SGA on HPC

What is SGA?

SGA (String Graph Assembler) is a de novo genome assembler, which is a computational tool used to construct a genome sequence from a set of raw sequencing data.

De novo assembly refers to the process of reconstructing a genome sequence without a reference genome. In other words, the assembler works by assembling small fragments of DNA called reads, generated by high-throughput sequencing technologies, into a larger, contiguous sequence.

SGA specifically constructs a de Bruijn graph from the reads and then simplifies the graph using an algorithm called string graph. The string graph is then used to construct the genome sequence.

One of the key advantages of SGA is its ability to handle large and complex genomes, as well as multiple sequencing platforms, including short-read and long-read sequencing data. Additionally, SGA has been optimized to be highly scalable, enabling it to assemble even extremely large genomes with high accuracy.

Links:

Official Website

ReadME

Versions Available:

The following versions are available on the cluster:

- SGA 0.10.15

How to load SGA?

To load SGA, use the following commands:

```
#Load the SGA module
module load bio/sga
```

To verify if the module is loaded correctly, use the following command,

```
# List all the module loaded in the environment
module list
```

In a fresh environment, this only SGA module should be loaded.

How to use SGA?

Here are few general steps about how to use SGA:

1. Quality control: Before starting assembly, it is essential to ensure that the sequencing reads are of good quality and free of contaminants. You can use quality control tools like FastQC to assess the quality of the sequencing data.
2. Pre-processing: Once the quality of the reads is confirmed, you need to pre-process the data to remove low-quality reads, adapters, and other artifacts. SGA has a built-in module called sga preprocess that performs these steps.
3. Error correction: SGA also has a module called sga correct that performs error correction of the pre-processed reads. This step is particularly useful for improving the accuracy of the assembly.
4. Assembly: The main assembly step involves constructing the de Bruijn graph and then simplifying it using the string graph algorithm. This step can be performed

using the sga assembly module, which allows you to specify various parameters such as k-mer size and minimum overlap length.

5. Post-processing: Once the assembly is complete, you can use SGA's post-processing modules to polish the assembly and identify potential misassemblies. These modules include sga gapfill, sga scaffold, and sga filter.
6. Evaluation: Finally, you should evaluate the quality of the assembly using various metrics such as N50, contig/scaffold number and size, and genome completeness. You can use tools like QUAST or BUSCO for this purpose.

Use the following slurm script as reference for using SGA,

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --mem=64G
#SBATCH --time=12:00:00
#SBATCH --job-name=sga_test
#SBATCH --p main
#SBATCH --qos main
module load bio/sga

# SGA COMMANDS TO CLEAN AND ASSEMBLE ON SAMPLE TWO FASTQ FILES
# SGA's pre-processing module
sga preprocess --pe-mode 1 -o reads.pp.fastq read1.fastq read2.fastq
# SGA's error correction module
sga correct -o reads.ec.fastq reads.pp.fastq
# SGA's assembly module
sga assembly -m 100 -o assembly --min-assembly-length 100
reads.ec.fastq
```

All the files should be in the same files and the script location. Check the GitHub page for more information and sample test cases.

Where to find help?

If you are confused or need help at any point, please contact OIT at the following address.

https://ua-app01.ua.edu/researchComputingPortal/public/oitHelp