GB used by velvet in different nodes.

		Time (s)	Memory (GB)
96 GB node	*velveth*	1098	52
	velvetg	5399	63
48 GB node	*velveth*	1515	51
	velvetg	47955	61

$$mem = 25S \tag{9}$$

To compare with the ABySS assembler, in other benchmarks, we have fitted also to a line the logarithms of the data. The obtained equation is

$$mem = \frac{n^{1.17}}{10^7} \tag{10}$$

where mem is the memory in GB and n is the number of sequences (do not confuse with the million of sequences N). The results of these fitting are qualitatively similar to the regression line. For long simulations, we can obtain the needed RAM memory in GB as a function of the size in GB S of the compressed data as:

$$mem = 19.4S^{1.17} \tag{11}$$

velvetg at some point seems to use more RAM for a short period of time (short compared to the complete execution time). In table 4 we think that we have not measure it always this memory peak, but probably the largest calculations has it included because the calculation has been more time in this part of the run and the system could have recorded it. Therefore, the above formula provably is quite good, or unless do not underestimate the memory for large calculations because the slope has been increased by the correct measurement of memory of the last data points and the shorter one of the first data points, in case of a worse measurement.

5 Memory swapping

Velvet is a very memory consuming program. It is possible to use the disk as RAM memory, this is called swapping. This usually slows down a lot of the calculation time but it depends on the application so we have checked it. We have measured the time and memory of a calculation with 24.4 millions of sequences in a node with 96 GB of RAM and in a node with 48 GB of RAM. We have used 4 cores. In the smaller node Velvet have to use the swap memory. In the table 5 we show the results.

We observe in the table 5 that the *velveth* calculation uses 52 GB of RAM. Therefore, in the node with 48 GB of RAM it is swapping more or less the 10% of the memory and the impact in the consumed time is an increase in a 40%. In the case of *velvetg* it needs 61 GB of memory, so it is swapping in disk about the 30% of the memory and this increases the calculation time in a 900%.

The increase of the time with the swapped memory is not lineal, a little increase in the used swap memory increase a lot the computing time, so calculations that exceed appreciably the amount of available RAM could not be performed. Some of the *velveth* benchmarks we tried to do and that should need about 10 minutes to finish if enough RAM was available did not finish in 30 hours.

6 Several samples

We have checked that the time and memory used by *velvetg* or *velveth* is independent of the number of samples or $CATEGORY$ and depends only on the number of sequences.

7 Compiling with different parameters

We have try to reduce the used memory recompiling the program without $BIGASSEMBLY$ but the used memory was the same. We tried as well to recompile and change the $KMER = 21$ but again the used memory was the same (In this case we used for *velvetg* the option `-accel_bits 15`). In addition, we reduced the optimization of the code at compilation time (`-O0`) but the memory usage was again the same.

8 Itanium architecture

Most of the shared memory machines (big amount of memory) are based on this architecture and this was the motivation to compile Velvet for this architecture. As expected in this kind of genetic codes, the computing time increased a lot. Even this computing time could be affordable to take the advantage of the big amount of memory, we have measured that the memory usage increased quite a lot as well, so we have not found a clear advantage on the Itanium architecture.

Anyway, in case it is need to use this machines, a more extensive compiling study must be done to try to reduce the amount of used memory, unless to approach to the values obtained for the Xeon architecture. For this report this has not been done, we only performed a standard compilation.

9 Practical example

If we want to assemble 100 GB of data compressed in fastq.gz format to run *velveth* we will need according to equations 1 and 3 about 12 hours and 1900 GB of RAM memory and for *velvetg* applying equations 7 and 9 about 60 hours and 2500 GB of RAM memory.

10 Conclusions

Velvet can assembly a file of sequences in a very short time from the point of view of the high performance computing, but it is a voracious consumer on RAM memory and very special machines are needed to assemble large data files or the data of several samples at the same time.