Received: October 11, 2020. Revised: January 4, 2021.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Fast Local Flow-based Method using Parallel Multi-core CPUs Architecture

Rashed Salem[1] **Wafaa Abdel-Moneim**[2]* **Mohamed Hassan**[2]

[1]*Information Systems Department, Faculty of Computers and Information, Menoufia University, Egypt*
[2]*Information Systems Department, Faculty of Computers and Informatics, Zagazig University, Egypt*
* Corresponding author's Email: eng_is_wafaa@yahoo.com

Abstract: Large graphs are available in everywhere such as social networks, web link analysis, and computer networks. Traditional methods of clustering are not suitable to tackle the problem of clustering large graphs because the computation is very costly, which is solved by local graph clustering using a given vertex set as input to detect an accurate cluster. SimpleLocal (SL) algorithm detects a best conductance cuts close to seed vertices set. In this paper, a new Parallel SimpleLocal (PSL) system is proposed using multicore CPUs. OpenMP parallel library is utilized to parallelize the first and second stages of 3StageFlow algorithm whereas the SL algorithm is used for enhancing the runtime. The experiments are performed on various applications from different domains, which are image segmentation and community detection. From the experiments, the proposed method improves the runtimes with 75.43% using 4-cores and 81.01% when using 8-cores over the sequential single core.

Keywords: Graph mining, Graph clustering, Local graph clustering, Parallel algorithms.

1. Introduction

In recent years, graph mining is one of the most relevant research topics due to there are many applications of graphs such as community detection, image segmentation, web search, and social networks analysis. The structure $G=(N, E)$ is used to represent a graph where the nodes are denoted by N and the edges (links that connect the nodes) are denoted by E. The process of collecting input datasets into groups called cluster is the clustering process. Graph clustering is the procedure of gathering the graph vertices into clusters taking into account that there are several edges within each cluster and there are little edges between the clusters.

Many graph clustering algorithms have been described in [1, 2]. These algorithms present definitions of graph clustering and quality measures of the cluster. There are two types of algorithms which are global and local graph clustering. The process of using the entire graph for clustering is known as global graph clustering. However, the process of utilizing specific seed vertex for clustering is known as local graph clustering.

Most conventional algorithms of graph clustering need to treat with the whole graph. Today, massive graphs are available, such as graphs from social media, scientific, and artificial intelligence applications. It is very expensive to perform computations using these algorithms, but local graph clustering algorithms are quicker than conventional algorithms that cover the whole graph. Currently, large-scale graph is available, in which local clusters can be bigger and this may increase running times of local clustering algorithms. Parallelization is utilized to promote the performance of these algorithms. Numerous local graph computations can be run independently in parallel and certain applications benefit from this. Since many input parameters of all local algorithms affect the cluster quality and computation time, it may be difficult to recognize before setting the input parameters for the different independent computations.

The main problem is to find a cluster of the larger graph using local graph clustering techniques, which are time-consuming process. The large processing time problem required to be solved in order to accelerate the execution time of local graph clustering methods.

A parallel version of local clustering algorithm has been contributed in this paper, *i.e.*, Local Flow-based algorithm [3], for improving the performance of the clustering process. This local algorithm is simple and efficient for solving problems in community detection on graph, and image segmentation. Extracting maximum flow computations is achieved by developing a three-stage method. Moreover, the proposed algorithm modifies Dinic's algorithm to reach the best runtime as well as realizing the flexibility and ease of implementation. This algorithm is called SimpleLocal, which based on constructing and updating the local subgraph of the modified augmented graph. Localization can be recognized using an implicit ℓ_1-norm penalty term.

In this paper, a parallel local clustering algorithm has been proposed for improving the quality of the cluster. The local flow-based algorithm is simple and strongly for solving problems in community detection, and image segmentation. The proposed method is implemented to important applications, such as image segmentation and community detection. Multicore CPUs are utilized for applying parallel local flow-based method on these applications. The proposed algorithm improves a local flow-based method to reach the best runtime.

This paper is organized as follows: Section 2 provides related work. Preliminaries of graph structure and multi-core CPUs are described in Section 3. While, Section 4 introduces the SimpleLocal (SL) algorithm, Section 5 introduces the proposed Parallel SimpleLocal (PSL) algorithm. Section 6 discusses the experiments and their results. Finally, Section 7 highlights both the conclusions and future works.

2. Related work

This section addresses related work of the local graph clustering algorithms in the literature. Firstly, Spielman and Teng [4] deal with local graph clustering to solve sparse linear systems using nearly-linear time algorithms. Their research presents Nibble algorithm to partition the graph. Then, Andersen et al. [5] produce an algorithm for local partitioning called PageRank-Nibble using personalized PageRank vectors which improves both Nibble approximation ratio and running time. This algorithm computes the approximate PageRank vectors. Furthermore, Spielman and Teng [6] design an algorithm for spectral method of sparsification graph using nearly linear time to detect an approximate sparsest cut. Finally, Fountoulakis et al. [7] present new trends for optimizing local graph clustering of PageRank-Nibble algorithm.

Local clustering algorithms include several improvements such as new algorithm called "Improve" [8]. This algorithm presents improvement of the quality of graph partitioning without affect the running time by using a subset of vertices as input and produce a new subset of vertices that has a smaller quotient cut.

In the past, sparse cuts can be found by algorithms of local clustering using personalized PageRank and random walks, but evolving sets are used to design the EvoCut algorithm [9]. Applications of local graph clustering algorithms as provided in [10] include identifying communities in networks by the algorithm of Spielman and Teng [4]. A small sparse cut is found using bicriteria approximation algorithm [11]. This algorithm is simple and uses truncated random walk to implement an algorithm locally.

In the literature, the algorithms of local graph clustering are utilized to handle social and web graphs [12-14]. Many problems of graph-based learning, including image segmentation [15] and seeded community detection [5, 16] are addressed, whereas the target is to find the remaining of the pixels by using a set of sample pixels or nodes. Voevodski, et al. [17] as well as Liao et al. [18] provide algorithms to find communities in protein networks. Other researchers deal with community detection by applying local algorithms, *e.g.*, [16, 19-23]. Chung designs a modified version of PageRank known as heat kernel PageRank [24] that contains two input parameters, which are a heat or temperature and a seed. Clusters can be obtained with better guarantees by using several PageRank-Nibble algorithms. These algorithms are internally well-connected in the cluster [25]. There are other local clustering algorithms developed with stronger guarantees [26, 27]. Orecchia and Zhu [28] introduce the first strong local flow-based method that has a quick runtime. For maximum flows, this method is based on a complex diversity of Dinic's algorithm, so it is complicated to be used in practice [29]. Veldt et al. [3] present a new simple algorithm called SL for locally-biased graph based learning. Strongly local is the main feature of this algorithm to find a good conductance cuts without require entire graph.

A lot of local graph clustering algorithms are paralleled by Shun et al. [30] such as Nibble [4,6], PageRank-Nibble [5], deterministic heat kernel PageRank [20], and randomized heat kernel PageRank [31] using the shared- memory multicore environment. The parallel complexities of such algorithms are analyzed. These parallel algorithms accomplish the best speedups on a shared memory multicore in the Ligra graph processing framework.

3. Background

3.1 Graph structure

A graph can be represented by $G\ (V, E)$, where V indicates vertices and E indicates edges of an undirected and unweighted graph. The number of vertices is represented by $n = |V|$ and the number of edges is represented by $m = |E|$. The degree of a vertex is represented by $d(v)$ which can be defined by the edges number incident on v. The set of vertices is denoted by S whose volume is calculated by $vol(S) = \Sigma_{v \in S}\ d(v)$. Moreover, the number of edges that leaves the set is represented by boundary ∂ whose value is calculated as $\partial(S) = \{(x, y) \in E \mid x \in S, y \not\subseteq S\}$. The conductance of cluster S is represented by $\Phi(S) = |\partial(S)|/min\ (vol(S), 2m - vol(S))$. A clustering quality is measured by conductance. Low conductance stands for high quality clustering and high conductance stands for low quality clustering.

3.2 Multi-core CPUs

A multi-core processor is an Integrated Circuit (IC). It has multiple processers that are independent, which called cores to read and carry out the instructions of the program for improved performance and decreased energy consumption. Moreover, multiple tasks are processed in the same time but more efficiently using a multicore CPU [32].

The computational processes are enhanced by utilizing multiple cores or during multithreading on cores that based on the algorithms and their codes [33]. The parallelism level is based on the program data partitioning degree where these sections are handled independently. Even volumes of big data can be reduced using parallel clustering techniques [34]. Among advantages of multi-core processors is the diversity of applications they can solve such as wireless network, biomedical systems, digital signal processing, and image recognition units.

Open Multi-Processing (OpenMP) starts with one thread, which titled with the master thread that works out for the program time. A variable set is ready to any thread, which called the context of the thread's execution. Through execution, the master thread may face the areas of parallel, where novel threads are branched by the master thread. In the finish area of parallel, the branched threads will be closed, and the execution is continued by the master thread. Many advantages of OpenMP are including easier implementation, and lower communication time required in comparison to Message Passing Interface (MPI). Unlike MPI, OpenMP preserves the sequential code and making good use of present-day multicore processors.

Multiple Instruction Multiple Data stream (MIMD) is a basic structure of multicore processor. The whole of threads can be carried out at various cores on the same stream with same shared memory. Therefore, such cores are executed on the same computer rather than utilizing one processor with one core shared with memory, as described in Fig. 1 [35].

4. Local flow-based method

SL [3] is a simple algorithm that is used to calculate the LocalImprove objective function. It starts with a graph $G = (V, E)$, and an initial seed set $R \subset V$ satisfying $vol(R) \leq vol(\bar{R})$. This algorithm depends on building local augmented subgraph $G_R^{'}\ (\alpha, \delta)$. To compute the accurate maximum flow calculations on $G_R^{'}\ (\alpha, \delta)$, a new three-stage method is followed instead of computing approximate maximum flows using Dinic's algorithm.

The local graph is firstly constructed, then passed to the three-phase process and frequented until convergence to maximum flow. The local graph is developed in each iteration, calculating the maximum $s-t$ flow, *i.e.*, flow from the source (s) to sink (t), then the local graph is updated depend on this flow.

4.1 3Stage local max-flow

The 3Stage algorithm computes the maximum $s-t$ flow of an altered augmented graph $G_R^{'}\ (\alpha, \delta)$. The algorithm starts a three-phase process that is frequented until convergence to maximum flow. Fig. 2 describes the 3Stage-Flow flowchart.

To initialize the method, let G^{r} refers to an altered augmented graph $G_R^{'}\ (\alpha, \delta)$. It starts by constructing the local graph $L = (V_L, E_L)$. A G' subgraph contains: union of nodes s, t, R, $neigh(R)$, edges start from s to R, edges among distinct nodes in R, edges start from R to $neigh(R)$, and edges start from t to $neigh(R)$. The vector of flow is represented by F and the overall flow value that transmitted from s to t is indicated by $flow\ (F)$.

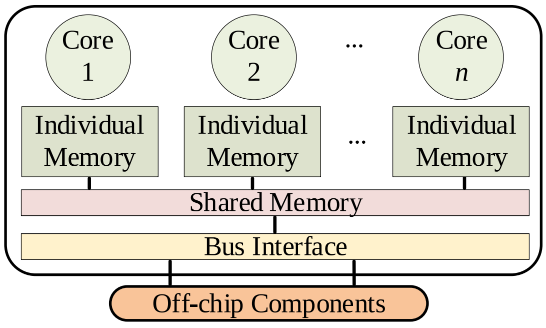


Figure. 1 Multicore processor with shared memory

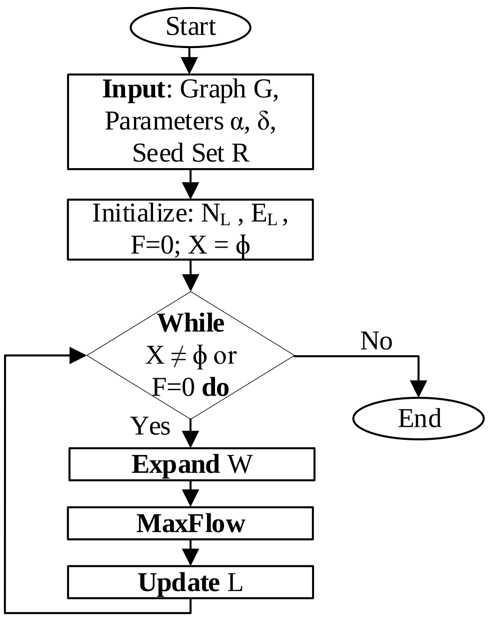

Start
Input: Graph G, Parameters α, δ, Seed Set R
Initialize: N_L , E_L , F=0; X = ϕ
While X ≠ ϕ or F=0 do
No
End
Yes
Expand W
MaxFlow
Update L

Figure. 2 The 3StageFlow flowchart

4.1.1. Stage 1. Expansion

For permitting large flow transmitted from s to t, local graph is expanded in the starting of each iteration. The set of nodes is represented by X to extend on at the starting of the repetition. For each vertex $x \in X$, whole x neighbors are added which now are not a portion of L, and it contains whole edges begin from x to whole its neighbors. For any novel vertex included to L, the edge it shares with the sink t is contained. Initially, starting with set $X = \varphi$ because there is no necessary to extend the local graph now.

4.1.2. Stage 2. Max-flow computation

After expanding the local graph L correctly, the maximum flow f is calculated by utilizing any existence max-flow subroutine. Then, the vector of the flow is redeveloped $F = F + f$. The residual graph of the flow is represented by L_f, which is constructed by changing the capacity c_{ij} of an edge in EL by $c_{ij} - f_{ij}$, where f_{ij} denotes the flow on (i, j) edge, and the edge (j, i) capacity is changed with f_{ij}.

4.1.3. Stage 3. Updates

The flow influences are resolved and defined if the local graph must be widened after calculating a maximum value. Then, the local graph is developed to become a residual graph of f and detects the nodes group that are linked to s by an unsaturated edges string. This stands for the source set S, which is returned when max flow is converged.

4.2 SimpleLocal (SL) algorithm

The SL algorithm uses 3StageFlow method. The SL algorithm takes a graph G and a reference set R as inputs. A best conductance cut of simple local is detected by rendering 3StageFlow frequently to discover α that has a smallest value where the $G_R^{'}$ (α, δ) maximum $s - t$ flow is smaller than $\alpha\ vol(R)$. Fig. 3 illustrates the SL flowchart.

5. The proposed parallel local flow-based methodology

This section shows how to parallelize sequential local clustering algorithms, *i.e.*, 3StageLocal Max-Flow algorithm and SL algorithm. Clustering algorithms are dependent on processing repeatedly sets of vertices and their edges in parallel, where the seed set includes multiple vertices.

5.1 Par3StageFlow

In order to tackle the problem of computation time in SL methodology, it has been necessary to think in parallel solutions to accelerate the process of 3StageFlow. Multi-core CPU is a powerful processing parallel architecture. A new parallel 3StageFlow (Par3StageFlow) algorithm is proposed. As shown in Fig. 4, the Par3StageFlow algorithm can be broken into three steps: expansion, max-flow computation, and updates.

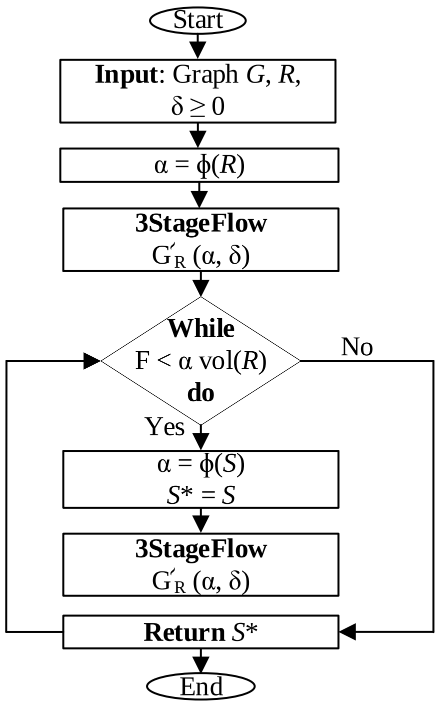


Figure. 3 The SL flowchart

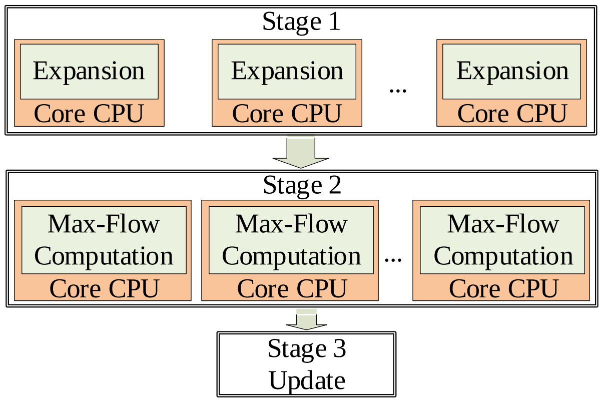


Figure. 4 Speedup steps of the Par3StageFlow algorithm

5.1.1. Step 1. Parallel expansion

In each iteration there is a need for expansion of local graph which starts with a seed set. Starting with a set of vertices of expanded set, each vertex *x* is added to the local graph and all neighbors that are not contained in *L* are added in addition to their edges with *x*. Moreover, each modern vertex added to the local graph *L* is linked with sink *t*. Because of consuming execution time, this step is parallelized.

5.1.2. Step 2. Parallel Max-Flow Computation

Maximum flow is calculated by using Ford-Fulkerson algorithm [36]. The Ford-Fulkerson algorithm solves the maxflow min-cut problem. The Ford-Fulkerson method is iterative, which starts with $f(u, v)$ for $(u, v) \epsilon V$, and initial flow of value equal 0. The method is based on the augmenting path. Algorithm 1 describes the Ford-Fulkerson algorithm. The execution of this phase takes more time to solve this problem, so it needs to use a multi-core processor to obtain high processing speed.

5.1.3. Step 3. Updates

This phase is used to study the flow effects to take a decision if the local graph required to be expanded. Then, updating the local graph is utilized to get the residual graph for discovering a set of vertices (source set *S*) that stay linked to s by an unsaturated

Algorithm 1: Ford-Fulkerson

```
Input: $G$, $R$, locality parameter $\delta \geq 0$
 $\alpha := \phi(R)$
 $[F; S]$:= Par3StageFlow $G'R$ $(\alpha, \delta)$
While flow $(F) < \alpha\, vol(R)$ Do
    $\alpha \leftarrow \phi(S)$; $S^* \leftarrow S$
    $[F, S]$:= Par3StageFlow $G'R$ $(\alpha, \delta)$
End While
Return: $S^*$
```

Algorithm 2: Par3StageFlow

```
Input: graph $G$, parameters $\alpha$, $\delta$, seed set $R$
Initialize: local graph $L = (V_L, E_L)$, $F = 0$; $X = \phi$
While $X \neq \phi$; or $F = 0$ Do
  /* Step 1. Parallel Expand $W$ */
  For $x \in X$ do in parallel
    $V_L \leftarrow V_L \cup$ neigh$(x)$
    $E_L \leftarrow E_L \cup \{(x, v)$: v $\in V_L\} \cup \{(y, t) : y \in neigh(x)\}$
  End For
  /* Step 2. Parallel MaxFlow */
  $f \leftarrow$ ParMaxSTflow$(L)$;
  $F \leftarrow F + f$
  /* Step 3. Update */
  Update $L$
End While
```

edges chain. Furthermore, the algorithm of Par3StageFlow is given in Algorithm 2.

5.2 ParSimpleLocal (PSL) algorithm

PSL algorithm uses Par3StageFlow method to speed up the runtime. PSL algorithm determines a best conductance cut by requesting a Par3StageFlow frequently to detect *α* with the smallest value. Fig. 5 shows the flowchart of PSL algorithm. Moreover, an outline for PSL is presented in Algorithm 3.

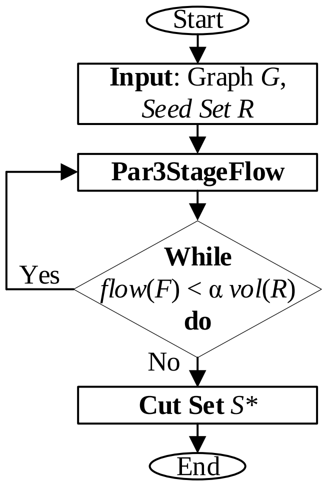


Figure. 5 Flowchart of PSL algorithm

Algorithm 3: PSL

```
Input: $G, s, t, G = (V, E)$
For each edge $(u, v)$ in $E$
  $f(u, v) = f(v, u) = 0$
  While $\exists$ path $p$ from $s$ to $t$ in residual network $G_f$ Do
     $c_f(p) = \min \{c_f(u, v)$: $(u, v)$ is in $p\}$
       For each edge $(u, v)$ on $p$
         $f(u, v) = f(u, v) + cf(p)$
         $f(v, u) = -f(u, v)$
       End For
   End While
End For
```

6. Experiments and discussion

The experiments were performed on undirected graphs of the SL algorithm [3] and PSL algorithm, which are described in Section 5. The first and second stages of 3StageFlow are the most time-consuming steps, with almost 81.09% of the SL algorithm running time. Thus, the most productive steps of the SL algorithm to be parallelized are expansion and max-flow computation stages. OpenMP is a popular parallel programming interface (API) that provides with multicore and shared memory multiprocessing [37]. OpenMP library is utilized to parallelize the first and second stages of 3StageFlow algorithm. Loop parallelism method is used instead of "for loop" that is popular type of parallelism. OpenMP parallel loops are distributed throw threads group by the compiler that is leaded by OpenMP-For.

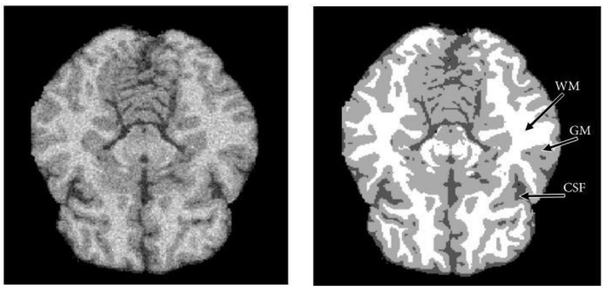
OpenMP supports C, C++, and Fortran on multiple platforms that involve Microsoft Windows and Unix platforms. It contains compiler directives set, library routines, and the variables of environment that effect on the run-time behavior. OpenMP removes the load of making and managing threads from the shoulder of the programmer. The programmer on each core/thread, while managing various data partitions must place a partition of code. Shared memory communicates different threads. As well, OpenMP supplies techniques to coincide the working threads and organize the tasks.

6.1 Image segmentation experiment

6.1.1. Dataset

MRI scan contains the 3D images. The 3D MRI scan region is identified to prove the scalability of the SL algorithm. Challenge of Medical Image Computing and Computer-Assisted Intervention (MICCAI 2012) is used to get a labeled MRI scan with $256 \times 287 \times 256 (\approx 18$ million) voxels [38]. A weighted graph based on adjacent voxel similarity is formed. This graph included about 18 million voxels and 467 million edges. PSL algorithm runs by taking a portion from this data. Fig. 6 shows an example of the brain MRI segmentation.

We begin with a small example from MRI scans to explain how the SL solves the large graph problems. MATLAB performs the execution of SL and 3StageFlow by utilizing Gurobi for solving the problems of max-flow. SL and 3StageFlow are implemented also with C language, using Ford–Fulkerson algorithm for solving the max-flow problems to enhance the runtime of the SL algorithm [3]. These experiments are implemented on a system with 2.3GHz Intel i5-core CPUs processor, 8GB of memory, and Windows 10 64-bit operating system. These experiments are implemented on different sizes of graph as shown in Table 1. Table 2 shows the runtimes of the SL implementation using MATLAB and C language.



(a) (b)

Figure. 6 Brain MRI (a) original MR image, and (b) segmented image with three labels: WM, GM, and CSF

Table 2 shows that the obtained results from C implementation of SL algorithm achieves an extremely improvement in the execution runtime compared with the MATLAB, because C is a lower level language and it provides low-level access to memory compared with MATLAB.

6.1.2. Experimental setup of multi-core system

The experiments were run on a different environment featured with 2.3GHz Intel 8-core CPU, 100 GB of RAM, and 64-bit Linux operating system. The programming language C is used to write the algorithm coding. For parallel implementation, the OpenMP thread library [33] is used. The parallel code

Table 1. Dataset with different graph sizes

Datasets	Vertices |V|	Edges |E|
Ex(200)	200	1576
Ex(300	300	2458
Ex(400)	400	3334
Ex(500)	500	4212
Ex(600)	600	5090
Ex(750)	750	6398

Table 2. Runtime of MATLAB and C of SL algorithm (in seconds)

Datasets	MATLAB	C
Ex(200)	5.309589	0.312487
Ex(300)	8.616222	0.312489
Ex(400)	17.290749	0.406235
Ex(500)	14.579523	0.39061
Ex(600)	22.19735	0.499981
Ex(750)	33.893649	0.593726

is run in 2-core, 4-core, 8-core and 16-core CPU. OpenMP library is utilized for enhancing the runtime of the SL algorithm. Fig. 7 shows the runtime of SL [3] and PSL that runs in 2-core, 4-core, and 8-core on dataset of 800 nodes.

The optimal ratios of speedup for many cores are 50% for 2 cores, 75% for 4 cores, and 87.5% for 8 cores. The proposed algorithm has the speedup ratios of 41.49% for 2 cores, 63.75% for 4 cores, and 81.01% for 8 cores. Fig. 8 clearly displays that the total attitude of the proposed PSL implementation of image segmentation calculations is coordinated with the best speedup.

Implementation is expanded by running the parallel code in 2, 4, 8, and 16-core. Fig. 9 displays an obvious comparison between the execution times of SL [3] and PSL algorithm using multicore CPUs for six datasets given in Table 1. Fig. 10 displays the average of time improvement percentage of PSL algorithm using multicore CPUs of these datasets.

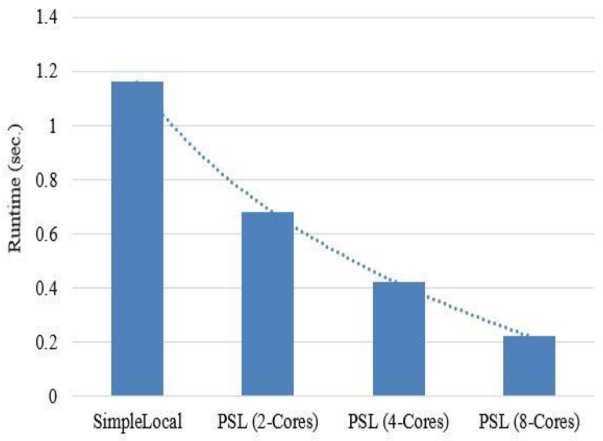


Figure. 7 Runtime of SL and PSL algorithm on a graph of size 800 nodes

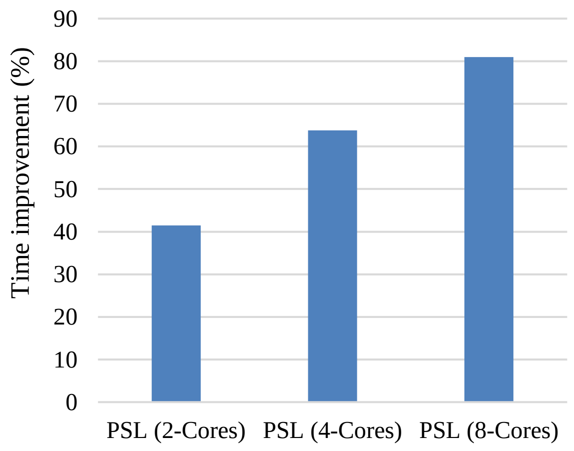


Figure. 8 Time improvement average of PSL using multicore CPUs on a graph of size 800 nodes

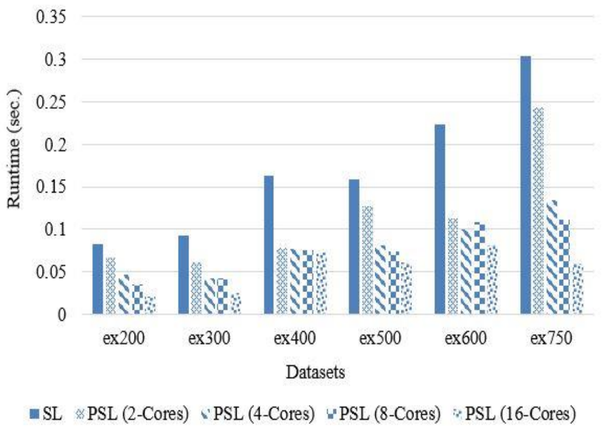


Figure. 9 Runtime of SL and PSL algorithm.

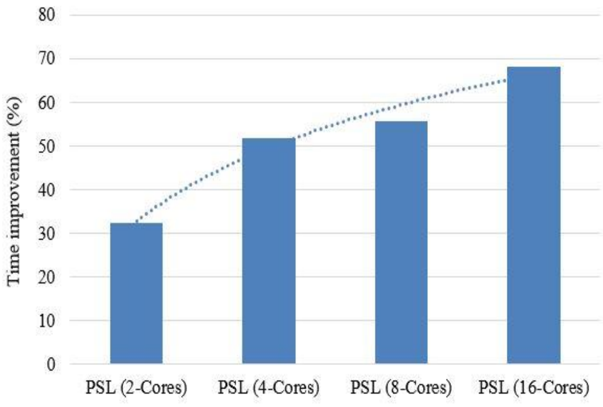


Figure. 10 Time improvement average of PSL algorithm using multicore CPUs.

The results confirm that the gained scores are near to the optimal ratios of parallelization. A quick overview of the scores displays highly refinement result in the running time when using 16-core over the sequential single core. Naturally, a delay is caused by increasing the number of the cores. This brings the enormous interconnect lateness (wire lateness) when datum has to be shifted through the multi-core chip from memories in specific [39].

6.2 Community detection experiment

A set of individuals like the frequency of interactions through the group is larger than that of the interactions among the groups, this called a community. In a network, discovering group where individuals set memberships are not explicitly given, this process called community detection. Many real-world graphs are available in biological networks, web graphs, and large social networks so that the community detection problem in these graphs has taken a great interest recently. This part shows the effects of applying our PSL algorithm on community detection application using multicore CPUs.

6.2.1. Dataset

Social network datasets are downloaded from the large network dataset collection SNAP (Stanford Network Analysis Platform) [40]. These datasets are undirected graph collected from the communities. YouTube is a website for video sharing which includes a social network. Table 3 presents the YouTube datasets with different sizes.

6.2.2. Experimental setup of multi-core system

The experiments were run on a different environment featured with Intel(R) Core(TM) i7-4790CPU (3.60 GHz), 4- core CPU, 8 GB of RAM, and 32-bit Linux operating system. The C language is used for coding the algorithm. OpenMP thread library is used for parallel implementation. The parallel code is run on 2-core and 4-core CPUs.

Table 4 displays the experimental results of applying SL [3] on community detection datasets using different languages. These experiments are implemented on different sizes of YouTube datasets. From the results, the C language improves the running time than MATLAB.

Fig. 11 displays an obvious comparison between the execution times of SL and PSL using multicore CPUs for eight datasets of community detection. Fig. 12 displays the time improvement percentage of PSL algorithm using multicore CPUs of these datasets.

Table 3. Youtube datasets with different sizes

Datasets	Vertices |V|	Edges |E|
G(200)	200	872
G(300)	300	1626
G(400)	400	3050
G(500)	500	4600
G(600)	600	5692
G(700)	700	6556
G(800)	800	8476
G(900)	900	12232

Table 4. Runtime of SL on community detection datasets using MATLAB and C (in seconds)

Datasets	MATLAB	C language
G(200)	6.727062	0.265614
G(300)	21.257542	0.609352
G(400)	280.316656	1.884063
G(500)	148.340843	1.755795
G(600)	219.561633	4.842398
G(700)	155.596251	3.088185
G(800)	455.229502	5.810315
G(900)	493.980067	7.015495

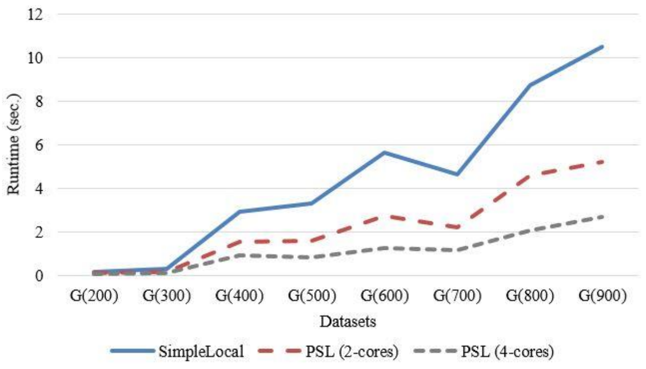


Figure. 11 Runtime of the SL and PSL using multicore CPUs on community detection datasets

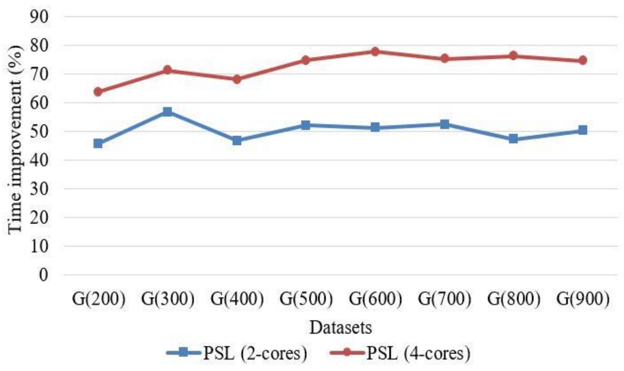


Figure. 12 Time improvement of PSL algorithm using multicore CPUs on community detection datasets

The proposed algorithm has the speedup ratios of 50.34% for 2 cores, and 72.75% for 4 cores. Fig. 13 displays the average of time improvement percentage of PSL algorithm using multicore CPUs of these datasets. The results of the community detection datasets are relative to the ratios of the optimal parallelization, where the trend of the proposed parallel implementation of community detection is coordinated with the best speedup. A quick review of the results presents a highly advanced difference in the execution time when using 4-core over the sequential single core.

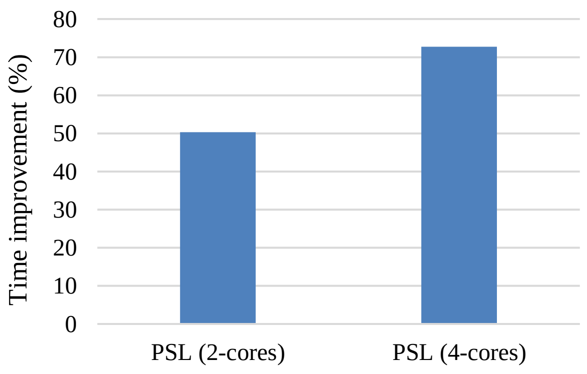


Figure. 13 Time improvement average of PSL algorithm using multicore CPUs of community detection datasets

7. Conclusions

SL algorithm has time-consuming in computation. A novel parallel algorithm is proposed and successfully implemented for computing image segmentation, and community detection. The proposed algorithm is evaluated on parallel architecture multicore CPUs. OMP parallel library is utilized to parallelize the first and second stages of the 3StageFlow algorithm where the PSL is used for enhancing the runtime.

A new parallel algorithm is suggested and executed successfully for graph clustering problem. The proposed algorithm of local flow-based method is implemented on parallel architecture multicore CPUs. The proposed method is implemented on different applications, which represent different domains including image segmentation and community detection. From the experiments, the proposed method improves the runtimes by 75.43% using 4-cores and 81.01% when using 8-cores over the sequential single core. The scores of the executed experiments are very near to the optimal ratios of parallelization for the multi-core CPUs. When comparing the results with sequential, the proposed parallel technique is very speed.

This area of graph clustering study is open where numerous opinions for future work could be presented to assist researchers. We organize to expand our contribution to approximate maximum-flow solutions by using the modern invention, e.g., approximate maximum-flows in nearly-linear time.

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

The entire work of conceptualization, formal analysis, validation, implementation, writing, editing and modification of manuscript were done by Rashed Salem and Wafaa Abdel-Moneim under the supervision of Mohamed Hassan.

References

[1] S. Schaeffer, "Graph clustering", *Computer Science Review*, Vol. 1, No. 1, pp. 27–64, 2007.

[2] C. Aggarwal and H. Wang, "A survey of clustering algorithms for graph data", *Managing and Mining Graph Data. Springer*, pp. 275–301, 2010.

[3] N. Veldt, D. Gleich, and M. Mahoney, "A simple and strongly-local flow-based method for cut improvement", In*: Proc. of International Conference on Machine Learning*, pp. 1938–1947, 2016.

[4] D. Spielman and S.-H. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems", In: *Proc. of the 36th Annual ACM Symposium on Theory of Computing*, pp. 81-90, 2004.

[5] R. Andersen, F. Chung, and K. Lang, "Local graph partitioning using PageRank vectors", In: *Proc. of 47th Annual IEEE Symposium on Foundations of Computer Science*, pp. 475-486, 2006.

[6] D. A. Spielman and S.-H. Teng, "A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning", *SIAM Journal on Computing*, Vol. 42, No. 1, pp. 1–26, 2013.

[7] K. Fountoulakis, X. Cheng, J. Shun, F. Roosta-Khorasani, and M. W. Mahoney, "Exploiting optimization for local graph clustering", *arXiv preprint arXiv:1602.01886*, 2016.

[8] R. Andersen and K. J. Lang, "An algorithm for improving graph partitions", In: *Proc. of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 651–660, 2008.

[9] R. Andersen and Y. Peres, "Finding sparse cuts locally using evolving sets", In*: Proc. of the 41st Annual ACM Symposium on Theory of Computing*, pp. 235–244, 2009.

[10] R. Andersen and K. J. Lang, "Communities from seed sets", In: *Proc. of the 15th International Conference on World Wide Web*, pp. 223–232, 2006.

[11] T. C. Kwok and L. C. Lau, "Finding small sparse cuts by random walk", *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. Springer*, pp. 615–626, 2012.

[12] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters", *Internet Mathematics*, Vol. 6, No. 1, pp. 29–123, 2009.

[13] J. Leskovec, K. J. Lang, and M. Mahoney, "Empirical comparison of algorithms for network community detection", In: *Proc. of the 19th International Conference on World Wide Web*, pp. 631–640, 2010.

[14] L. G. Jeub, P. Balachandran, M. A. Porter, P. J. Mucha, and M. W. Mahoney, "Think locally, act locally: Detection of small, mediumsized, and large communities in large networks", *Physical Review E*, Vol. 91, No. 1, p. 012821, 2015.

[15] M. W. Mahoney, L. Orecchia, and N. K. Vishnoi, "A local spectral method for graphs:

With applications to improving graph partitions and exploring data graphs locally", *Journal of Machine Learning Research*, Vol. 13, No. 1, pp. 2339–2365, 2012.

[16] I. M. Kloumann and J. M. Kleinberg, "Community membership identification from small seed sets", In: *Proc. of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1366–1375, 2014.

[17] K. Voevodski, S.-H. Teng, and Y. Xia, "Finding local communities in protein networks", *BMC Bioinformatics*, Vol. 10, No. 1, p. 297, 2009.

[18] C.-S. Liao, K. Lu, M. Baym, R. Singh, and B. Berger, "IsoRankN: spectral methods for global alignment of multiple protein networks", *Bioinformatics*, Vol. 25, No. 12, 2009.

[19] Y. Wu, R. Jin, J. Li, and X. Zhang, "Robust local community detection: on free rider effect and its elimination", In: *Proc. of the VLDB Endowment*, Vol. 8, No. 7, pp. 798–809, 2015.

[20] K. Kloster and D. F. Gleich, "Heat kernel based community detection", In: *Proc. of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

[21] J. Whang, D. Gleich, and I. Dhillon, "Overlapping community detection using seed set expansion", In: *Proc. of the 22nd ACM International Conference on Information & Knowledge Management*, pp. 2099–2108, 2013.

[22] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth", *Knowledge and Information Systems*, Vol. 42, No. 1, pp. 181–213, 2015.

[23] D. Gleich and C. Seshadhri, "Vertex neighborhoods, low conductance cuts, and good seeds for local community methods", In: *Proc. of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 597–605, 2012.

[24] F. Chung, "A local graph partitioning algorithm using heat kernel PageRank", *Internet Mathematics*, Vol. 6, No. 3, pp. 315–330, 2009.

[25] Z. A. Zhu, S. Lattanzi, and V. S. Mirrokni, "A local algorithm for finding well-connected clusters." In: *Proc. of International Conference on Machine Learning*, pp. 396–404, 2013.

[26] S. O. Gharan and L. Trevisan, "Approximating the expansion profile and almost optimal local graph clustering", In: *Proc. of IEEE 53rd Annual Symposium on Foundations of Computer Science*, pp.187–196, 2012.

[27] T. C. Kwok, L. C. Lau, and Y. T. Lee, "Improved Cheeger's inequality and analysis of local graph partitioning using vertex expansion and expansion profile", *SIAM Journal on Computing*, Vol. 46, No. 3, pp. 890–910, 2017.

[28] L. Orecchia and Z. A. Zhu, "Flow-based algorithms for local graph clustering", In: *Proc. of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1267–1286, 2014.

[29] C. Qi and J. Diao, "A bio-inspired algorithm for maximum matching in bipartite graphs", *IAENG International Journal of Computer Science*, Vol. 47, No. 1, 2020.

[30] J. Shun, F. Roosta-Khorasani, K. Fountoulakis, and M. W. Mahoney, "Parallel local graph clustering", *arXiv preprint: 1604.07515*, 2016.

[31] F. Chung and O. Simpson, "Computing heat kernel PageRank and a local clustering algorithm", *European Journal of Combinatorics*, Vol. 68, pp. 96–119, 2018.

[32] A. Sethi and H. Kushwah, "Multicore processor technology advantages and challenges", *International Journal of Research in Engineering and Technology,* Vol. 4, No. 9, pp. 87–89, 2015.

[33] A. Vajda, "Multi-core and many-core processor architectures", *Programming Many-Core Chips*. *Springer*, pp. 9–43, 2011.

[34] V. S. Moertini, G. W. Suarjana, L. Venica, and G. Karya, "Big data reduction technique using parallel hierarchical agglomerative clustering", *IAENG International Journal of Computer Science*, Vol. 45, No. 1, 2018.

[35] R. Chhibber and R. Garg, "Multicore processor, parallelism and their performance analysis", *IJARCST*, Vol. 2, pp. 31–37, 2014.

[36] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network", *Canadian Journal of Mathematics,* Vol. 8, pp. 399–404, 1956.

[37] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming", *IEEE Computational Science and Engineering,* Vol. 5, No. 1, pp. 46–55, 1998.

[38] D. S. Marcus, T. H. Wang, J. Parker, J. G. Csernansky, J. C. Morris, and R. L. Buckner, "Open Access Series of Imaging Studies (OASIS): cross-sectional mri data in young, middle aged, nondemented, and demented older adults", *Journal of Cognitive Neuroscience,* Vol. 19, No. 9, pp. 1498–1507, 2007.

[39] B. Venu, "Multi-core processors-an overview", *arXiv preprint arXiv*:1110.3535, 2011.

[40] J. Leskovec, "Stanford Large Network Dataset Collection", *https://snap.stanford.edu/data/*, 2020, [Online; accessed 01 Aug. 2020].