paraGSEA tutorial

Shunyun Yang

May 4, 2017

1 Build instructions

paraGSEA runs on Mac and Linux as a command line application. You can download the source code of paraGSEA from Github with the following command on a standard terminal.

git clone https://github.com/ysycloud/paraGSEA.git

Note that this requires that you already have a Github account and that the computer you are working on has an SSH key registered on Github. If this is not the case, follow the instructions from https://help.github.com/articles/generating-ssh-keys/.

This should download a directory named paraGSEA. To build paraGSEA, execute the following.

cd paraGSEA
make all
make install

This should succeed on most Linux systems because make is available by default. If this is not the case, you can obtain it by typing **sudo apt-get install make** on Ubuntu. Other Linux systems can also easily obtain the *make* tool by some simple commands. On Mac, you need to install *XCode*, which may take some time. First, you will need an Apple ID, then you will need to download it from the developer website of Apple https://developer.apple.com/xcode/downloads/. Then, you may need to follow the instructions shown on the following link to install the command line version of *make*. http://stackoverflow.com/q/10265742/1248687.

Calling make should create some executable files. Note that you need root authority to run **make install** command, then you can running the commands of paraGSEA in any path of this system. If you cannot, the application can be only used in paraGSEA/bin directory. To check that the building is successful, execute the following command.

./quick_search_serial

If you obtain the output shown below, then everything went fine and you are done with the build. If not, then something went wrong. In this case, you can explain how to reproduce the problem on https://github.com/ysycloud/paraGSEA/issues. Then, we will solve it for you as quick as possible.

Usage: quick_search_serial [options]

general options:

-n --topn: The first and last N GSEA records ordered by ES. [default 10]

input/output options:

-i --input: input file/a parsed profiles's file from pretreatment stage.

-s --sample: input file/a parsed sample sequence number file from pretreatment stage.

-r --reference: input a directory includes referenced files about genesymbols and cids.

2 paraGSEA basics

paraGSEA implements a MPI and OpenMP-Based parallel GSEA algorithm for multi-core or cluster architecture. But some pretreatment tasks for original data must use *1ktools*, that is an open-source tool with a variety of implementations. In our work, we use the *Matlab* version.

Therefore, make sure you have installed the common tools we listed below before you could use paraGSEA.

1. **Matlab R2009a and above**
2. **MPI**
3. **gcc compiler supports OpenMP**

There are mainly three parts of work in paraGSEA.

First, we implement GSEA approach in efficient parallel strategy with MPI and OpenMP to perform a quick search task, which needs users input a gene set and it will output the top N results after searching the profile data set by carrying out GSEA calculations. In this part, on the one hand, we reduced the computational overhead of standard procedure to calculate the Enrichment Score by pre-sorting, indexing and removing the prefix sum. On the other hand, we will take a global permutation method to wipe off the redundant overhead of estimation of significance level step.

Second, we expanded GSEA's application to quickly compare two gene profile sets to get an Enrichment Score matrix of every gene profile pairs. In this part, in addition to using the previous optimization strategies, our implementation also allows to generate a second level of parallelization by creating several threads per MPI process. The assignment of tasks to threads or processes is performed through a strict load balancing strategy, which leads to a better performance.

Third, we clustered the gene profile based on the Enrichment Score matrix which we can get by the second part. In this part, Enrichment Score is served as the metric to measure the similarity between two gene profiles. We implemented a general clustering algorithm like K-Mediods which is an improved version of K-Means. The algorithm can quickly converge and then output the corresponding results. Also, we improved algorithm and provided an implementation of *k*-mediods++, which is able to ensure the mutual distances between initial centers as far as possible to achieve better results.

3 Input formats and Pretreatment

The original input data stored in the HDF5 file format with a ‘gctx’ or ‘gct’ suffix. In order to use and analysis the data, we must use *1ktools*, which is an Open-Source project published in github, to parse it and extract the information we care about. You can browse https://github.com/cmap/l1ktools to find this project.

There is an example file ‘**modzs_n272x978.gctx**’ in ‘paraGSEA/data’ directory. It is our profiles data set. ‘**n272x978**’ means there are 272 profiles with 978 genes for each. In this file, every gene has a ‘*rid’* , which is corresponding to a gene name(symbol). Every profile has a ‘*cid*’, which identifies a set of experimental conditions to get this profile. By parsing the HDF5 file to get every part condition of these profiles, we provide user-friendly parsed method to allow user set their own conditions of profile they need.

In order to achieve this goal, we must generate some reference data to facilitate our main work. There is a *Matlab* script in ‘paraGSEA/matlab_for_parse’ directory named ‘**genReferenceforNewDataSet.m**’ to help us finish this work.

To use this script, we should first set the MATLAB path:

Enter the “pathtool” command, click “Add with Subfolders...”, and select the directory ‘paraGSEA/matlab_for_parse’. Or, if you cannot use *Matlab* by a visual way, you can just run ‘addpath('paraGSEA/matlab_for_parse')’ after setup the *Matlab* environment to add the directory path. Then executing the following command in *Matlab* environment.

```
datasource='../data/modzs_n272x978.gctx';
gene_symbol_rhd = 'pr_gene_symbol';
sample_conditions_chd = {'cell_id', 'pert_iname', 'pert_type', 'pert_itime', 'pert_idose'};
genReferenceforNewDataSet
```

When we get a new profile file keeps in correct format with a ‘gctx’ or ‘gct’ suffix, we can set its path in ‘**datasource**’ variable. Because different data set of LINCS may have different field names of sample conditions, you must make sure what it actually is in your data set and set them into ‘**gene_symbol_rhd**’ and ‘**sample_conditions_chd**’ variables. In order to know them, you can parse them first and see these field names in ‘**rhd**’ and ‘**chd**’ struct.

The following Matlab script can help you do these.

```
ds= parse_gctx('../data/modzs_n272x978.gctx');
ds.rhd
ds.chd
```

Most of time the gene symbol field named **'pr_gene_symbol'**, just like the example, where you need not to modify it. However, the sample conditions have a variety of field names in different LINCS data set. You must make sure them and set correct field names. There are five conditions you provided to support the user-friendly parsed method. Using ‘**modzs_n272x978.gctx**’ as an example, **'cell_id'** means the cell line,

'pert_iname' means perturbation name, **'pert_type'** means perturbation type, **'pert_itime'** means duration whose unit is usually 'hour', **'pert_idose'** means concentration. Also you must keep them in order like above shown. By the way, if the data set with a 'gct' suffix, you can use '**parse_gct**' to parse it, which is also provided in 'paraGSEA/matlab_for_parse/lib' directory.

Then, three files will generate in '../data/Reference' directory.

1. Gene_List.txt: all gene names of every profile in original order recorded in new data source file.

2. Samples_Condition.txt: treatment conditions of all profiles in original order recorded in new data source file.

3. Samples_RowByteOffset.txt: Bytes offset of every line in file2 in order to locate Specific line conditions directly without loading all file2 into memory.

After generating the reference data, we also provide two *Matlab* scripts to support user-friendly parsed method to allow user set their own conditions of profile they need and extract corresponding profiles to analysis.

1. **PreGSEA.m**
2. **paraPreGSEA.m**

For example, you can execute following script.

```
sample_conditions_chd={'cell_id', 'pert_iname', 'pert_type', 'pert_itime', 'pert_idose'};
file_input='../data/modzs_n272x978.gctx';
file_name='../data/data_for_test.txt';
file_name_cidnum='../data/data_for_test_cidnum.txt';
cell_id_set={'A549','MCF7','A375','A673','AGS'};
pert_set={'atorvastatin','vemurafenib','venlafaxine'};
pert_type_set = {'trt_cp'};
duration = 6;
concentration=10;
PreGSEA;
```

'**sample_conditions_chd**' is obviously needed to help find the fields' index and get the field values to judge whether this profile is fit to our conditions. Also, we can see '**file_input**' represents the original profile file, '**file_name**' represents needed profiles the script extracts, '**file_name_cidnum**' represents the sequence number of profiles the script extracts, '**cell_id_set**' represents the cell lines set where the profiles should be get from, '**pert_set**' represents the perturbations that should be used in experiments to get the profiles, '**pert_type_set**' represents the perturbation types that should be used in experiments to get the profiles, '**duration**' represents the time to carry out the experiments and '**concentration**' represents the concentration is supposed to be kept during the experiments.

If you not set these parameters, the path-relative parameters, such as ' **file_input**', '**file_name**' and ' **file_name_cidnum**', will have some default values in '**PreGSEA**'.

However, those may not fit your need or definitely correct. And the other condition parameters, such as '**cell_id_set**', '**pert_set**', '**pert_type_set**', '**duration**' and '**concentration**', will no longer be taken into account in the extracting process. Moreover, '**paraPreGSEA**' script can parse the original data in a more efficient way by a multi-thread method but you must make sure that you have a multi-core environment first. There is another parameter '**cores**' can be set to determine the parallel level. However, you must notice that the number of cores must be smaller than the actual core number in your system.

Moreover, because the results be splitted into several parts, we need finish some remedial work by *unix* command line to combine them into whole correct file. Fortunately, we provide two shell scripts to handle all the processes.

1. **example/runPreGSEAbyMatlab.sh**
2. **example/runparaPreGSEAbyMatlab.sh**

The only thing that users need to do is just set some parameters in these two scripts.

4 Quick search

Once we get the standard txt file which is parsed from the original input data, paraGSEA can read it quickly and then keep on subsequent calculations. As we mentioned above, Quick Search needs users input a gene set and it will output the top N results after searching the profile data set by carrying out GSEA calculations.

It is worth mentioning that there are several implementations in three versions. The MPI version can run on multiple nodes to handle larger amounts of data. Moreover, it supports parallel IO. The OpenMP implemented a more lightweight version of parallel computing, and there is no extra overhead of communication between nodes. Actually, there is no advantage of Serialized version as compared to the previous two, it is just for comparative analysis.

The Usage of three version is shown below.

Usage: quick_search_serial [options]
general options:
-n --topn: The first and last N GSEA records ordered by ES. [default 10]
input/output options:
-i --input: input file/a parsed profiles's file from pretreatment stage.
-s --sample: input file/a parsed sample sequence number file from pretreatment stage.
-r --reference: input a directory includes referenced files about genesymbols and cids.

Usage: quick_search_omp [options]
general options:
-t --thread: the number of threads. [default 1]"
-n --topn: The first and last N GSEA records ordered by ES. [default 10]

input/output options:

-i --input: input file/a parsed profiles's file from pretreatment stage.

-s --sample: input file/a parsed sample sequence number file from pretreatment stage.

-r --reference: input a directory includes referenced files about genesymbols and cids.

Usage: quick_search_mpi [options]

general options before command by MPI:

-n process_num : Total number of processes. [default 1]

-ppn pernum: the number of processes in each node. [default 1]

-hostfile hostfile: list the IP or Hostname of nodes. [default localhost]

general options:

-n --topn: The first and last N GSEA records ordered by ES. [default 10]

input/output options:

-i --input: input file/a parsed profiles's file from pretreatment stage.

-s --sample: input file/a parsed sample sequence number file from pretreatment stage.

-r --reference: input a directory includes referenced files about genesymbols and cids.

The Usages have been detailed enough. Only note that '*-i –input'* corresponding to '**file_name**', '*-s –sample'* corresponding to '**file_name_cidnum**' and '*-r –reference'* corresponding to '../data/Reference' directory we set in pretreatment stage.

Here is an example.

./quick_search_serial -i data/data_for_test.txt -s data/data_for_test_cidnum.txt -n 5 -r data/Reference

In principle, you can conduct following interactive process.

Profile Set is Loading...!

profilenum:272 genelen:978

loading IO and prework time: 0.0237 s

which way do you want to input the GeneSet(0 -> standard input , others -> file input):1

input the path of file that has GeneSet until 'exit'(each line has a Gene Symbol/name):

data/GeneSet.txt

printf the high level of TopN GSEA result:

NO.1 -> SampleConditions: cid:CPC006_SKLU1_6H:BRD-K56343971-001-02-3:10; cell_line: SKLU1; perturbation: vemurafenib; perturbation type: trt_cp; duration ES:0.315086 NES:2.540525 pv:0.0000000000

NO.2 -> SampleConditions: cid:LJP001_MCF10A_24H:BRD-K56343971-001-04-9:0.08; cell_line: MCF10A; perturbation: vemurafenib; perturbation type: trt_cp; dura ES:0.225345 NES:1.881119 pv:0.0010856895

NO.3 -> SampleConditions: cid:NMH001_NEU.KCL_6H.4H:BRD-K69726342-001-02-6:10; cell_line: NEU.KCL; perturbation: atorvastatin; perturbation type: trt_cp; dur ES:0.223448

NES:1.843819 pv:0.0014393966

NO.4 -> SampleConditions: cid:LJP001_BT20_24H:BRD-K56343971-001-04-9:0.4; cell_line: BT20; perturbation: vemurafenib; perturbation type: trt_cp; duratio ES:0.221078 NES:1.843653 pv:0.0016556464

NO.5 -> SampleConditions: cid:LJP001_MCF7_24H:BRD-K56343971-001-04-9:2; cell_line: MCF7; perturbation: vemurafenib; perturbation type: trt_cp; duration: ES:0.216035 NES:1.790939 pv:0.0023914223

printf the low level of TopN GSEA result:

NO.1 -> SampleConditions: cid:LJP001_HS578T_6H:BRD-K56343971-001-04-9:0.4; cell_line: HS578T; perturbation: vemurafenib; perturbation type: trt_cp; durati ES:-0.229569 NES:-1.821155 pv:-0.0023933623

NO.2 -> SampleConditions: cid:CPC006_HEC108_6H:BRD-U88459701-000-01-8:10; cell_line: HEC108; perturbation: atorvastatin; perturbation type: trt_cp; duratio ES:-0.219655 NES:-1.762991 pv:-0.0029483190

NO.3 -> SampleConditions: cid:CPC006_SNUC5_6H:BRD-K56343971-001-02-3:10; cell_line: SNUC5; perturbation: vemurafenib; perturbation type: trt_cp; duration ES:-0.202629 NES:-1.648118 pv:-0.0058889213

NO.4 -> SampleConditions: cid:CPC004_A375_6H:BRD-A51714012-001-03-1:10; cell_line: A375; perturbation: venlafaxine; perturbation type: trt_cp; duration: ES:-0.194224 NES:-1.559712 pv:-0.0068219395

NO.5 -> SampleConditions: cid:CPC006_SNGM_6H:BRD-U88459701-000-01-8:10; cell_line: SNGM; perturbation: atorvastatin; perturbation type: trt_cp; duration: ES:-0.192112 NES:-1.541771 pv:-0.0081038799

finish GSEA time: 0.1511 s

input the path of file that has GeneSet until 'exit'(each line has a Gene Symbol/name):

exit

Note that you can choose input a gene set directly or input a file path where there is a gene set. Second way may be more convenient such as the example shows.

The other two versions are totally same interactive processes, where we no longer give an example.

5 Compare profiles

If there are two txt files we get from pretreatment stage serve as the input, we can quickly compare them to get an Enrichment Score matrix of every gene profile pairs. Our implementation will allow to generate a second level of parallelization by creating several threads per MPI process. The assignment of tasks to threads or processes is performed through a strict load balancing strategy. There are still three versions of implementation with different data split strategies, where they are no communication, point to point communication and collective communication respectively.

The Usage of three versions is shown below.

Usage: ES_Matrix_ompi_nocom [options]

general options before command by MPI:

-n process_num : Total number of processes. [default 1]

-ppn pernum: the number of processes in each node. [default 1]

-hostfile hostfile: list the IP or Hostname of nodes. [default localhost]

general options:

-t --thread: the number of threads in per process_num. [default 1]

-l --siglen: the length of Gene Expression Signature. [default 50]

input/output options:

-1 --input1: a parsed profiles's file from pretreatment stage.

-2 --input2: another parsed profiles's file from pretreatment stage.

-o --output: output file, distributed in every nodes ,with ES Matrix.

Usage: ES_Matrix_ompi_p2p [options]

general options before command by MPI:

-n process_num : Total number of processes. [default 1]

-ppn pernum: the number of processes in each node. [default 1]

-hostfile hostfile: list the IP or Hostname of nodes. [default localhost]

general options:

-t --thread: the number of threads in per process_num. [default 1]

-l --siglen: the length of Gene Expression Signature. [default 50]

input/output options:

-1 --input1: a parsed profiles's file from pretreatment stage.

-2 --input2: another parsed profiles's file from pretreatment stage.

-o --output: output file, distributed in every nodes ,with ES Matrix.

Usage: ES_Matrix_ompi_cocom [options]

general options before command by MPI:

-n process_num : Total number of processes. [default 1]

-ppn pernum: the number of processes in each node. [default 1]

-hostfile hostfile: list the IP or Hostname of nodes. [default localhost]

general options:

-t --thread: the number of threads in per process_num. [default 1]

-l --siglen: the length of Gene Expression Signature. [default 50]

input/output options:

-1 --input1: a parsed profiles's file from pretreatment stage.

-2 --input2: another parsed profiles's file from pretreatment stage.

-o --output: output file, distributed in every nodes ,with ES Matrix.

The Usages of three versions are totally same and detailed enough. Here is an example of ‘nocom’ method.

mpirun -n 2 -ppn 2 -hostfile example/hostfile ES_Matrix_ompi_nocom -t 4 -l 50 -1 data/data_for_test.txt -2 data/data_for_test.txt -o data/ES_Matrix_test

In principle, It may produce the following output:

Profile Set is Loading...!

loading IO and prework time in no communication way: 0.2017 s

Paral compute the ES_Matrix is Starting...!

Paral compute the ES_Matrix time : 2.7954 s

Writing file is Starting...!

Write Result spent: 0.0390 s

There is no more need of you to input anything in command line. However, Only note that the ES_Matrix will be written to file '**data/ES_Matrix_test_*.txt**' in every processes in a distributed way. The other two versions are almost same output, where we no longer give an example.

6 Clustering profiles

In this part, the results of 'Compare profiles' are served as input. Then we cluster the gene profiles based on the Enrichment Score matrix which can be seemed as the similarity Matrix of gene profiles. The implementation also supports a second level of parallelization. But we should note that input matrix should include the same identity of rows and columns, which means the last part program is supposed to use same two file as its input. Only in this way we can get the similarity of each profile pair.

We implemented a general clustering algorithm like *k*-mediods which is an improved version of *k*-means. The difference lies in the way of how to find new clustering centers in each iteration. Instead of using the average vector of each cluster as the new clustering center, we use the profile, which has the greatest average similarity of other profiles in the same cluster, as the new clustering center.

In order to solve the randomized problem of initial clustering centers to achieve a better result and better convergence speed, we improved algorithm again and provided an implementation of *k*-mediods++, which is able to ensure that the mutual distance between initial cluster centers as far as possible. Nevertheless, it will spend more time to determine the initial clustering centers.

The Usage of three versions is shown below.

Usage: Cluster_KMediods_ompi [options]
general options before command by MPI:
-n process_num : Total number of processes. [default 1]
-ppn pernum: the number of processes in each node. [default 1]
-hostfile hostfile: list the IP or Hostname of nodes. [default localhost]
general options:
-t --thread: the number of threads in per process_num. [default 1]
-c --cluster: the number of clusters we want to get. [default 5]
input/output options:
-i --input1: distributed ES_Matrix file we get from stage 2(Compare Profiles)
-o --output: output class flags file of every profiles in root node
-s --sample: input file/a parsed sample sequence number file from pretreatment stage.

-r --reference: input a directory includes referenced files about genesymbols and cids.

Usage: Cluster_KMediods++_ompi [options]

general options before command by MPI:

-n process_num : Total number of processes. [default 1]

-ppn pernum: the number of processes in each node. [default 1]

-hostfile hostfile: list the IP or Hostname of nodes. [default localhost]

general options:

-t --thread: the number of threads in per process_num. [default 1]

-c --cluster: the number of clusters we want to get. [default 5]

input/output options:

-i --input1: distributed ES_Matrix file we get from stage 2(Compare Profiles)

-o --output: output class flags file of every profiles in root node

-s --sample: input file/a parsed sample sequence number file from pretreatment stage.

-r --reference: input a directory includes referenced files about genesymbols and cids.

The Usages of two versions are totally same and detailed enough. Here is an example of '*k*-mediods' method.

mpirun -n 2 -ppn 2 -hostfile example/hostfile Cluster_KMediods_ompi -t 4 -c 8 -i data/ES_Matrix_test -o data/Cluster_result_test.txt -s data/data_for_test_cidnum.txt -r data/Reference

In principle, It may produce the following output:

Matrix is Loading...!

loading IO and prework time : 0.0214 s

Paral KMediods compute the Cluster Centers is Starting...!

Init cluster centers is:

127 216 145 245 71 29 265 35

1th iteraction cluster_center_new is:

174 30 136 60 86 58 265 38

2th iteraction cluster_center_new is:

18 198 68 120 0 136 212 265

3th iteraction cluster_center_new is:

63 30 42 248 136 88 140 265

4th iteraction cluster_center_new is:

18 30 63 122 136 208 240 265

5th iteraction cluster_center_new is:

56 60 63 220 136 46 240 265

6th iteraction cluster_center_new is:

20 56 0 200 136 248 42 265

7th iteraction cluster_center_new is:

63 128 30 112 220 200 248 265

8th iteraction cluster_center_new is:

38 56 66 174 0 192 248 265

9th iteraction cluster_center_new is:

63 76 112 136 174 220 32 265

10th iteraction cluster_center_new is:

231 200 76 256 136 174 248 265

……

Paral KMediods compute the Cluster Centers Spent: 0.2829 s.

Also, there is also no more need of you to input anything in command line. However, the results will be written to file '**data/Cluster_result_test.txt**' in root process in the format shown below.

cluster 1 :

cid:CPC006_A549_6H:BRD-U88459701-000-01-8:10; cell_line: A549; perturbation: atorvastatin; perturbation type: trt_cp; duration: 6 h; concentration: 10 ?M

cid:CPC020_A375_6H:BRD-A82307304-001-01-8:10; cell_line: A375; perturbation: atorvastatin; perturbation type: trt_cp; duration: 6 h; concentration: 10 ?M

……

cluster 2 :

cid:CPC006_NCIH508_6H:BRD-K56343971-001-02-3:10; cell_line: NCIH508; perturbation: vemurafenib; perturbation type: trt_cp; duration: 6 h; concentration: 10 ?M

cid:CPC006_PC3_6H:BRD-K56343971-001-02-3:10; cell_line: PC3; perturbation: vemurafenib; perturbation type: trt_cp; duration: 6 h; concentration: 10 ?M

……

cluster 3 :

cid:LJP001_MDAMB231_6H:BRD-K56343971-001-04-9:0.4; cell_line: MDAMB231; perturbation: vemurafenib; perturbation type: trt_cp; duration: 6 h; concentration: 500 nM

……

cluster 4 :

……

cluster 5 :

……

cluster 6 :

……

cluster 7 :

……

cluster 8 :

……

Through the results, we can clearly see there are what profiles in each cluster. Another version are almost same output, where we no longer give an example.

Note that input matrix should include the same identity of rows and columns, which means 'Compare Profiles' stage is supposed to use same two files as its inputs ('*-1 --input1*', '*-2 –input2*'), and the first three parameters ('*-n process_num*', '*-ppn pernum*',

‘*-hostfile hostfile*’) should not be changed compare with ‘Compare Profiles’ stage, so that the program can find all the distributed ES_Matrix file in all nodes.