Implementing a Parallel Graph Clustering Algorithm with Sparse Matrix Computation

Jun Chen, Peigang Zou
High Performance Computing Center,
Institute of Applied Physics and Computational Mathematics

chenjun@iapcm.ac.cn

IPDPS'2018 workshop

OUTLINE

- Graph clustering
  - Peer pressure clustering(PPCL)
- Challenges
- A solution: based on Large graph platforms/libraries
  - Combinatorial BLAS
- Related works
- Parallel PPCL algorithm with matrix computation
- Numerical Results
- Discussions
- Conclusion

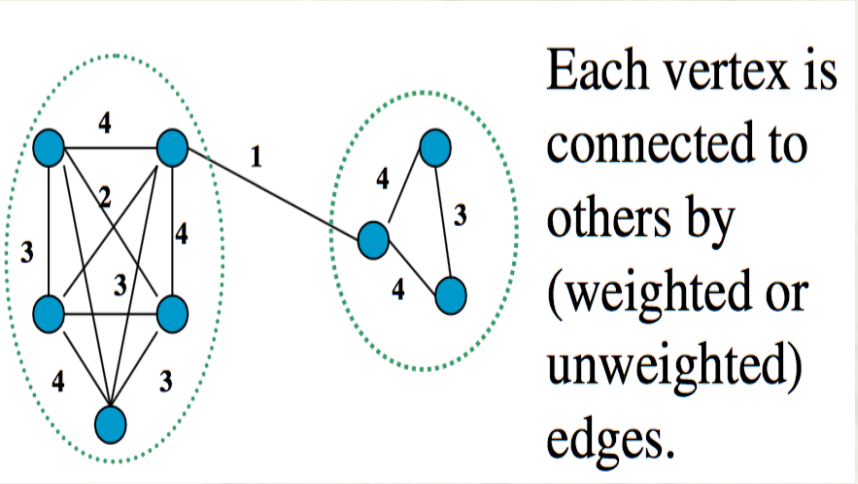
1、Graph Clustering

- A wide class of algorithms to classify vertices in a group into many clusters, where the vertices in the same cluster have high connectivity than those in various clusters.

Graph clustering Vs. Vector clustering

Clustering： find natural groups.

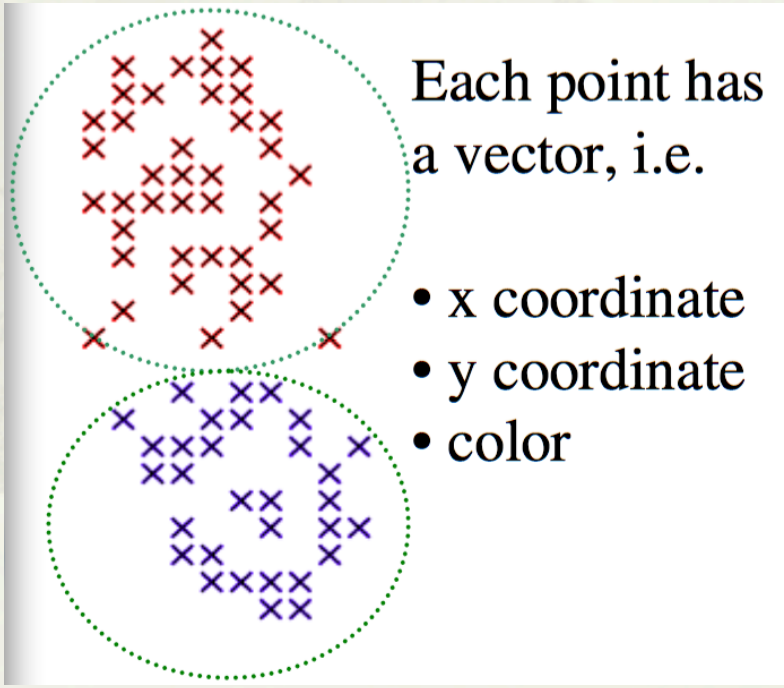
Graph clustering



Each vertex is connected to others by (weighted or unweighted) edges.

Classified by relationship between points.

Vector clustering

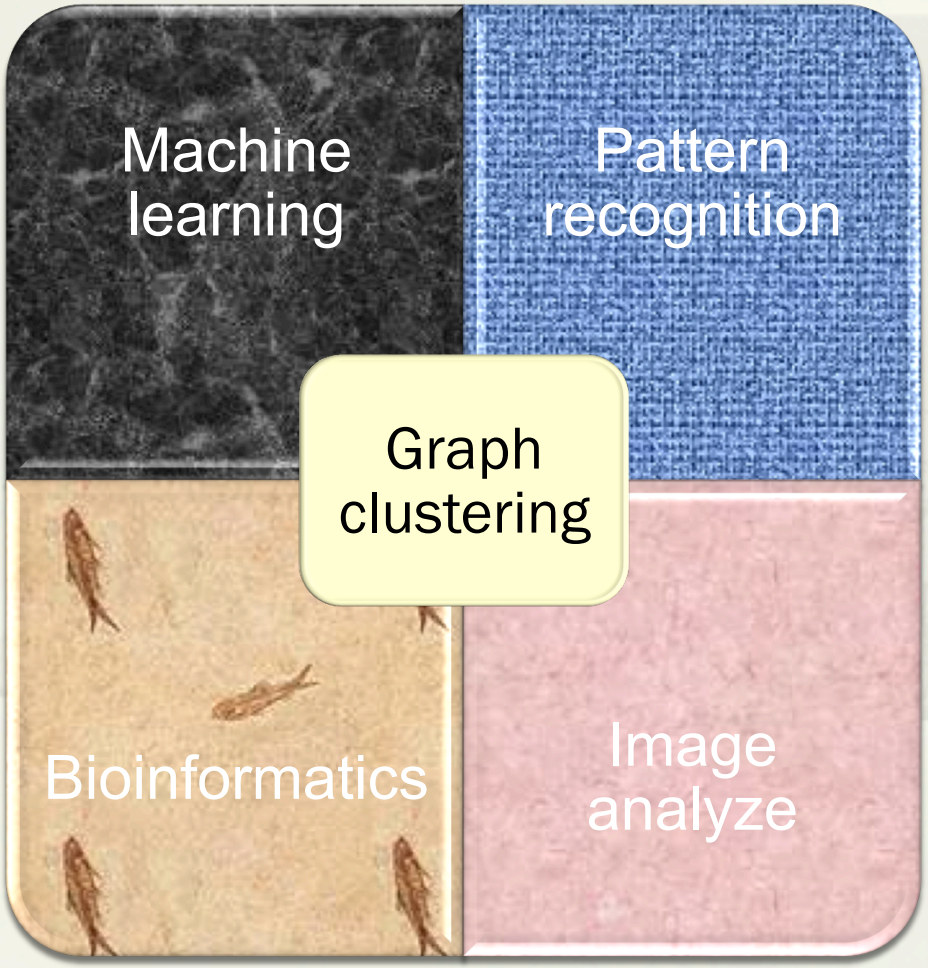


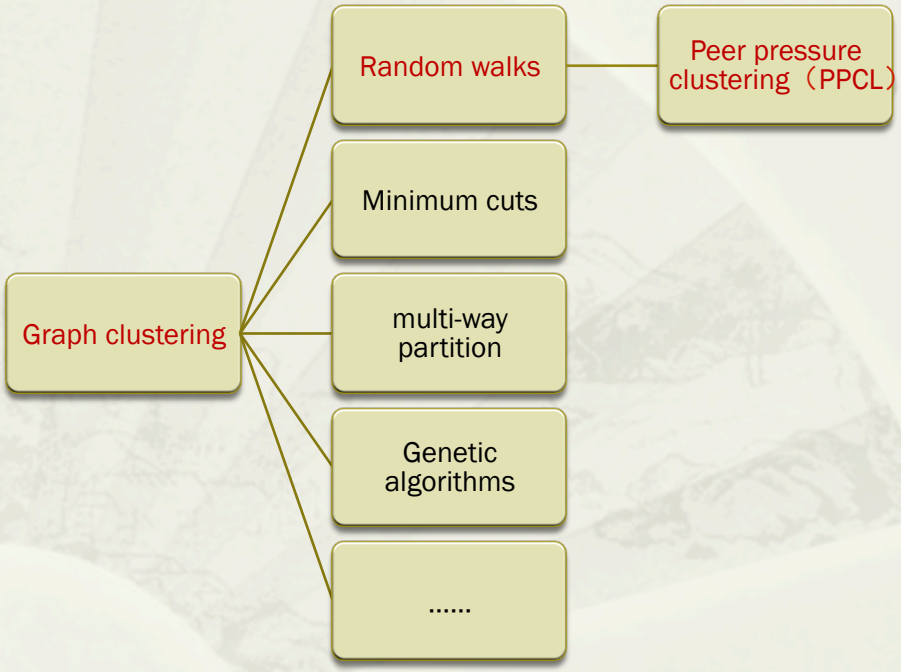
Each point has a vector, i.e.

- x coordinate
- y coordinate
- color

Classified by distances between points.

Graph clustering （cont.）



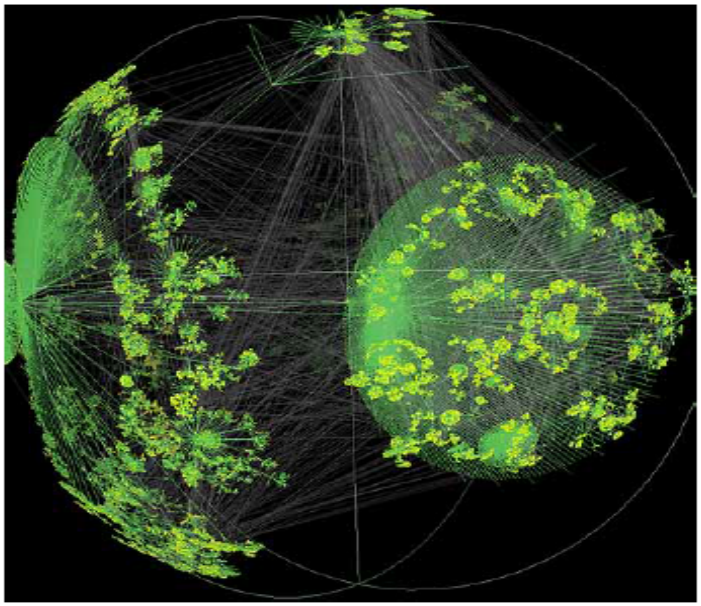


2. Challenges

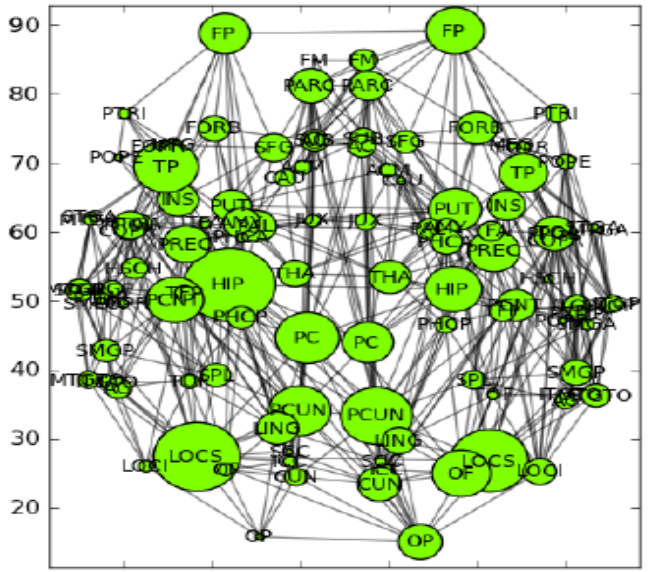
2.1 The size of Graph is growing

Internet structure

Social interactions



Graph Theoretical analysis of Brain Connectivity



E.g., # of Facebook users > 1 billion. A big graph!

2.2. Large Graph Computation

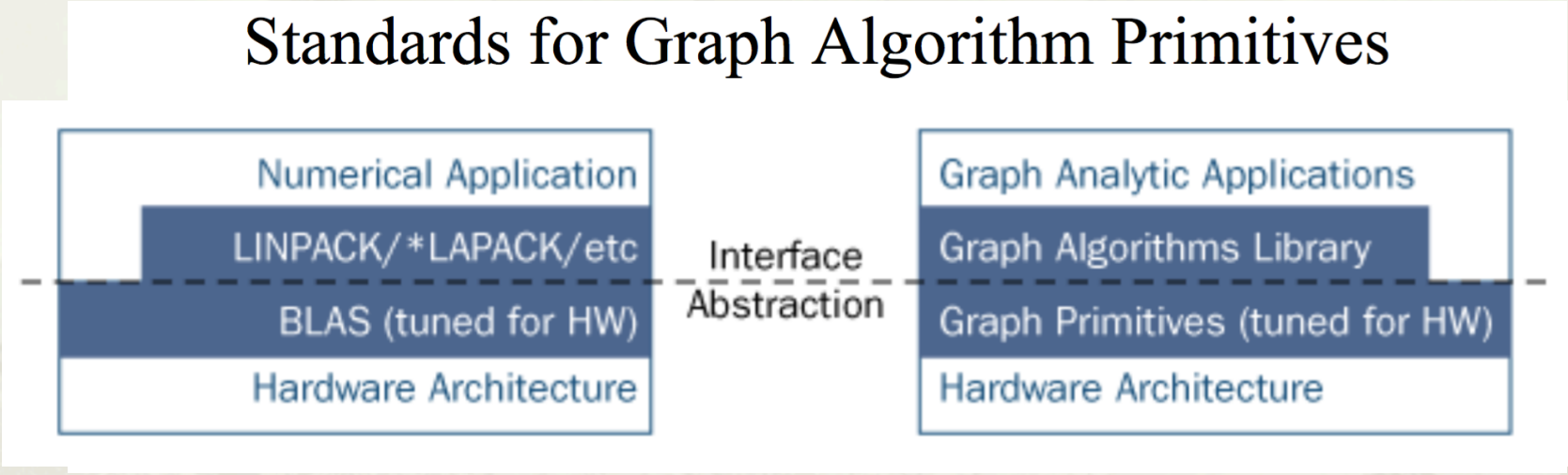
- **Parallel graph clustering implement is difficult. It requires:**
  - well-suited description of the natural sparse locality
  - Storage them effectively
  - High performance computing
- **High performance challenge**
  - Scalability: time should <= O(n). Memory consumption <O(nxn).
  - Parallel patterns for solving PDEs in typical scientific computing is based on dense computations. They are not suitful for the sparse characteristic in large graph computation.
  - MapReduce pattern for big data problems has low efficiency.

✓ A Solution: Based on Large Graph platform/library

3. Large graph platforms/Libraries

- ScaleGraph
  - TITECH, JAPAN
  - Implements Pregel model provided by google， and optimize its collective communication and memory management methods.
  - Goal: can analyze the graphs containing 10 billion nodes and edges.
- DisBelief
  - Google
  - A parallel framework for deep learning
- PBGL(Parallel Boost Graph Library)
  - Indiana University, USA
  - C++ graph library
- GAPDT（Graph Algorithm and Pattern Discovery Toolbox）
  - UCSB, USA
  - Provide interactive graph operations, and can parallel run on Star-P, a parallel version of MATLAB.
  - Use distributed sparse array to describe the parallel operations.

- Graph BLAS
  - Defines a core set of matrix-based graph operations that can be used to implement a wide class of graph algorithms in a wide range of programming environments.



- etc.

Combinatorial BLAS

CombBLAS:

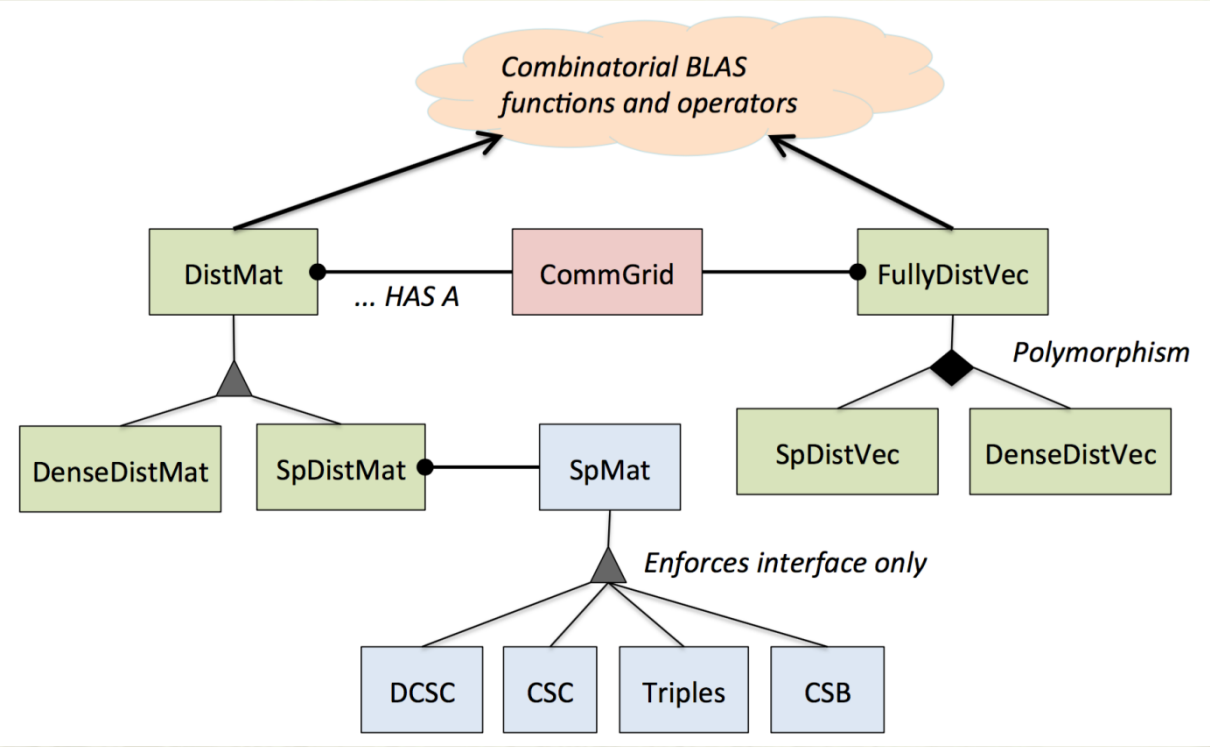
- A representation implementation of Graph BLAS
- Multi-core parallelism based on MPI
- A collective set of basic linear algebra operations.

Table 1. Many basic linear algebra operations in CombBLAS

Function	Parameters	Returns	Math Notation
SpGEMM	- sparse matrices **A, B** and **C** - unary functors (op)	sparse matrix	**C** += op(**A**) * op(**B**)
SpM{Sp}V (Sp: sparse)	- sparse matrix **A** - sparse/dense vector **x**	sparse/dense vector	**y** = **A** * **x**
SpEWiseX	- sparse matrices or vectors - binary functor and predicate	in place or sparse matrix/vector	**C** = **A** .* **B**
Reduce	- sparse matrix **A** and functors	dense vector	**y** = sum(**A**, op)
SpRef	- sparse matrix **A** - index vectors **p** and **q**	sparse matrix	**B** = **A**(**p**,**q**)
SpAsgn	- sparse matrices **A** and **B** - index vectors **p** and **q**	none	**A**(**p**,**q**) = **B**
Scale	- sparse matrix **A** - dense matrix **B** or vector **X**	none	∀ **A**(i,j)!=0: **A**(i,j) *=**B**(i,j) + related forms for **X**
Apply	- any matrix or vector **X** - unary functor (op)	none	op(**X**)

*based on the Combinatorail BLAS from Buluç and Gilbert

Combinatorial BLAS (Cont.)



[Buluç A, Gilbert J R. The Combinatorial BLAS: design, implementation, and applications]

4. Related works

- Introduction of random walks method
  - The cluster assignment of a vertex will be the same at that of most of its neighbors.
- Current representative parallel PPCL methods.
  - Parallel PPCL in SPARQL
  - Parallel PPCL in STAR-P

Related works about parallel PPCL

Paper 1：

Implementing Iterative Algorithms with SPARQL

Robert W. Techentin, Barry K. Gilbert
Mayo Clinic
Rochester, MN
{techentin.robert, gilbert.barry}@mayo.edu

Adam Lugowski, Kevin Deweese, John Gilbert
UC Santa Barbara
Santa Barbara, CA
{alugowski,kdeweese, gilbert}@cs.ucsb.edu

Eric Dull, Mike Hinchey, Steven P. Reinhardt
YarcData LLC
Pleasanton, CA
{edull,mhinchey,spr} @yarcdata.com

PPCL in SPARQL

Results：

- ✓ Maximum processor number：64
- ✓ RDF Graph：10000 vertices, 232,000 edges
- • 200 秒

➢ SPARQL is a similar SQL tool for RDF graph.

RDF Graph：stores meta-data for web resources. Its vertex identify a resource, its arc describes the resource attributes.

Paper 2：

High–performance graph algorithms from parallel sparse matrices

John R. Gilbert[1]*, Steve Reinhardt[2], and Viral B. Shah[1]

[1] University of California, Santa Barbara
{gilbert,viral}@cs.ucsb.edu†
[2] Silicon Graphics Inc.
spr@sgi.com

PPCL in STAR-P

Results：

- ✓ Maximum processor number：128
- ✓ R-MAT graph：2097152 vertices, 18305177 edges (scale 21)
- ✓ Low performance.

➢ STAR-P is a parallel implementation of MATLAB.

5. Parallel PPCL Algorithm with Matrix Computation

- Standard PPCL algorithm
- Alternative PPCL algorithm based on linear algebra
- Parallel PPCL algorithm based on linear algebra
- Parallel PPCL implementation on ComBLAS

5.1、standard PPCL algorithm

Algorithm 1.

PeerPressure($G = (V, E), C_i$)

1 for $(u, v, w) \in E$

2 do $T(v)(C_i(u)) \leftarrow T(v)(C_i(u)) + w$

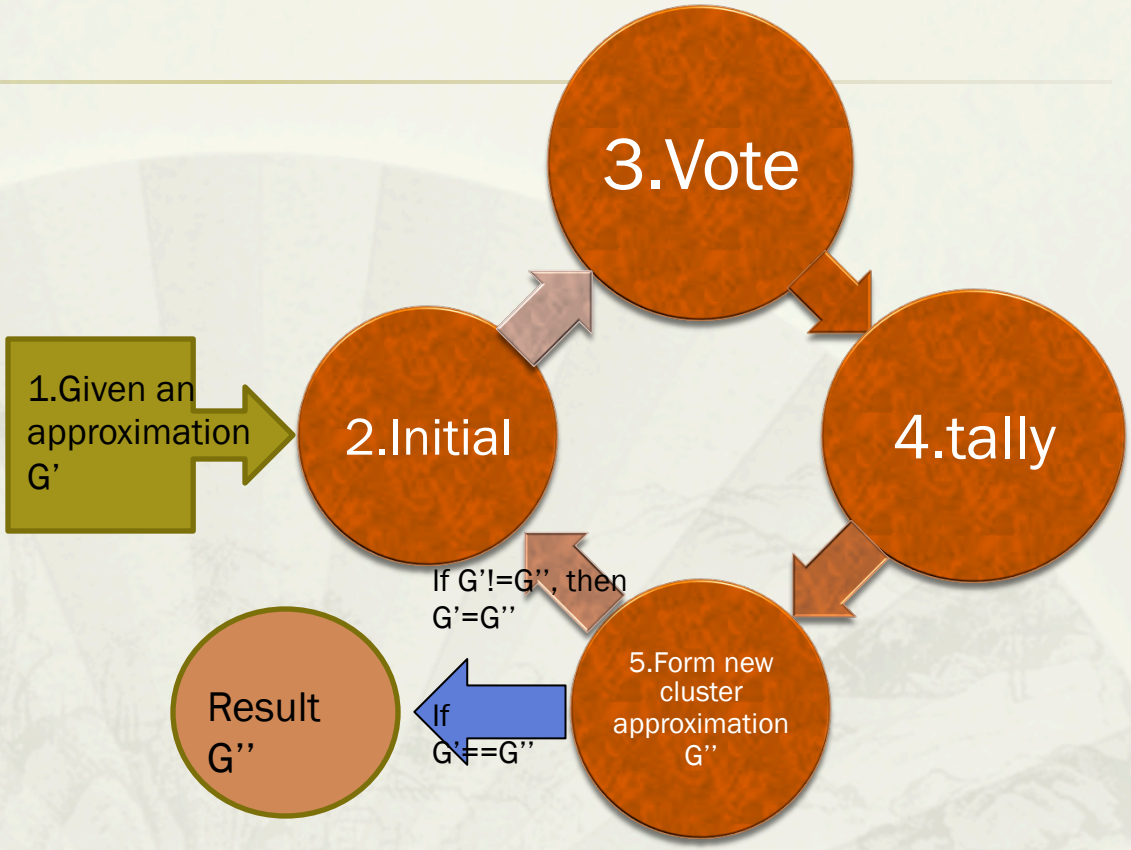
3 for $n \in V$

4 do $C_f(n) \leftarrow i : \forall j \in V : T(n)(j) \le T(n)(i)$

5 if $C_i == C_f$

6 then return C_f

7 else return PeerPressure($G = (V, E), C_f$)



It starts with an initial cluster assignment, e.g., each vertex being in its own cluster.Each iteration performs an election at each vertex to select its cluster num. The votes are the cluster assignments of its neighbors. Ties are settled by selecting the lowest cluster ID to maintain determinism here. The algorithm converges when two consecutive iterations have a tiny difference between them.

5.2 PPCL algorithm based on Linear algebra

Algorithm 2:

PeerPressure($G = \mathrm{A} : \mathrm{R}_{+}^{N\times N}, \mathrm{C_i} : \mathrm{B}^{N\times N}$)

1 $\mathrm{T} : \mathrm{R}_{+}^{N\times N}$ $C_f : \mathrm{B}^{N\times N}$ $\mathrm{m} : \mathrm{R}_{+}^{N}$

2 $\mathrm{T} = \mathrm{C_i A}$

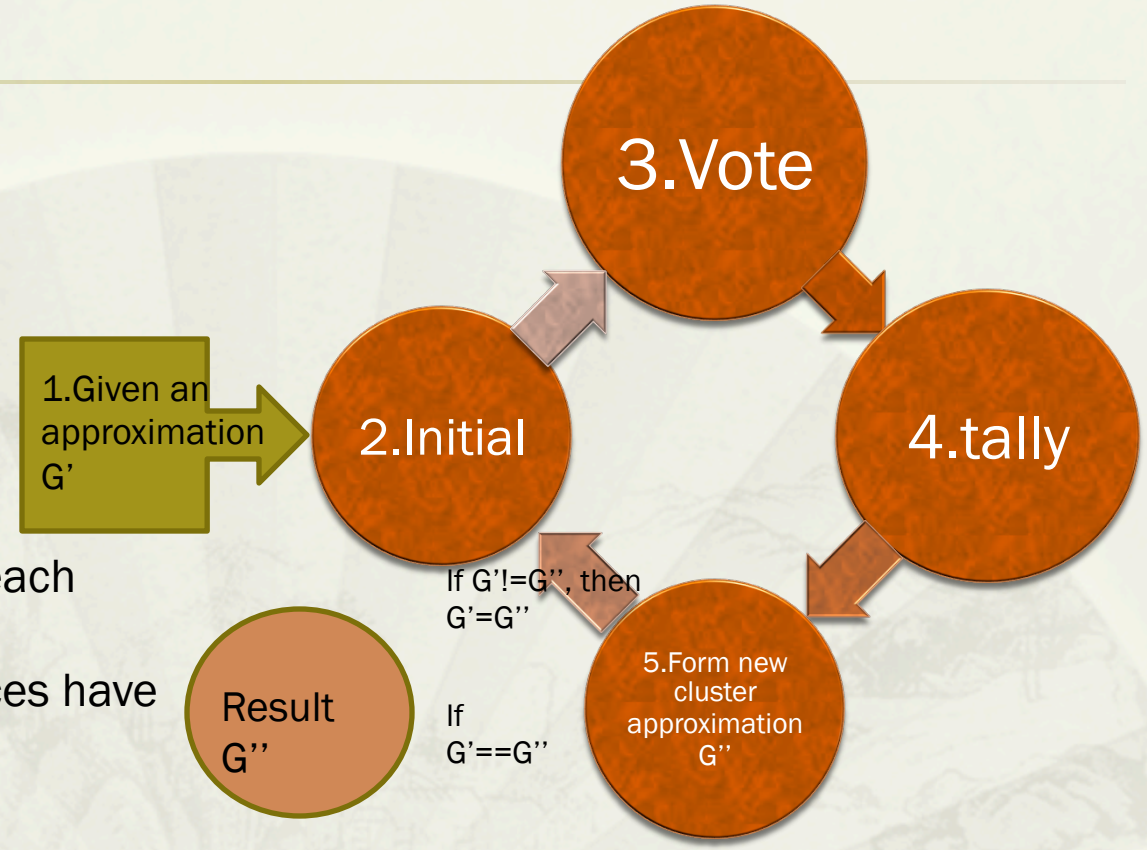
3 $\mathrm{m} = \mathrm{T}$ max.

4 $\mathrm{C_f} = \mathrm{m} .== \mathrm{T}$

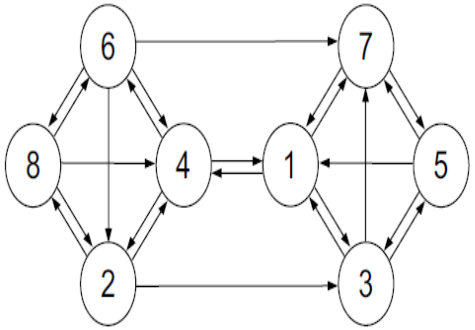
5 if $\mathrm{C_i} == \mathrm{C_f}$

6 then return $\mathrm{C_f}$

7 else return PeerPressure($G, \mathrm{C_f}$)



1. Starting approximation G': each vertex is a cluster.
2. Initialization: assuring vertices have equally votes.
3. Vote: Each node vote for its neighbors.
4. Tally: (1) normalize. (2) Settling ties: what to do if two clusters tie for the maximum number of votes for a vertex.
5. Form a new approximation G''



V	Out-degree
1	4
2	4
3	4
4	4
5	4
6	5
7	3
8	4

(a) The object graph G

$$A = \begin{pmatrix} 0.25 & 0 & 0.25 & 0.25 & 0 & 0 & 0.25 & 0 \\ 0 & 0.25 & 0.25 & 0.25 & 0 & 0 & 0 & 0.25 \\ 0.25 & 0 & 0.25 & 0 & 0.25 & 0 & 0.25 & 0 \\ 0.25 & 0.25 & 0 & 0.25 & 0 & 0.25 & 0 & 0 \\ 0.25 & 0 & 0.25 & 0 & 0.25 & 0 & 0.25 & 0 \\ 0 & 0.2 & 0 & 0.2 & 0 & 0.2 & 0.2 & 0.2 \\ 0.33 & 0 & 0 & 0 & 0.33 & 0 & 0.33 & 0 \\ 0 & 0.25 & 0 & 0.25 & 0 & 0.25 & 0 & 0.25 \end{pmatrix}$$

(b) The adjacency A of G after initialization

$$C = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

(c) Temporary results of matrix C

$$C_f = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(d) Temporary matrix C after 1st tires-settling

Fig.1 The procedure when applying algorithm 2 for object graph G.

5.3 Parallel PPCL algorithm based on linear algebra

Algorithm 3:

Input : a matix-based tageted graph $A : R_{+}^{N\times N}$ and a matrix-based initial approximation garph $C_i : B^{N\times N}$.

Output : a matrix-based clustering result.

procedure PeerPressure($G = A : R_{+}^{N\times N}, C_i : B^{N\times N}$)

```
1   SpParMat<unsigned,double,SpDCCols,<unsigned,double>>  A, C;
    /*reduce to Row,columns are collapsed to single entries*/
2   DenseParVec<unsigned,double>  rowsums = A.Reduce(Row,plus<double>);
    /*multinv < double > function is user defined for double*/
3   rowsums.Apply(multinv<double>);
    /*nomalize A,scale each Column with given vector*/
4   A.DimScale(Row, rowsums);
5   while  C != T  do
        /*vote*/
6       SpParMat<unsigned,double,SpDCCols,<unsigned,double>>   T = SpGEMM(C,A);
        /*renomalize T*/
7       Renomalize(SpParMat<unsigned,double,SpDCCols,<unsigned,double>>  &T);
        /*settling ties*/
8       settling_ties(SpParMat<unsigned,double,SpDCCols,<unsigned,double>>  &T);
```

Initialization

Vote

Normalization

Settling ties

5.4 Parallel PPCL implementation on CombBLAS

- Data distribution and storage
  - DCSC storage structure
- Algorithm expansion & MPI implementation
  - Parallel voting
  - Renormalization
  - Parallel ties-settling

5.4.1 Data distribution and storage

- Distribute the sparse matrices on a 2D <Pr, Pc> processor grid.
  - Processor P(I,j) stores the sub-matrix Aij of dimensions (m/Pr)x(n/Pc) in its local memory.
- HyperSparseGEMM operates on O(nnz) DCSC data structure.

(1) DCSC storage structures

A.I	A.J	A.V
6	1	0.1
8	1	0.2
4	7	0.3
2	8	0.4

Fig.1.1. trip description of matrix A

CP	=	1	3	3	3	3	3	3	4	5	5
		↓						↓	↓		
IR	=	6	8					4	2		
NUM	=	0.1	0.2					0.3	0.4		

Fig.1.2. CSC structure of matrix A

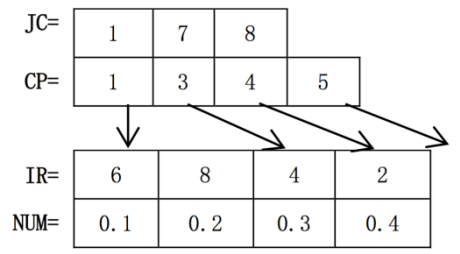
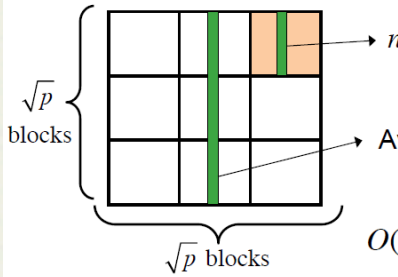


Fig.1.3. DCSC structure of matrix A

CSC or CSR structure

Submatrices are *"hypersparse"* (*i.e. nnz << n*)



$nnz' = \frac{c}{\sqrt{p}} \to 0$

Average of c nonzeros per column

Total Storage:

$O(n + nnz) \to O(n\sqrt{p} + nnz)$

- A data structure or algorithm that depends on matrix dimension n (e.g. CSR or CSC) is asymptotically too wasteful for submatrices
- Use doubly-compressed (DCSC) data structures or compressed sparse blocks (CSB) instead.

DCSC structure

Total Storage :

$O(nnz)$

DCSC is more efficiency than CSR or CSC in storing large scale sparse matrix.

5.4.2 Algorithm expansion & MPI implement.

（1）Voting：sparse SUMMA algorithm，our MPI implementation

SIAM J. SCI. COMPUT.
Vol. 34, No. 4, pp. C170–C191

PARALLEL SPARSE MATRIX-MATRIX MULTIPLICATION AND INDEXING: IMPLEMENTATION AND EXPERIMENTS*

AYDIN BULUÇ† AND JOHN R. GILBERT‡

Abstract. Generalized sparse matrix-matrix multiplication (or SpGEMM) is a key primitive for many high performance graph algorithms as well as for some linear solvers, such as algebraic multigrid. Here we show that SpGEMM also yields efficient algorithms for general sparse-matrix indexing in distributed memory, provided that the underlying SpGEMM implementation is sufficiently flexible and scalable. We demonstrate that our parallel SpGEMM methods, which use two-dimensional block data distributions with serial hypersparse kernels, are indeed highly flexible, scalable, and memory-efficient in the general case. This algorithm is the first to yield increasing speedup on an unbounded number of processors; our experiments show scaling up to thousands of processors in a variety of test scenarios.

（2）Normalization: Reduce() and DimApply() in CombBLAS, our MPI implementation.

（3）Settling ties： both MPI and MPI-OpenMP implementation.

(90% time in the full code)

(1) Parallel voting

* SpGEMM algorithm

Algorithm 3 SpGEMM algorithm

Input: $A \in S^{m \times k}$, $B \in S^{k \times n}$:sparse matrices distributed on a $p_r \times p_c$ processor grid.

Output: $C \in S^{m \times n}$: the product AB ,similarly distributed.

Procedure SparseSUMMA(A,B,C)

1 for all processors $P(i, j)$ in parallel do

2 $B_{ij} \leftarrow (B_{ij})^T$

3 for all q=1 to k/b do *//blocking parameter b evenly divides* k/p_r *and* k/p_c

4 $c = (q \cdot b)/(k/p_c)$ *//the broadcasting processor column*

5 $r = (q \cdot b)/(k/p_r)$ *//the broadcasting processor row*

6 $lcols = (q \cdot b) \bmod c : ((q+1) \cdot b) \bmod c$ *//local column*

7 $lrows = (q \cdot b) \bmod r : ((q+1) \cdot b) \bmod r$ *//local row*

8 $A^{rem} \leftarrow Broadcast(A_{ic}(:, lcols), P(i,:))$

9 $B^{rem} \leftarrow Broadcast(B_{rj}(:, lrows), P(:, j))$

10 $C_{ij} \leftarrow C_{ij} + HyperSparseGEMM(A^{rem}, B^{rem})$

11 $B_{ij} \leftarrow (B_{ij})^T$ // Restore the original B.

(2) Renormalization

* Leverage two primitives in CombBLAS to assure the elements in matrix T remain to 1 or 0.

Algorithm 4 Renormalize the T matrix

```
Renormalize(SpParMat &T)
    DenseParVec colmax = T.reduce(Column, max);
    T.DimApply(Column, colmaxs, equal_to<double>).
```

(3) Settling ties

- ✓ Function: Selecting the lowest numbered cluster with the highest number of votes.

* Algorithm 5.

```
Settling_ties(SpParMat<unsigned,double,SpDCCols,<unsigned,double>> &T)
1   for all processors P(i, j) in parallel do
      /*creat an vector tallying the processor number */
2     vector<int> v = T.CreatVec();
      /*do MPI _ Allreduce on every processor column*/
3     MPI_Allreduce(v, min_v, k, MPI_INT, MPI_MIN, Colworld());
      /*generate a new clustering result matrix*/
4     T.PruneMat(min_v);
```

Recording

Communication

Selection

The for…in parallel do construct indicates that all of the docode blocks execute in parallel by all the processors. In line 2, constructing a vector where every element is corresponding to each column in T matrix. If the value in some column contain 1, then tallying its processor ID in its corresponding element in the vector, or tallying the maximal integer.

6. Numerical Experiments

- Platform and Input Graph
- Testing CombBLAS performance on Dawning
- Testing our parallel implementation

6.1 Platform and Input Graph

Dawning supercomputer：

- Intel Omni-Path network， 100Gbps two-side connection；
- 172 nodes, each has 24 2.5GHz Intel E5-2680 processor cores with 64GB memory.
- Mvapich2.

Input Graph:

1.A permuted R-MAT graph of scale 16 with self loops added.

- 65,536 vertices, 490,563 edges.

2. A permuted R-MAT graph of scale 21 with self loops added.

- 2,097,152 vertices, 18,305,177 edges.

R-MAT graph：RMAT model is one of graph generation models . It has simple algorithm and its resulting graph fits for the Power-Law distribution

6.2 Testing CombBLAS performance on Dawning

- Using BFS code included in CombBLAS.
- Input:
  - R-MAT graph of scale 17 with self-loops added
  - Contains a sparse matrix of size 131072^2.
- Performance:
  - Speedup: 15 on 64 processors.
  - MTEPS(Mega Traversed Edges per second).

Table 1. MTEPS of BFS code of CombBLAS running on Dawning with different proc. number.

# of processor cores	MTEPS
1	74
4	198
16	406
64	945

The results have not gained good enough performance as predicted. The reason maybe that the CombBLAS version we used is not optimized to match the Dawning supercomputer.

6.3 Testing our parallel implementation

- Case 1: R-mat graph of scale 16
- Case 2: R-mat graph of scale 21

(1) Case 1: R-MAT with scale 16

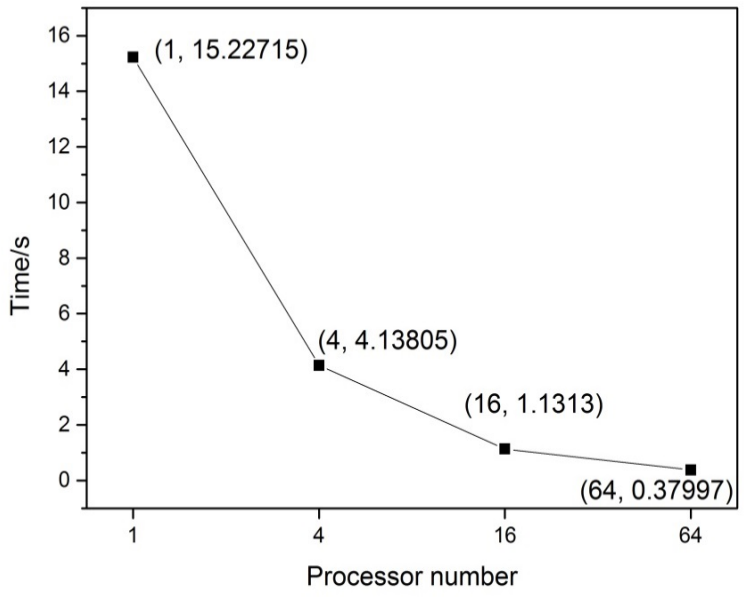


Fig.2. Time vs. processor number

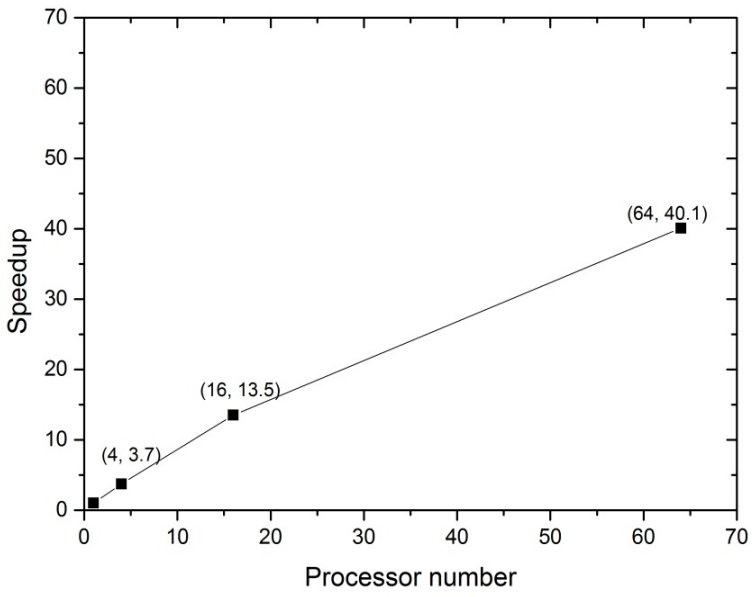


图3. Speedup vs. processor number

Parallel efficiency on 4 processors：92.5%

...

Parallel efficiency on 64 processors：62.7%

(2) Case 2: R-MAT of scale 21

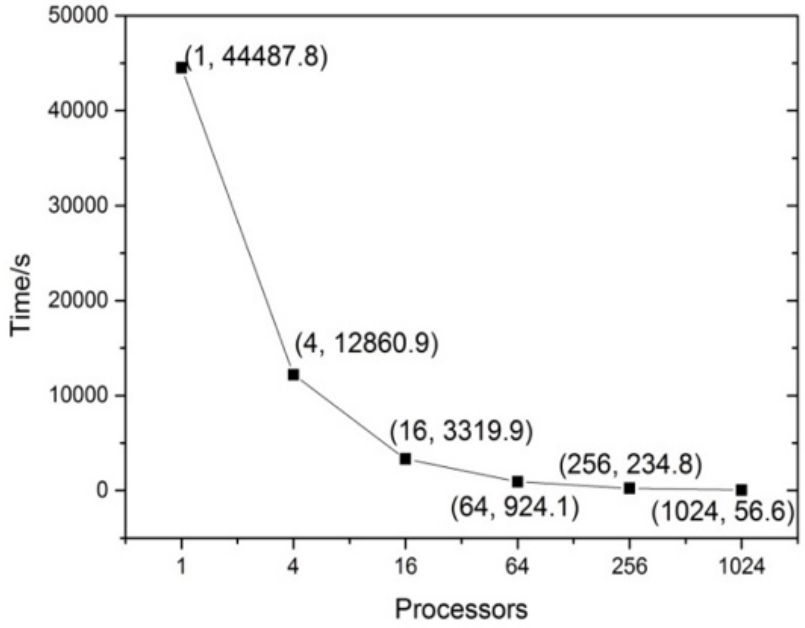


Fig. 4.Time vs. processor numbers

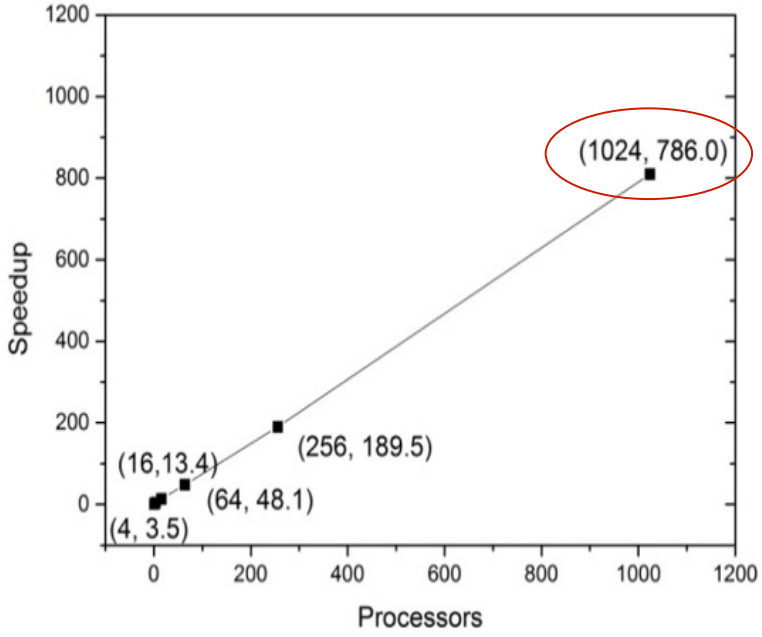


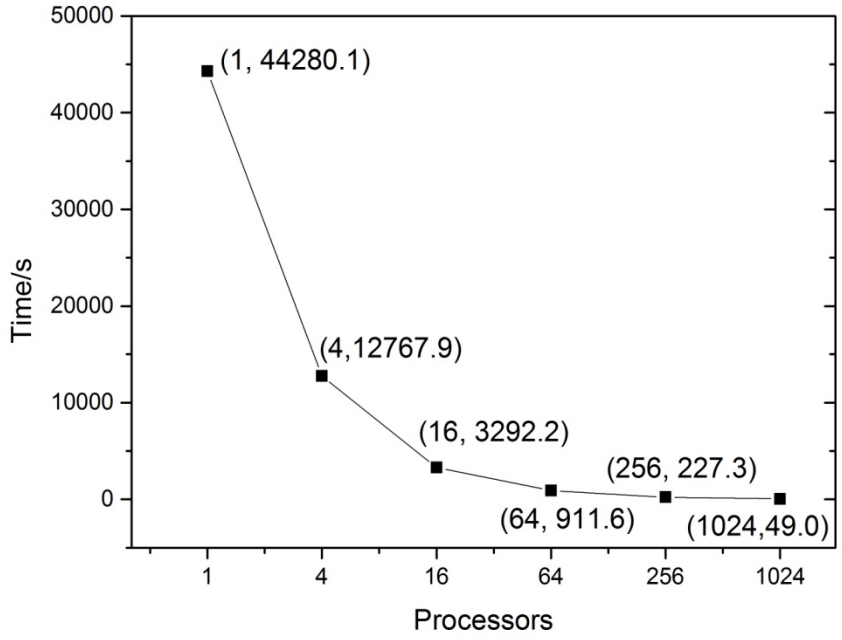
Fig.5.Speedup vs. processor numbers

E on 4 processors：87.5%

…

E on 1024 processors：75%

Settling ties performance: R-MAT of scale 21



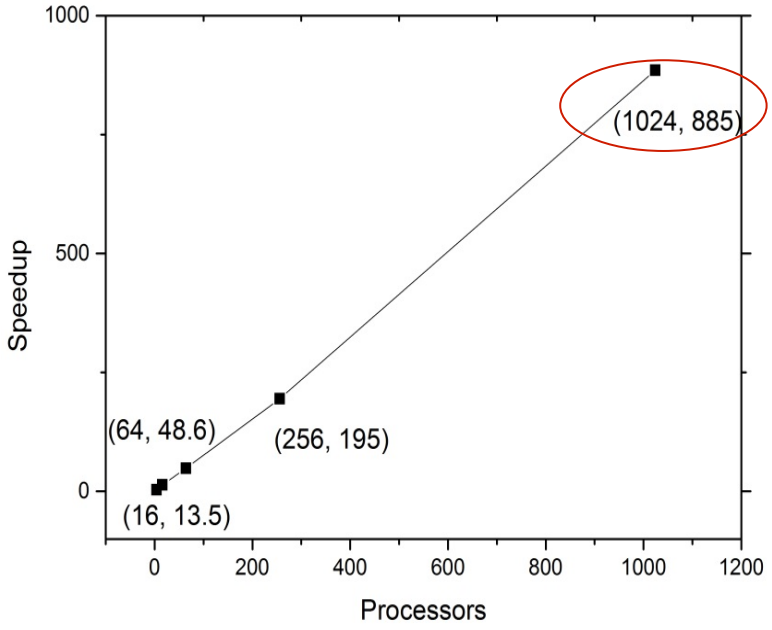


Fig. 8. MPI implementation： （left）parallel time.（right）speedup

E on 4 processors：87.5%

…

E on 1024 processors：86.4%

From the test…

- The sparse degree of the input graph
  - All the two input graphs have highly sparse degree>99%. (=(# of zeros)/(# of total elements))
  - Also high irregular degree.
- Scalability of our algorithm
  - Under the same conditions besides the size of graph and sparse degree, the speedups of the small and large graphs we test are not much difference. It also proves the scalability of the algorithm.

7. Discussions

- Good scalability of our parallel algorithm.
- Limit on the core numbers: mxm
  - # of columns = # of rows.
  - Current version uses even distribution between processors.
- Settling ties step consumes most of the time
  - Reason: all-to-all communication between processors.

- BFS testing on Dawning indicates optimizing the relative algebra operations in CombBLAS for object supercomputer is needed.

- There are not so many research results about parallel PPCL algorithm on large computation. Our implementation has gained high performance on more than thousands processors for large scale graph.

- Our implementation VS. direct parallelization
  - It is applicable to provide direct parallelization of PPCL algorithm
  - But the tedious treatment process for irregular data structure of graph can not ignored. It also causes a large amount of time costs and memory occupations.
  - We translated the graph computations into a series of matrix operations, and the computations on the irregular data structures of graph can be transformed to be structural representation based on sparse matrix. Thus high performance can be achieved.

Conclusion

- We design and implement parallel PPCL algorithm based on CombBLAS, which representing the PPCL algorithm as sparse matrix computations.
- When the input is a permuted R-MAT graph of scale 21 with self loops, the MPI implementation achieves up to 809.6x speed up on 1024 cores of Dawning supercomputer.
- Next work includes:
  - Optimizing the performance of parallel PPCL implementation and testing larger graph for high scalability
  - Optimizing the performance of relative matrix algebra building blocks in CombBLAS to match the under computer architecture.
  - Studying more parallel irregular algorithms presented by sparse matrix operations.

THANKS!