Published online 31 July 2017

Nucleic Acids Research, 2017, Vol. 45, No. 17 ***e155***
doi: 10.1093/nar/gkx679

paraGSEA: a scalable approach for large-scale gene expression profiling

Shaoliang Peng[1,2,†], Shunyun Yang[2,†], Xiaochen Bo[3,*] and Fei Li[3,*]

[1]College of Computer Science and Electronic Engineering & National Supercomputer Centre in Changsha, Hunan University, Changsha 410082, China, [2]School of Computer Science, National University of Defense Technology, Changsha 410073, China and [3]Department of biotechnology, Beijing Institute of Radiation Medicine, Beijing 100850, China

Received July 02, 2017; Editorial Decision July 21, 2017; Accepted July 27, 2017

ABSTRACT

More studies have been conducted using gene expression similarity to identify functional connections among genes, diseases and drugs. Gene Set Enrichment Analysis (GSEA) is a powerful analytical method for interpreting gene expression data. However, due to its enormous computational overhead in the estimation of significance level step and multiple hypothesis testing step, the computation scalability and efficiency are poor on large-scale datasets. We proposed *paraGSEA* for efficient large-scale transcriptome data analysis. By optimization, the overall time complexity of paraGSEA is reduced from *O*(*mn*) to *O*(*m*+*n*), where *m* is the length of the gene sets and *n* is the length of the gene expression profiles, which contributes more than 100-fold increase in performance compared with other popular GSEA implementations such as GSEA-P, SAM-GS and GSEA2. By further parallelization, a near-linear speed-up is gained on both workstations and clusters in an efficient manner with high scalability and performance on large-scale datasets. The analysis time of whole LINCS phase I dataset (GSE92742) was reduced to nearly half hour on a 1000 node cluster on Tianhe-2, or within 120 hours on a 96-core workstation. The source code of paraGSEA is licensed under the GPLv3 and available at http://github.com/ysycloud/paraGSEA.

INTRODUCTION

Genome-wide expression analysis with high-throughput technologies has become an indispensable tool in comparative transcriptomics (1,2). Affymetrix microarray platform, Luminex-based multiplex RNA assays and high-resolution RNA sequencing (RNA-Seq) result in unprecedented accumulations of massive gene expression data from diverse experimental conditions. Gene expression profile archives (i.e. LINCS (3), GEO (4), ArrayExpress (5) and TCGA (6)) represent the gene expression conditions of many diseases, tissues, pathogen invasion, chemical compound and gene mutation status.

The resource of large-scale transcriptomic datasets makes it possible to mine hidden biological knowledge systematically and foster the big data paradigm of biomedical research. Given the biological phenotype as an annotation, relevant expression profiles as reference sources can be highly informative if scalable, efficient mining and query methods are properly applied (7–9). For example, the Library of Integrated Network-based Cellular Signatures (LINCS) project phase I released more than 1.3 million whole genome expression profiles, including more than 3000 gene silencing, 5000 chemical molecules and 15 cell types. In reference to the LINCS database, the biological effect of any new chemical molecules can be evaluated by comparing gene expression profiles against the LINCS dataset and then identifying the entry with most similar LINCS profile.

There exist several distributed parallelization approaches to accelerate large-scale biological data processing and analysis (10–14). However, an efficient analytical approach designed for large-scale transcriptome datasets is still not available.

L1000CDS^2 (15) is a web-based search engine application using correlation coefficient between an input signature vector and the LINCS L1000 data to prioritize small molecules and drugs to either reverse or mimic observed changes in gene expression. With extended usage, this tool can also conveniently predict the most likely targets for each small molecule profiled by the L1000 assay. However, the L1000CDS^2 tool simply uses the overlap or cosine similarity as the metric to estimate the relationship between input signature vector and the LINCS-L1000 data, which may not

*To whom correspondence should be addressed. Tel: +86 10 6693 2251; Fax: +86 10 6693 2251; Email: pittacus@gmail.com
Correspondence may also be addressed to Xiaochen Bo. Tel: +86 10 6693 2251; Fax: +86 10 6693 2251; Email: boxiaoc@163.com
†These authors contributed equally to the paper as first authors.

lead to reliable results with biological meaning in a statistically rigorous manner.

GSEA (16,17) is a widely accepted knowledge-based metric for interpreting gene expression profiles, mainly using a rank-based Kolmogorov–Smirnov statistic, and has several implementations from different perspectives such as GSEA-P (16,17), SAM-GS (18,19) or GSEA2 (20,21). However, due to performance and scalability, these tools have difficulty meeting the time requirement of computationally intensive analysis when confronting large-scale expression profiles.

To overcome this bottleneck in large-scale GSEA application, we proposed paraGSEA, a fast parallel implementation of GSEA for large-scale transcription profiles to accelerate querying, all-pairs comparison and global clustering analysis on transcriptomic datasets. We implemented the GSEA algorithm in an efficient parallel strategy with MPI and OpenMP first. The computational overhead of standard enrichment score (ES) calculation procedure by presorting, filtering, inverted indexing and prefix sum removing was reduced. Meanwhile, a global permutation method was used to eliminate the redundant overhead in estimation of the significance level step. Second, we expanded GSEA's application to quickly compare two groups of gene profiles to get an ES matrix of all gene expression profile pairs. This implementation can be effectively applied to the Connectivity Map project (22,23), which aims to discover the functional connections among diseases, genetic perturbation and drug action. In this part, in addition to the previous optimization strategies, our implementation allows generating a second level of parallelization by creating multiple threads per MPI process. Through reasonable data partitioning and efficient global communication mechanism, we can achieve a better load balance, maximal data locality and minimal disk I/O. Third, we clustered the gene profile based on the previous ES matrix to demonstrate how to perform a landscape analysis on the whole LINCS dataset phase I in a scalable and efficient way. In this part, ES serves as a metric to measure the similarity between two gene expression profiles. We implemented a general clustering algorithm like *k*-mediods (24), which is an improved version of *k*-means algorithm. It is more robust to noise and outliers than *k*-means because it minimizes the sum of pairwise dissimilarities instead of the sum of squared Euclidean distances. Also, the algorithm converges and output corresponding results quickly.

In general, paraGSEA optimizes and implements a scalable parallel algorithm to accelerate GSEA for large-scale datasets and exploits the computational power of current high performance workstations and clusters by employing both MPI and OpenMP. As a highly reliable and scalable tool with remarkably low execution time, paraGSEA is greatly suitable for large-scale transcriptome analysis of gene expression profiles.

MATERIALS AND METHODS

With the 'para' standing for parallel, paraGSEA follows a classical expression profile analysis pipeline with Gene Set Enrichment Analysis (Figure 1A). This flowchart shows the design of paraGSEA framework, consisting of four layers. Layer 1 represents data pre-treatment. Original gene expression profiles from LINCS are parsed and pre-sorted by L1000 Analysis Tools, which is an open source project for parsing LINCS datasets published on GitHub. Given different treatment conditions such as cell lines, perturbations, concentration and duration of treatment, different output file parsed from profiles will be generated and taken as input to layer 2 along with the user-provided gene set. Layer 2 represents GSEA core operation. This part contain two components: Quick Search for computing most similar and dissimilar gene expression profiles to input gene set, and Comparing Profiles by pairwise ES matrix computation in parallel. Layer 3 represents the first level result. Quick search module output Top-N results consist of ES, *P*-value and normalized ES. At the same time, the ES matrix is exported by Comparing Profiles module in a distributed way. Layer 4 represents gene expression profiles clustering. Two algorithms are implemented to cluster gene expression profiles based on similarity matrix. The module aggregates the final clustering result, including clustering labels and corresponding profile list. More details will be described in the following sections.

LINCS data

The original input data come from LINCS, which evolved from the Connectivity Map project. The data are stored in the HDF5 file format with a gctx or gct extension. To use and analyze the data, we used l1ktools to parse them and extract the information we needed. We can set different treatment conditions as data filters such as cell lines, perturbations, concentration and duration of treatment, and then we will get different parsed text files with profiles conforming to the treatment conditions we set. In this process, when we found the correct profiles, we first numbered every gene of them starting with one and then ordered them according to their differential expression. Finally, we will write out these ranked lists to a text file, which can be easily used as an input to subsequent calculations.

The corresponding column ID list of each profile will be pre-sorted for efficiency, which identifies the profile sequence number in the original HDF5 file. The pre-sorting operation is necessary because once written, the file can be used multiple times to avoid repeating work. The version of l1ktools we used implemented with MATLAB can simply and efficiently support this work. Meanwhile, gene name list file is also generated as an annotation, in which each line is a gene name corresponding to the gene sequence of every profile in LINCS dataset. When users input a gene name, we can get its sequence number by searching this file, which is only needed in subsequent calculations.

It should also be noted that the sample condition list file need to be pre-processed by a two-level indexing technique for fast filtering. Each line in sample condition list is a sample condition description, which includes cell lines, perturbations, concentration and duration of treatment. Each description represents a profile in LINCS dataset and keeps the same order. When we get a sequence number of a profile by calculating, we can also get its treatment condition by searching this file. By two-level indexing, we are able to quickly locate specific line and get the treatment condition without loading all the file into memory.

GSEA optimization

Once standard data are parsed from the original input HDF5 file, paraGSEA will read it and keep on making subsequent calculations. There are several implementations in three versions: serialized version, MPI version and OpenMP version. The MPI version can run on multiple compute nodes to handle larger amounts of data. Moreover, it supports parallel I/O for access dataset on disk simultaneously. The OpenMP implemented a more lightweight version of parallel computing, and there is no extra overhead of communication between compute nodes. The serialized version is provided for basic computational environment without MPI or OpenMP installation. If memory is big enough to store all data in a single shared-memory compute node, one can use the multi-threaded implementation with OpenMP. It can be more efficient than the MPI version because thread model is more lightweight than the process model in parallel programming. However, in most of the time, the data are too big to store in a single node or a cluster with many cores need to use, we need to use the MPI multi-process implementation correspondingly. In fact, if communication between nodes is relatively stable and unblocked, performance will not change dramatically because the efficient parallel MPI I/O routines allow us to parallelize the reading of the input dataset, which is not provided with OpenMP.

For each pair of gene expression profiles, we compute the ES using a rank-based Kolmogorov–Smirnov statistic (16,17). Two optimization techniques have been applied to the standard procedure to calculate the ES. First, we calculate a hitting vector, which marks the location of each gene in a gene set that appears in the profile. A straightforward resolution is a traverse gene set and profile alternately. Its cost will be a product of the size of the gene set and profile. Instead of this method, we first established an inverted index by scanning the profile. It will be completed very quickly because in the pre-treatment stage, we used unique figures to represent the profile. Then we can obtain the hitting vector by just scanning the gene set one time. Therefore, the final overhead depends on the sum of both sizes. In general, this optimization reduces the time complexity of iteratively scanning from $O(mn)$ to $O(m+n)$ by creating a hit locus index only once, where m is the length of gene sets and n is the length of gene expression profiles.

The second optimization is prefix sum removing. This step aims to find the maximum deviation between the hitting vector and the corresponding missing vector. Actually, the maximum deviation will only appear around the hitting points. Through a detailed analysis of conditions in these points, we can also get the correct result. Hence, Instead of standard GSEA processing with time complexity $O(n)$, paraGSEA reduced it to $O(\log m)+O(m) = O(m)$, The extra $O(\log m)$ is a sorting operation on the hit locus. Obviously, in practical application, the size of the profile will be much larger than a gene set. Therefore, this optimization strategy is very worthwhile.

The whole process of standard ES calculation and our optimization are summarized in the pseudocode shown in Algorithms 1 and 2.

Algorithm 1. Old Algorithm
Calculate the Enrichment Score

```
1: Input: profile, gene set
2: Output: es
3: Variables: max, index, tmp, siglen, len
4: Containers: isgs, scorehit, scoremiss
5: isgs←initial zero vector
6: siglen←length of geneset
7: len←length of profile
7: for gene g1 in profile do
8:    for gene g2 in geneset do
9:       if profile[g1] is equal to geneset[g2] then
10:         isgs[g1]←1
11:      end if
12:   end for
13:end for
14:scorehit[0]←isgs[0]
15:scoremiss[0]←1-isgs[0]
16:max ← absolute value of scorehit[0]/siglen-
scoremiss[0]/(len-siglen)
17:index←0
18:for gene g in profile do
19:   scorehit[g]←isgs[g]+scorehit[g-1]
20:   scoremiss[g]←(1-isgs[g])+scoremiss[g-1]
21:         tmp ← absolute value of
scorehit[g]/siglen-scoremiss[g] /(len-siglen)
22:   if tmp is bigger than max then
23:      max←tmp
24:      index←g
25:   end if
26:end for
27:es ←
scorehit[index]/siglen-scoremiss[index]/(len-siglen)
```

Algorithm 2. New Algorithm
Calculate the Enrichment Score

```
1: Input: profile, gene set
2: Output: es
3: Variables: siglen, len
4: Containers: isgs, index
5: siglen←length of geneset
6: len←length of profile
7: for gene g in profile do
8:    index[profile[g]]←g
9: end for
10:for gene g in geneset do
11:   isgs[g]←index[geneset[g]]
12: end for
13:sort isgs ascendingly
14:for gene g in geneset do
15:   if g is the first gene in geneset then
16:      if isgs[g] is not the first gene in profile then
17:         if prev gene before g reach max absolute
value of current es then
18:            es←isgs[g]/(1-siglen)
19:         end if
20:      end if
21:      if the gene after g is not isgs[g+1] and g reach
max absolute value of current es then
22:         es←(g+1)/siglen-(isgs[g]-g)/(len-siglen)
23:      end if
24:   else if g is the last gene in geneset then
25:      if g reach max absolute value of current es then
26:         es←(g+1)/siglen-(isgs[g]-g)/(len-siglen)
27:      end if
28:      if the gene before g is not isgs[g-1] and reach
max absolute value of current es then
29:         es←g/siglen-(isgs[g]-g)/(len-siglen)
30:      end if
31:   else
32:      if the gene after g is not isgs[g+1] and g reach
max absolute value of current es then
33:         es←(g+1)/siglen-(isgs[g]-g)/(len-siglen)
34:      end if
35:      if the gene before g is not isgs[g-1] and reach
max absolute value of current es then
36:         es←g/siglen-(isgs[g]-g)/(len-siglen)
37:      end if
38:   end if
40:end for
```

Estimation of the significance level of ES is a crucial and time-consuming step of the GSEA approach. The goal is to measure the reliability of the ES by statistical methods. However, the biggest challenge lies in the permutation operation. Actually, in the statistical domain, we usually need to carry out the permutation operation thousands of times before we can analyze their statistical meaning. So, if we want to get a statistical index like *P*-value, we need to calculate new ESs of random permutated profile thousands of times. This will directly expand the computational overhead thousands of times, which is inefficient and unaffordable for large-scale analysis.

To solve this problem, we adopt a global permutation and random sampling strategy to wipe out the redundant overhead and manage computing expenses. In fact, permutation simply means changing the hitting position in the hitting vector. For the same gene set, it is unnecessary to generate all possible permutations for each expression profile. Instead, we just need to do permutation many times and get their ESs to generate the Global Enrichment Scores Pool (GESPool) only at the beginning. Once we get ES of each gene set and profile pair, we only need to extract more than thousands of results from GESPool to carry out the following statistical analysis. That means, it results in a thousand-fold decrease in time overhead. The results of this strategy are consistent with the original strategy within the error allowable range.

All-pairs comparison

Given two text files from the pre-treatment stage as the input, we can quickly compare them to get an ES matrix of every gene profile pair. For each profile–profile pair, we compute an ES for the probe sets representing the up- or downregulated signature genes separately using a rank-

based Kolmogorov–Smirnov statistic (22). The scores from up- and downregulated genes are combined into a single connectivity score for each profile pair combination.

Besides the optimization strategies we have described, our implementation will also allow generating 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 depends on the way the data are partitioned to give each process equal amounts of gene expression profiles. Typically, for many profile–profile pairs, the list of first profiles is directly distributed among the processes, so that for each process, the number of profiles will be the same or have a difference no greater than one. However, we must make sure that each process holds whole second profiles. Otherwise, we will not be able to get the ESs of every pair. Therefore, the problem is how we read and distribute second profiles to achieve better performance. We offer three strategies for solving this problem.

As a first strategy, we duplicates second profiles for all nodes in a cluster, so that each process can directly read them locally. In this way, there will be no communication among nodes but with a huge IO overhead. Instead, as a second strategy, we can assign a root process to read entire second profiles and then send it to other processes. In this method, IO overhead is greatly reduced, and there will be no more duplications, but it will be followed by a mass communication. Actually, due to the parallelization of MPI IO in the first strategy, the second strategy often leads to worse performance. Finally, a compromise solution is to make each process take a different part of second profiles, then gather and redistribute them to each other. There will be less IO overhead and less communication. Due to a relatively efficient global communication mechanism is provided by MPI, the final strategy often achieves better performance.

After the completion of each calculation, processes will write the results to their own nodes in parallel. The overall of this part is shown in Figure 1B.

Fast clustering

Clustering is a kind of very common and useful method in data mining to analyze large-scale biological data (25,26). Also, some tools provide a convenient web interface for integrated clustering of multi-dimensional biological data (27) or extract some knowledge-based characterizations of biological data to facilitate the clustering process (28). However, a more efficient clustering algorithm is needed for large-scale transcriptome analysis.

We implemented a new scalable parallel clustering algorithm like *k*-mediods, which clusters the gene profiles based on the ES matrix as similarity matrix, and also supports a second level of parallelization we have already described. The difference lies in the way to find new clustering centers in each iteration. It is more robust to noise and outliers compared to *k*-means because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances. Hence, in our implementation, instead of using the average vector of each cluster as the new clustering center, we use a profile with the greatest average similarity of other profiles in the same cluster as the new clustering center. This strategy offers two advantages. First, this method has computational efficiency. Since no new profiles are introduced, there is no need to recalculate ESs. Second, this method can reduce the effects of noise points and improve the compact degree of clusters for a better clustering result.

The implementation of this part is shown in Figure 1C. We can see that in each iteration, every process just holds several lines of ES matrix by parallel IO and strict load balance strategy, which provides the basis for the efficient second level of parallelization in Steps 2 and 3. Only three small vectors will be communicated among the processes, the impact on performance is extremely limited. Therefore, we can expect that this implementation is a very efficient parallel clustering strategy.

However, like the shortcomings of *k*-means, *k*-mediods also need to determine the initial cluster centers artificially. Different initial cluster centers may lead to a completely different clustering result. To solve this problem, we improved the algorithm again and provided an implementation of *k*-mediods++ to ensure that the mutual distance between initial cluster centers is as far as possible. Nevertheless, it will take more time to determine the initial clustering centers. Therefore, one should decide which implementation is needed.

Once the clustering centers do not change, the program outputs the final clustering result with clustering label and corresponding profile list to the root node. Actually, the algorithm converges quickly and then outputs corresponding result.

Benchmark

To evaluate the performance, we first benchmarked paraGSEA against current GSEA implementations: GSEA-P-R.1.0 (16,17), pygsa-1.1, which is a Python version of SAM-GS (18,19), and gsea2, which is a MATLAB version of GSEA2 (20,21). Because these software do not have an efficient parallel behavior to support multi-node cluster architecture, all the tests were performed in serial mode with a single-node workstation.

All the tests were performed on both: a workstation and a cluster. The workstation used in the test is a 96 core high performance workstation with 8 Intel® Xeon® E7–8850@2.3GHz*12 v2 processors, 1TB memory and CentOS release 6.5 installed. The cluster is on Tianhe-2 supercomputer consists of 16 000 compute nodes, each of which equips two Intel® Xeon® E5–2692@2.2GHz*12 v2 processors and three Intel® Xeon Phi™ 31S1P coprocessors and remains 64 GB memory, 32, 000 Intel® Xeon® processors and 48 000 Intel® Xeon Phi™ coprocessors in total.

The test dataset is LINCS phase I dataset (GSE92742) with 1 319 138 transcriptome profiles. Some of these implementations have their own test datasets but do not have the same scale to enable comparison. To ensure a fair experiment, we set up a benchmark on an artificial dataset, which has the same scale with LINCS to keep 978 genes of each profile. We set the permutation at 1000 times or none at all (no statistical analysis), and different number of profiles respectively to carry out this benchmark. Note that when we

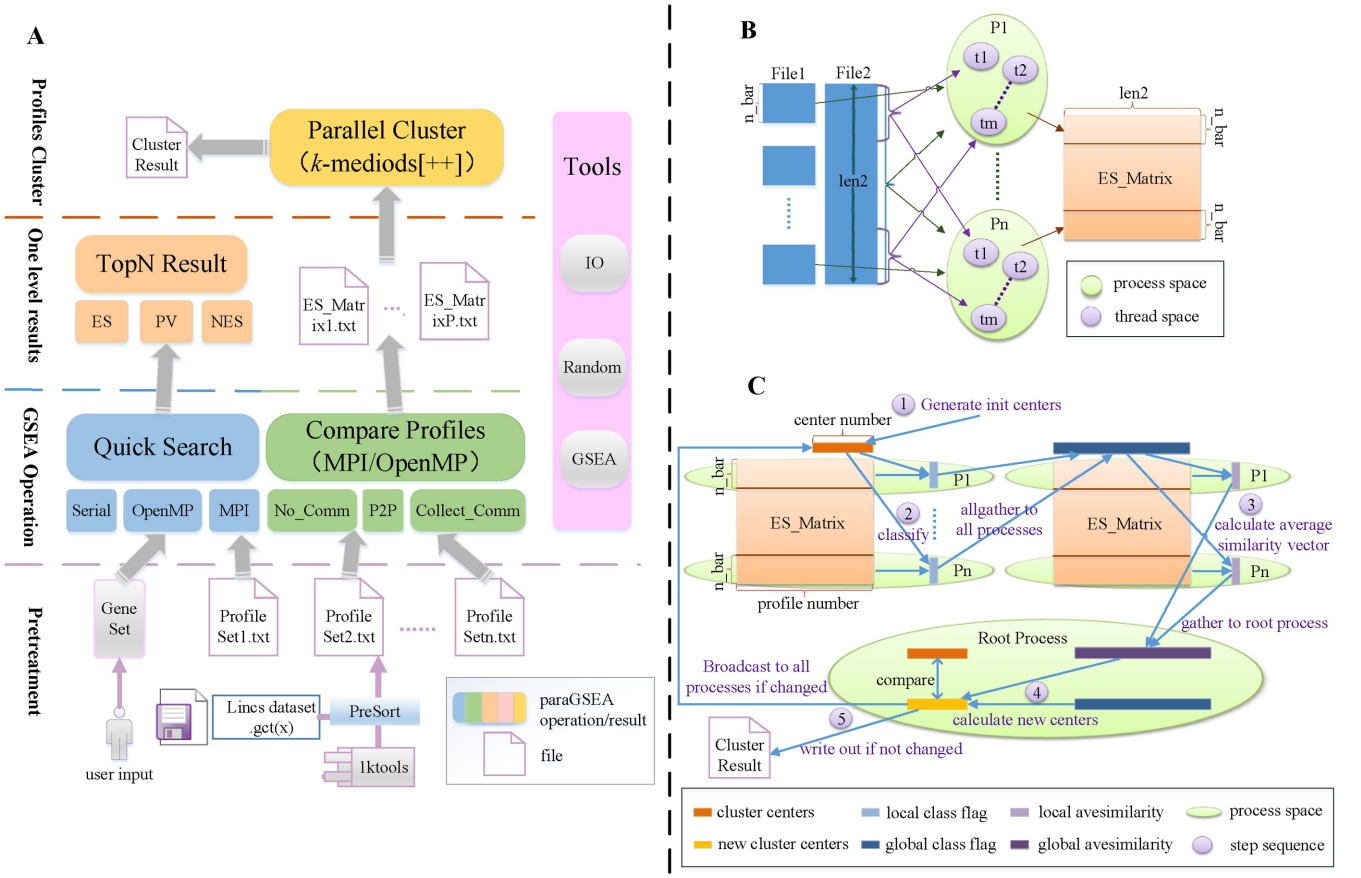

Figure 1. paraGSEA technical diagrams. (**A**) The paraGSEA pipeline. Schematic outline of paraGSEA mainly illustrates input/output data files, data flow and algorithm steps. The original inputs are expression profile files from LINCS in the HDF5 file format with a gctx or gct extension. The core modules consist of Quick Search, Compare Profiles and Parallel Cluster. The first two can be performed independently, but the third module depends on the results of the Compare Profiles module. Only the first module prints the results to the command line window. The other two modules all write out the results to external text files, because the result is too huge to show in the command line window. (**B**) Data partitioning and output strategies in comparison to the profiles module. Profiles are divided into several nodes, and each MPI process is in charge of different nodes. All processes read the input file in a parallel manner with efficient MPI I/O routines. After each reading stage is completed, the processes communicate and exchange data to ensure each process holds the entire second file. Then tasks will be divided equally between several threads in all processes, and similarity matrix calculation can be carried out in a parallel manner with a strict load balancing strategy. Finally, each process writes a part of the ES matrix into an output text file. (**C**) The parallel *k*-mediods' clustering process in one iteration. Schematic outline of parallel *k*-mediods in one iteration includes five steps. Step 1: generating initial centers. The algorithm first needs to generate some clustering centers in a specified number to continue, which is a simple random number generation process but must be carried out in one process and then broadcast to all processes to make sure that every process holds the same clustering centers. Step 2: classifying all profiles. This step is a second level of parallelization. Every process just holds several lines of the ES matrix, in which their scale is substantially equal, by reading the results of the Comparing Profiles step, and they can classify a part of profiles to get a local class flag vector respectively by multi-threads method using OpenMP. Then we gather all local results to form a global class flag vector to all processes. Step 3: calculating the average similarity vector. This step is also a second level of parallelization, which is very similar to Step 2, but calculates and forms a global average similarity vector to root process. Step 4: calculating new clustering centers. This step is only carried out on root process in a serial manner. Through comparing class flag vector and average similarity vector, we can find these profiles with the greatest average similarity of other profiles in the same cluster. Step 5: write out or move on. If the new centers are different than the old centers, they will replace the old centers and move on to the next iteration. Otherwise, the algorithm will stop and write out the cluster result.

change the number of profiles, the length of the gene set remains 50.

RESULTS

The result of benchmark with different numbers of profiles is shown in Figure 2. The paraGSEA implementation offers significant advantages in performance of every condition over others. We can see paraGSEA is up to two orders of magnitude faster than original GSEA implementations even on a single core CPU. There are some reasons for this phenomenon. First, because of the global permutation and random sampling strategy, the permutation of 1000 times no longer has a significant impact on computational performance for paraGSEA. In contrast, the other three implementations take approximately a hundred times longer to complete GSEA calculation tasks, which includes statistical analysis. Second, through various optimizations to improve the performance of ES calculation, such as establishing an inverted index or removing the prefix sum, the overall time complexity is reduced from $O(mn)$ to $O(m+n)$, where m is the length of gene sets and n is the length of gene expression profiles. Since m is much smaller than n, n may determine their performance to a greater extent. Because of the

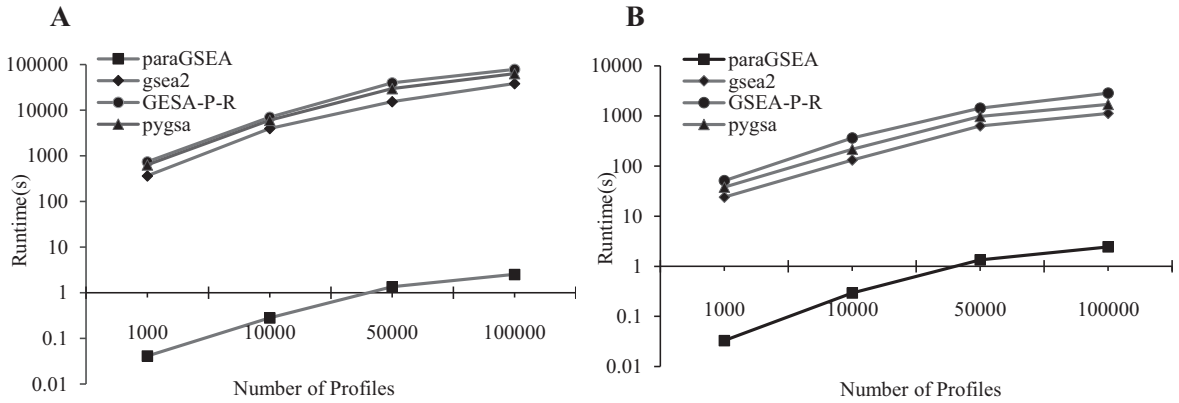

Figure 2. Runtime of four GSEA implementations. (**A**) Runtime of four GSEA implementations with 1000 times permutations, 50 gene sets versus different numbers of expression profiles. (**B**) Runtime of four GSEA implementations with no permutations and 50 gene sets versus different numbers of expression profiles.

fixed length of profiles in LINCS dataset, which always contains 978 genes of each profile, paraGSEA achieves nearly a thousand times better performance comparing to the other three implementations. Of course, the innate performance advantages of C compared to other script languages such as Python, R or MATLAB also contributes to this result.

All-pairs comparison

To measure the efficiency of the parallel algorithm of comparing profiles, the runtimes are recorded about different stages of the second level of parallelization. All experiments were performed on Tianhe-2 supercomputer within 1 h and 96-core workstation within 120 h. We used all the transcriptome profiles of LINCS phase I dataset to build a complete connectively map quickly while analyzing the runtime of different stages of the second level of parallelization. Figure 3A and B show these results.

We can see that loading time is relatively small compared to other stages (Figure 3A), which indicates that the parallel read strategy is quite efficient. Among them, the collective communications showed the best loading performance (Figure 3B). Also, the writing file operation uses the number of processes as a benchmark. Therefore, the output time is approximately the same among all the conditions with the different number of processes. The test results indicate paraGSEA has well scalability with increasing number of processes.

Then we performed experiments on the 96-core high performance workstation to focus on the most computationally expensive stage. Figure 3C reflects a strong scalability in computing ES matrix stage on whole LINCS phase I dataset. Basically, running time is reduced to half while the number of cores doubles which is largely due to the good load balancing strategy. Finally, paraGSEA built a complete connectively map on this dataset with 424 385 s, ∼117 h and gained a speedup of around 96 times. The result is also showed in Supplementary Table S4 in Supplemental Materials.

To ensure that this condition can be maintained with the continued increase in the number of cores and the scale of dataset, we used different proportion of LINCS phase I dataset, including 5, 10, 20, 40, 60, 80 and 100% respectively, as input to comparative analysis by using different number of nodes on Tianhe-2 supercomputer. Each node made full use of its computing potential and kept running all the 24 cores on a node. The result is showed in Supplementary Table S5 and 6 in Supplementary Materials. We can see the brief visual comparison in Figure 3D. The results show that parallel efficiency can be achieved with an increase in the number of cores while the size of dataset is increasing, which means our tools have very good scaled speedup when applied to large-scale data. As each node only have 64 GB memory and the scale of complete connectively map will increase dramatically with the increase of input dataset, we can also find that when we used few nodes, some big datasets cannot get any result. Therefore, we need more nodes to share the memory pressure and perform large-scale comparison. However, we are surprised to find that the analysis time of whole profiles, involving more than 1 trillion calculations of ESs, can be further reduced within 1843 s, nearly half hour on a 1000-node cluster on Tianhe-2 with a near-linear speedup.

Fast clustering

Because of the randomness of initial cluster centers, convergence steps and total runtime is not the same when we execute it repeatedly under the same parameters. Therefore, we cannot use the total runtime to evaluate the algorithm's performance. However, the runtime of each iteration will be consistent when we use the same number of cores. Thus, we use the ES matrix of front 20 000 and 50 000 expression profiles extracted from phase I dataset as input separately to quickly carry out comparative analysis. The single iteration parallel efficiency of two clustering algorithms shown in Figure 4A. The detail data can also be seen in Supplementary Table S7 in Supplemental Materials.

The results reflect good scalability in finding new clustering centers of each iteration. Specifically, the algorithm can maintain a high level of parallel efficiency in the early stages. By the way, *k*-mediods and *k*-mediods++ basically remained close in runtime and parallel efficiency in the same dataset because they differ only in the discovery of the ini-

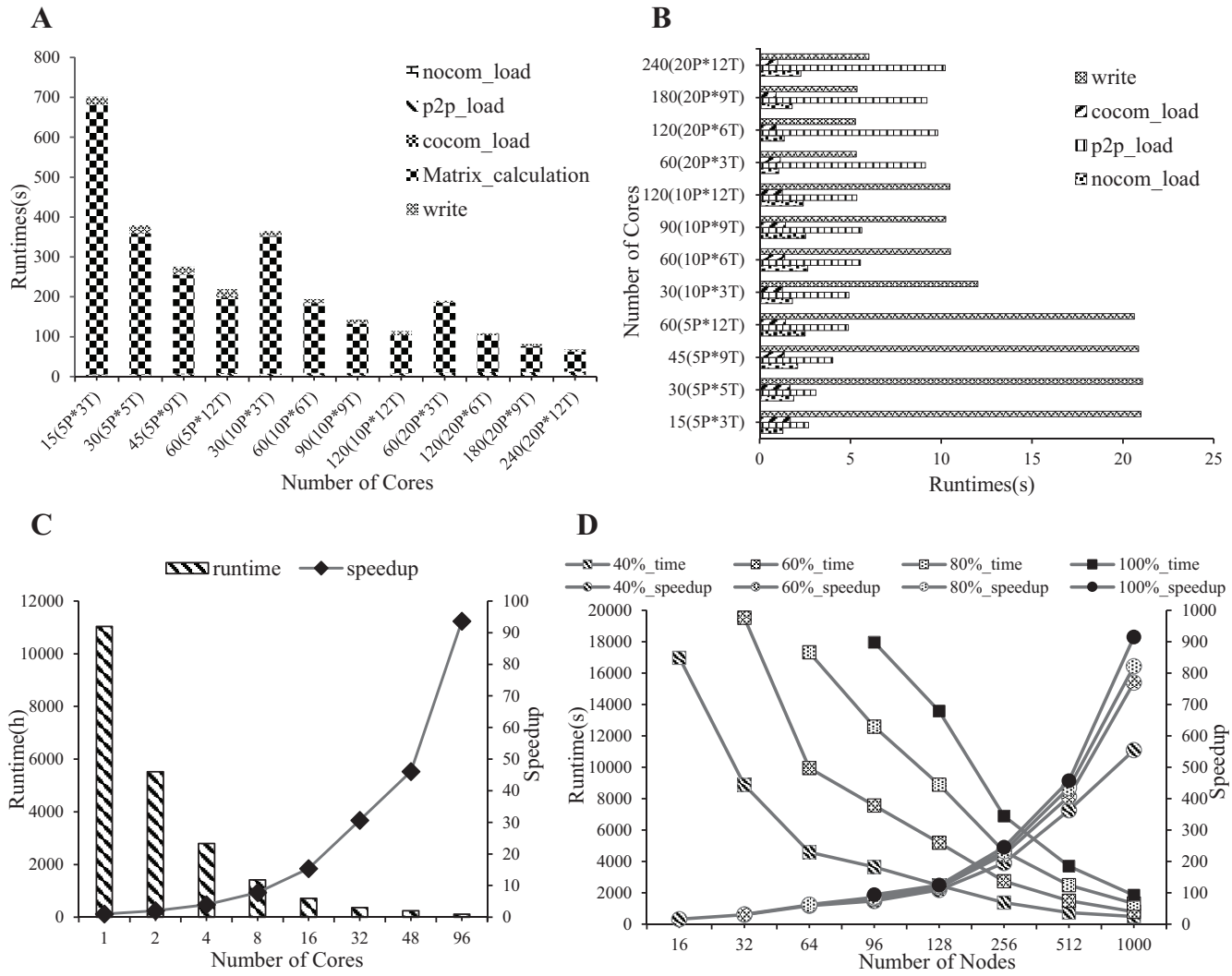

Figure 3. Performance evaluation diagrams of Comparing Profiles. (**A**) Runtime of Comparing Profiles in each stage. (**B**) Runtime of Comparing Profiles in each stage except calculating ES matrix. The 'Cores' in coordinate axis means the degree of second level of parallelization where 'P' means the number of processes and 'T' means the number of threads per process. (**C**) Runtime and Speedup of Comparing Profiles in computing ES matrix stage of whole LINCS phase I dataset. (**D**) Runtime and Speedup of Comparing Profiles in computing ES matrix stage of different proportion of LINCS phase I dataset on Tianhe-2 supercomputer. The 'Nodes' in coordinate axis means the number of node in Tianhe-2 supercomputer used in calculation. Each node made full use of its computing potential and kept running all the 24 cores.

tial cluster centers. Also, we can easily find that parallel efficiency will be better maintained with the increase in the number of cores for larger datasets, which means our cluster tools will also be well applied to large-scale data with enough cluster support.

A good clustering algorithm should make the process of clustering rapidly converge. To evaluate the convergence of our algorithm, we test our clustering tools by setting different numbers of clusters. The result is shown in Figure 4B and C. The detail data can also be seen in Supplementary Table S8 in Supplemental Materials. By the way, we use 60 cores to finish this work.

Because of the randomness of the initial cluster centers, the runtime and convergence steps we got are average results after testing repeatedly. The bigger the number of clusters we set, the slower the speed of convergence we will get. Since we calculate the initial centers to make them as far apart as possible in the *k*-mediods++ algorithm, it can be faster convergence when the number of clusters is bigger compared to the *k*-mediods algorithm (Figure 4B). Also, it may have a better clustering effect.

Given the predefined number of clusters, we can also see that the larger the data we use, the faster the speed of convergence we can get (Figure 4C). For further optimization, our implementation remains a list of previous clustering centers for all iterations before. Once we get a new set of cluster centers, we will judge whether it is in the list to decide whether we can stop the process of clustering. Up to 3000 latest historical records will be hold to ensure efficiency and limited memory consumption, which is also the upper limit of the number of iterations. We should be aware that the algorithm is supposed to be able to converge quickly when the number of centers and the selection of initial centers are appropriate. In case that the algorithm do not converge after thousands

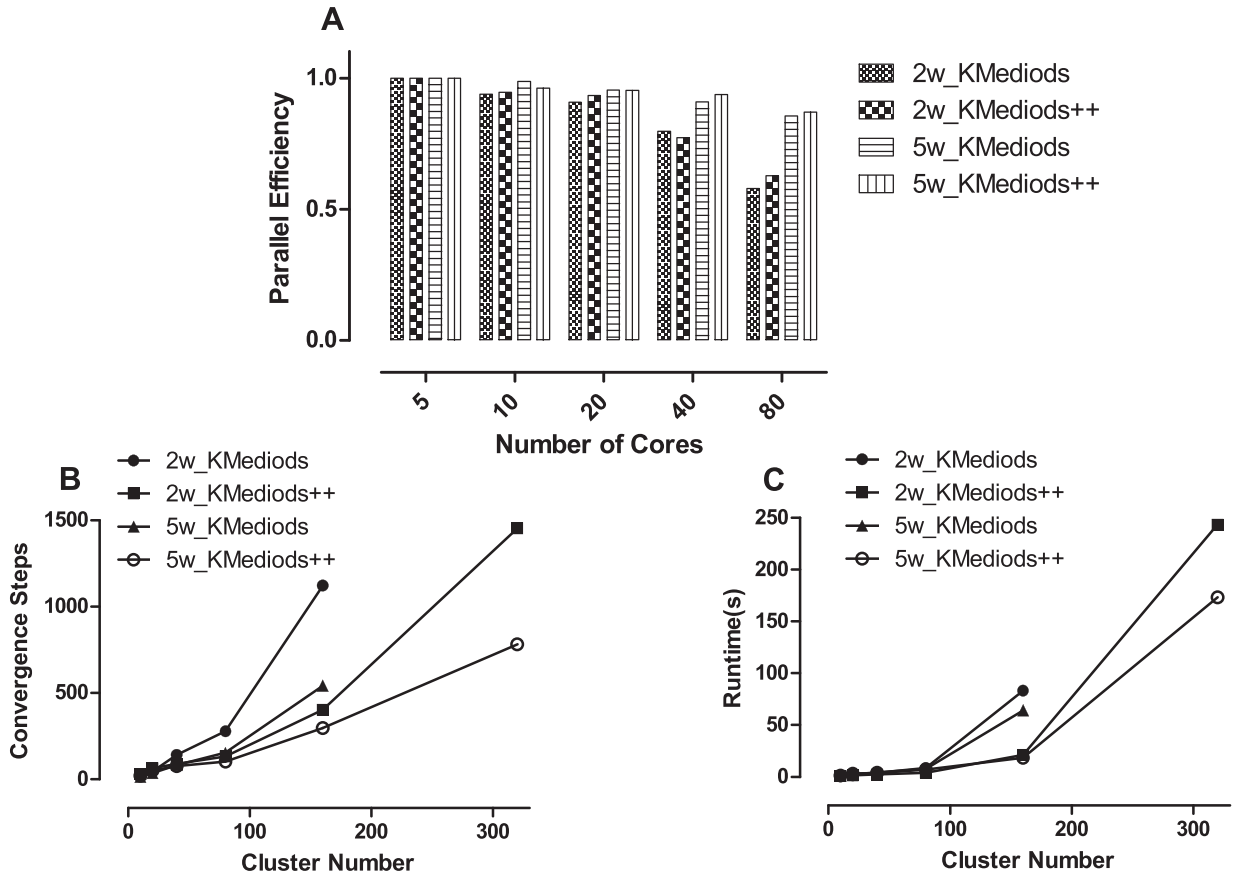

Figure 4. Evaluation diagrams of Comparing Profiles. (**A**) Parallel efficiency of each Iteration. (**B**) Convergence steps of different numbers of clusters. (**C**) Runtime of different numbers of clusters. Note: some data points are not drawn in 320 clusters because the growth of value here is exaggerated.

of iterations, we should stop the algorithm to avoid useless overhead. It is better to choose to re-run the algorithm from different initial centers or different number of centers.

Case study

To demonstrate the capacity of paraGSEA to analyze data, we downloaded a LINCS L1000 file obtained through experiments updated on 03 March 2017 with 1 319 138 profiles (GSE92742) from the NCBI GEO (4). Because these profiles in LINCS did not have category labels and the combination of different experimental conditions can lead to a situation in which seemingly very different sample groups present a similar expression profile formation, it is difficult for us to observe the explicit clustering results without controlling the experimental variables. Thus, we extracted the experimental groups with the perturbation of small molecule 'Vemurafenib' (http://lincs.hms.harvard.edu/db/sm/10068--101/) and kept the duration 24 h with 'trt_cp' perturbation type. Then we obtained 254 profiles from these datasets, and we can expect that the experimental groups acting on the same cell lines should be clustered together.

First, to evaluate the performance and accuracy of paraGSEA, we compared the execution time and results of paraGSEA and GSEA2 (20,21) on these 254 profiles with the same gene set example (Supplementary Table S9) and permutations at 1000 times in the statistical analysis stage. The execution time shows that GSEA2 must take more than 50 s to complete this work, while the Quick Search Tools of paraGSEA only need several milliseconds with a single node. As mentioned, paraGSEA has an absolute advantage on running performance. On the other hand, the accuracy of the three main indicators has also been evaluated (Figure 5A–C). The results indicate that ES maintains absolute consistency between two tools. Also, due to the randomness of the permutation, the two tools have a slight error in calculating normalized ES ($max\ error = 0.073$, $min\ error < 1.0 \times 10^{-3}$, $average\ error = 0.018$) and P-value ($max\ error = 0.009$, $min\ error < 1.0 \times 10^{-5}$, $average\ error = 0.002$). More experiments show that the error with similar degree will also occur even if GSEA2 carries out the same tests twice, which means the accuracy of paraGSEA's results is unquestionable.

Second, after computing ES matrix of these 254 profiles, we clustered them into five groups based on k-mediods++ (Figure 5D). The first group includes 58 profiles. The experimental groups acting on the cell lines 'A549.' 'HCC515, ' 'HEPG2' and 'PC3' are significantly enriched in this group. The second group includes 72 profiles. The experimental groups acting on the cell lines 'A375, ' 'HS578T' and 'HT29' are significantly enriched in this group. The third group in-

Table 1. Integrated clustering results from case study in order to Kappa statistic

	Expected category 1	Expected category 2	Expected category 3	Expected category 4	Expected category 5
Category 1	51	5	0	2	0
Category 2	18	45	5	2	2
Category 3	0	0	43	0	0
Category 4	1	6	0	57	0
Category 5	0	0	0	1	16

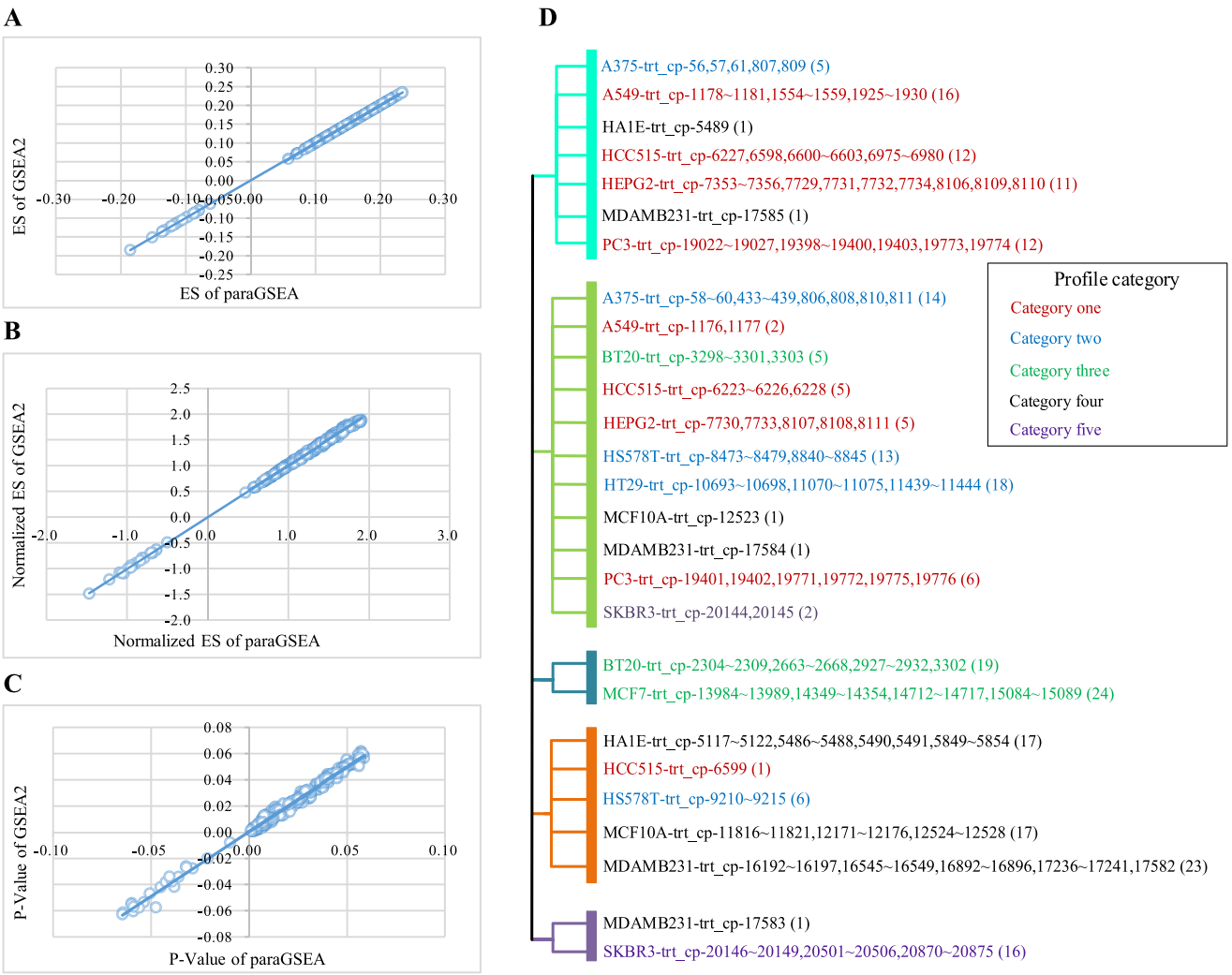

Figure 5. Evaluation results of paraGSEA. (**A**) Comparison between paraGSEA and GSEA2 in calculating ES. (**B**) Comparison between paraGSEA and GSEA2 in calculating normalized ES. (**C**) Comparison between paraGSEA and GSEA2 in calculating *P*-value. (**D**) The clustering results of 254 profiles based on *k*-mediods++. Because these profiles in LINCS did not have category labels, we think the experimental groups acting on the same cell lines should be clustered together after controlling some experimental conditions. The profiles are classified into five groups marked by the color of the dendrogram line. The expected profile classifications are labeled by the color of the font. The figures in parentheses represent the number of profiles for the item.

cludes 43 profiles. The experimental groups acting on the cell lines 'BT20' and 'MCF7' are significantly enriched in this group. The fourth group includes 64 profiles. The experimental groups acting on the cell lines 'HA1E, ' 'MCF10A' and 'MDAMB231' are significantly enriched in this group. The fifth group includes 17 profiles. Only the experimental groups acting on the cell lines 'SKBR3' are significantly enriched in this group. Using the Kappa statistic, we evaluated inter-observer agreement between this classification and the manual classification in these profiles. We first integrated clustering results into an appropriate formation (Table 1), and then the Kappa index showed a substantial agreement ($Kappa = 0.787$, $P < 1.0 \times 10^{-5}$), which means paraGSEA can achieve an ideal clustering effect.

DISCUSSION

paraGSEA provides a scalable parallel computational implementation of Gene Set Enrichment Analysis, which is a widely accepted knowledge-based metric for interpreting gene expression profiles. It can serve as a fast and efficient query tool to make full use of previous large-scale expression profile data to systematically mine the hidden useful knowledge.

paraGSEA achieves high performance on both clusters and workstations, and in benchmark experiments, it can be significantly faster than other popular GSEA implementations such as GSEA-P, SAM-GS and GSEA2 on single node serial mode. One reason why paraGSEA appears to be the fastest tool is that it is designed to adopt many optimizations such as pre-sorting, establishing inverted index or prefix sum removing, so that the overall time complexity to calculate an ES is reduced from $O(mn)$ to $O(m+n)$, where m is the length of gene sets and n is the length of the gene expression profiles. Also, we adopt a global permutation and random sampling strategy in estimating the significance level of ES step, which measures the reliability of ES by statistical methods. This strategy prevents the computing expenses from increasing along with the permutation times. Therefore, there is no doubt that paraGSEA can achieve this absolute advantage on running performance.

More importantly, paraGSEA is designed to accelerate large-scale transcription profile querying and clustering analysis to make up for the lack of tools in parallel computing. Our implementation also allows generating a second level of parallelization by creating several threads per MPI process. Through reasonable data partitioning and an efficient global communication mechanism, we can achieve a strict load balance, maximize data locality and minimize disk I/O. The results of our experiment reflect strong scalability in computing the ES matrix stage, in which we can see that running time is reduced to half when we doubled the number of cores. Also, parallel efficiency will be better maintained with the increase in the number of cores for larger datasets, which means paraGSEA can be well applied to large-scale data with clusters support.

As for clustering analysis, we implemented a general clustering algorithm like k-mediods, which is more robust to noise and outliers compared to k-means, and we have no need to recalculate the ES matrix. This step must depend on the results of comparing profiles. ES is served as a metric of the similarity between two profiles. To ensure that the mutual distance between the initial cluster centers is as far as possible, we improve the algorithm and provide an implementation of k-mediods++, but it will pay more to determine the initial clustering centers. The results of the experiment also reflect excellent scalability in finding new clustering centers of each iteration. Moreover, the bigger the number of clusters we set, the slower the speed of convergence we will get. However, k-mediods++ algorithm can achieve faster convergence when the number of clusters is bigger compared to k-mediods algorithm. Also, to prevent a non-convergence result, we will retain a list consisting of clustering centers in all previous iterations, which may consume more memory with the increase of iteration time. Hence, there is no recommendation to set too many clustering centers to lead to lots of iterations. However, the good news is our parallel cluster tools will also be well applied to large-scale data with enough clusters support and can achieve an ideal clustering effect that we saw in the case study.

In general, paraGSEA is a fast parallel implementation of GSEA to accelerate large-scale transcription profile querying and clustering analysis with support for standard RNA-Seq and microarray data profiles mainly coming from LINCS to meet the need for timely computationally intensive analysis. It can be well applied to large-scale data with enough clusters support by an efficient second level of parallelization and strict data partition and communication strategies.

SUPPLEMENTARY DATA

Supplementary Data are available at NAR Online.

ACKNOWLEDGEMENTS

We thank Wenjian Xu, Hao Hong and Zihan Zhou for helpful discussions, comments and solidarity.

FUNDING

National Nature Science Foundation of China [U1435222, in part]; National Natural Science Foundation of China [61540052, in part]; National Natural Science Foundation of China [61625202, in part]; National Natural Science Foundation of China [61272056, in part]; National Key R&D Program of China [2017YFB0202600, in part]; National Key R&D Program of China [2016YFC1302500, in part]. Funding for open access charge: National Natural Science Foundation of China.
Conflict of interest statement. None declared.

REFERENCES

1. Lucas,A.S., Fowler,J., Chang,K., Kopetz,S., Vilar,E. and Scheet,P. (2014) Cancer in silico drug discovery: a systems biology tool for identifying candidate drugs to target specific molecular tumor subtypes. *Mol. Cancer Therapeut.*, **13**, 3230–3240.
2. Christinat,Y., Pawłowski,R. and Krek,W. (2016) jSplice: a high-performance method for accurate prediction of alternative splicing events and its application to large-scale renal cancer transcriptome data. *Bioinformatics*, **32**, 2111–2119.
3. Vidović,D., Koleti,A. and Schürer,S.C. (2014) Large-scale integration of small molecule-induced genome-wide transcriptional responses, kinome-wide binding affinities and cell-growth inhibition profiles reveal global trends characterizing systems-level drug action. *Front. Genet.*, **5**, 342–355.
4. Barrett,T., Wilhite,S.E., Ledoux,P., Evangelista,C., Kim,I.F., Tomashevsky,M., Marshall,K.A., Phillippy,K.H., Sherman,P.M., Holko,M. *et al.* (2011) NCBI GEO: archive for functional genomics data sets—update. *Nucleic Acids Res.*, **41**, 991–995.
5. Parkinson,H., Kapushesky,M., Shojatalab,M., Abeygunawardena,N., Coulson,R., Farne,A., Holloway,E., Kolesnykov,N., Lilja,P., Lukk,M. *et al.* (2007) ArrayExpress—a public database of microarray experiments and gene expression profiles. *Nucleic Acids Res.*, **35**, 747–750.
6. Tomczak,K., Czerwinska,P. and Wiznerowicz,M. (2015) The Cancer Genome Atlas (TCGA): an immeasurable source of knowledge. *Contemp. Oncol.*, **19**, 68–77.
7. Won,S.J., Wu,H.C., Lin,K.T., Yu,C. H., Chen,Y.T., Wu,C.S., Huang,C.F., Liu,H.S., Lin,C.N. and Su,C.L. (2015) Discovery of molecular mechanisms of lignan justicidin A using L1000 gene

expression profiles and the Library of integrated Network-based Cellular Signatures database. *J. Funct. Foods*, **16**, 81–93.
8. Shao,H., Peng,T., Ji,Z., Su,J. and Zhou,X. (2013) Systematically studying kinase inhibitor induced signaling network signatures by integrating both therapeutic and side effects. *PLoS One*, **8**, 1254–1258.
9. Duan,Q., Flynn,C., Niepel,M., Hafner,M., Muhlich,J.L., Fernandez,N.F., Rouillard,A.D., Tan,C.M., Chen,E.Y., Golub,T.R. *et al.* (2014) LINCS Canvas Browser: interactive web app to query, browse and interrogate LINCS L1000 gene expression signatures. *Nucleic Acids Res.*, **42**, 449–460.
10. Gaggero,M., Leo,S., Manca,S., Santoni,F., Schiaratura,O. and Zanetti,G. (2008) Parallelizing bioinformatics applications with MapReduce. *Cloud Computing and Its Applications*. pp. 22–23.
11. Qiu,X., Ekanayake,J., Beason,S., Gunarathne,T., Fox,G., Barga,R. and Gannon,D. (2009) Cloud technologies for bioinformatics applications. *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*. ACM, Vol. **6**.
12. Stein,L.D. (2009) The case for cloud computing in genome informatics. *Genome Biol.*, **11**, 79–82.
13. Decap,D., Reumers,J., Herzeel,C., Costanza,P. and Fostier,J. (2015) Halvade: scalable sequence analysis with MapReduce. *Bioinformatics*, **31**, 2482–2488.
14. González-Domínguez,J. and Schmidt,B. (2016) ParDRe: faster parallel duplicated reads removal tool for sequencing studies. *Bioinformatics*, **32**, 1562–1564.
15. Duan,Q., Reid,S.P., Clark,N.R., Wang,Z., Fernandez,N.F., Rouillard,A.D., Readhead,B., Tritsch,S.R., Hodos,R., Hafner,M. *et al.* (2016) L1000CDS^2: LINCS L1000 characteristic direction signatures search engine. *NPJ Syst. Biol. Appl.*, **2**, 16015–16026.
16. Mootha,V.K., Lindgren,C.M., Eriksson,K.F., Subramanian,A., Sihag,S., Lehar,J., Puigserver,P., Carlsson,E., Ridderstråle,M., Laurila,E. *et al.* (2003) PGC-1α-responsive genes involved in oxidative phosphorylation are coordinately downregulated in human diabetes. *Nat. Genet.*, **34**, 267–273.
17. Subramanian,A., Tamayo,P., Mootha,V.K., Mukherjee,S., Ebert,B.L., Gillette,M.A., Paulovich,A., Pomeroyh,S.L., Golub,T.R., Lander,E.S. *et al.* (2005) Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *PNAS*, **102**, 15545–15550.
18. Dinu,I., Potter,J.D., Mueller,T., Liu,Q., Adewale,A.J., Jhangri,G.S., Einecke,G., Famulski,K.S., Halloran,P. and Yasui,Y. (2007) Improving gene set analysis of microarray data by SAM-GS. *BMC Bioinformatics*, **8**, 242–249.
19. Liu,Q., Dinu,I., Adewale,A.J., Potter,J.D. and Yasui,Y. (2007) Comparative evaluation of gene-set analysis methods. *BMC Bioinformatics*, **8**, 431–438.
20. Lim,W.K., Lyashenko,E. and Califano,A. (2009) Master regulators used as breast cancer metastasis classifier. *Pac. Symp. Biocomput.*, **14**, 504–515.
21. Carro,M.S., Lim,W.K., Alvarez,M.J., Bollo,Robert.J., Zhao,X.D., Snyder,E.Y., Sulman,Erik.P., Anne,S.L., Doetsch,F., Colman,H. *et al.* (2010) The transcriptional network for mesenchymal transformation of brain tumours. *Nature*, **463**, 318–325.
22. Lamb,J., Crawford,E.D., Peck,D., Modell,J.W., Blat,I.C., Wrobel,M.J., Lerner,J., Brunet,J.P., Subramanian,A., Ross,K.N., Lamb,J. *et al.* (2006) The Connectivity Map: using gene-expression signatures to connect small molecules, genes, and disease. *Science*, **313**, 1929–1935.
23. Qu,X.A. and Rajpal,D.K. (2012) Applications of Connectivity Map in drug discovery and development. *Drug Discov. Today*, **17**, 1289–1298.
24. Park,H.S. and Jun,C.H. (2009) A simple and fast algorithm for K-medoids clustering. *Expert Syst. Appl.*, **36**, 3336–3341.
25. Wang,M., Zhang,W., Ding,W., Dai,D., Zhang,H., Xie,H., Chen,L., Guo,Y. and Xie,J. (2014) Parallel clustering algorithm for large-scale biological datasets. *PLoS One*, **9**, e91315.
26. Zorita,E., Cusc,P. and Filion,J.G. (2015) Starcode: sequence clustering based on all-pairs search. *Bioinformatics*, **31**, 1913–1919.
27. He,S., He,H., Xu,W., Huang,X., Jiang,S., Li,F., He,F. and Bo,X. (2016) ICM: a web server for integrated clustering of multi-dimensional biomedical data. *Nucleic Acids Res.*, **44**, 154–159.
28. Cui,X., He,H., He,F., Wang,S., Li,F. and Bo,X. (2015) Network fingerprint: a knowledge-based characterization of biomedical networks. *Sci. Rep.*, **5**, 13286–13293.