Fast Distributed Graph Partition and Application

Bilel Derbel (LaBRI, Bordeaux 1),

joint work with

Mohamed Mosbah
Akka Zemmari

28 April 2006
IPDPS06
Outline

- Introduction and motivation
- A semi-sequential algorithm: Basic_Part
- A fully distributed algorithm: Dist_Part
- Sublinear algorithms
  - deterministic algorithm: Fast_Part
  - randomized algorithm: Elect_Part
- Case study
- Applications
- Conclusion
Introduction

- Huge networks
  - Non-efficiency of traditional protocols
  - Huge information requirements: Memory
  - Information maintaining cost: Time

- A global knowledge is not always essential
  - Some network tasks have a local nature

- Locality-Preserving (LP)-representation [D. Peleg]
  - An efficient data structure that captures some topological properties
LP-representation

Skeletal Representations
- Spanning trees
- Graph spanners

Clustered Representations
- Cluster: possibly overlapping connected subset
- Locality: radius
- Sparsity: overlap or interaction

Goals
- Find efficient distributed algorithms
  - improve the complexity of related applications
The model

A network of $n$ processes is modeled by a simple unweighted undirected graph $G = (V, E)$

A node performs only local computations
- communication: Message Passing System
- no failure

Each node has a unique identity

Time complexity
- synchronous case: a global clock
- asynchronous case: no global clock
- negligible time for local computations
Basic algorithm

Set $S \leftarrow \emptyset$

while $V \neq \emptyset$ do
    Select an arbitrary vertex $v \in V$
    Set $S = \{v\}$
    while $|\Gamma(S)| > n^{1/k} |S|$ do
        $S \leftarrow \Gamma(S)$
    end while
    Set $S \leftarrow S \cup S$ and $V \leftarrow V - S$
end while

return $S$

$\Gamma(S) = \bigcup_{v \in S} N(v)$; with $N(v)$ : the neighborhood of $v$
Basic algorithm

Set $S \leftarrow \emptyset$

while $V \neq \emptyset$ do

  Select an arbitrary vertex $v \in V$

  Set $S = \{v\}$

  while $|\Gamma(S)| > n^{1/k}|S|$ do

    $S \leftarrow \Gamma(S)$

  end while

  Set $S \leftarrow S \cup S$ and $V \leftarrow V - S$

end while

return $S$

- $S$ is a partition
- $Rad(S) \leq k - 1$ (locality)
- The graph induced by the clusters has at most $n^{1 + \frac{1}{k}}$ edges (sparsity)
Related Works

[Awerbuch 85] Complexity of Network Synchronization, J. ACM

[Awerbuch et al 89] Network Decomposition and Locality in Distributed Computation, FOCS

[Awerbuch & Peleg 90] Sparse Partitions, FOCS


[Moran & Snir 00] Simple and Efficient Network Decomposition and Synchronization, TCS

[Peleg 00] Distributed Computing : A Locality-Sensitive Approach, SIAM MDMA

...
Basic implementation

Semi-sequential

- The clusters are constructed one by one
- One cluster is constructed in a distributed way
- **Step 1**: find a potential cluster center,
- **Step 2**: grow a new cluster,
- **Step 3**: repeat Steps 1 and 2 until there are no potentials centers.

Problem

- Next center election
改善 [MS 00]

主要思想
- 一棵根的生成树 $T$ 从 $G$ 构造
- 所有节点标记为潜在领导者
- 第一个领导是 $T$ 的根
- 下一个领导者由执行 $T$ 的 DFS 遍历来找到
  - 当 $v$ 是新领导时 DFS 搜索停止
- 当 $v$ 的簇的构造完成时 DFS 搜索重新开始

复杂度
- $\mathcal{O}(|V|)$ 时间和 $\mathcal{O}(|E|)$ 消息
A fully distributed algorithm

- We can construct two clusters in parallel and in a concurrent way.

- There is no need to wait until the termination of a cluster construction.

- Two clusters in two $2^k$-separated regions could be constructed in parallel.
A fully distributed algorithm: $\text{Dist}_\text{Part}$

- At the beginning, each node forms a single cluster
- The identity of a cluster is the identity of its center (root) node
- Each cluster grows in a layered fashion
- Clusters have to manage conflicts between each others
Algorithm $Dist\_Part : \text{concurrency}$

Cluster $S_1$

Cluster $S_2$

Cluster $S_3$

Cluster $S_4$
Algorithm *Dist Part* : Main idea

1. The Exploration (Attack) Rule
   - A cluster can explore a new layer *iff* it has the biggest identity at distant 2
   - A node can not be explored by more than one cluster
Algorithm $Dist_{Part}$: Main idea

1. The Exploration (Attack) Rule

- A cluster can explore a new layer iff it has the biggest identity at distant 2
- A node can not be explored by more than one cluster

2. The Growth Rule

- A cluster grows iff the sparsity condition is satisfied.
- If the sparsity condition is satisfied, then a new Exploration Rule is applied
- Otherwise, the construction of the cluster is finished and the last layer is rejected.
Algorithm \textit{Dist Part} : Main idea

3. The Battle Rule

- If a cluster is invaded, then it loses all of its last layer
- A cluster is invaded if a neighboring cluster successfully applies an Exploration Rule
- A cluster which loses an exploration is not automatically invaded
Algorithm $Dist\_Part : Main\ idea$

3. The Battle Rule

- If a cluster is invaded, then it loses all of its last layer
- A cluster is invaded if a neighboring cluster successfully applies an Exploration Rule
- A cluster which loses an exploration is not automatically invaded

The cluster with the biggest identity always succeeds in applying the Exploration Rule
Algorithm \textit{Dist\_Part} : Example

\[Id_1 > Id_2 > Id_3 > Id_4 > Id_5\]
Algorithm $\text{Dist\_Part}$: Example

$Id_1 > Id_2 > Id_3 > Id_4 > Id_5$
Algorithm $\textit{Dist\_Part}$ : Example

$Id_1 > Id_2 > Id_3 > Id_4 > Id_5$
Algorithm $\text{Dist Part}$: Example

$Id_1 > Id_2 > Id_3 > Id_4 > Id_5$
Algorithm $Dist_{Part}$: Example

$Id_1 > Id_2 > Id_3 > Id_4 > Id_5$
Algorithm $Dist_{Part}$: Example

$Id_1 > Id_2 > Id_3 > Id_4 > Id_5$
**Algorithm** \( Dist_{Part} : \) summary

\[
\begin{align*}
\text{continue} & \leftarrow \text{True} \\
\textbf{while} & \hspace{1em} \text{continue} \hspace{1em} \textbf{do} \\
& \hspace{1em} \text{execute the} \hspace{1em} \textit{Exploration Rule} \\
& \hspace{1em} \textbf{if} \hspace{1em} \text{success of the} \hspace{1em} \textit{Exploration Rule} \hspace{1em} \textbf{then} \\
& \hspace{2em} \text{add the new layer} \\
& \hspace{2em} \text{execute the} \hspace{1em} \textit{Growth Rule} \\
& \hspace{2em} \textbf{if} \hspace{1em} \text{Non success of the} \hspace{1em} \textit{Growth Rule} \hspace{1em} \textbf{then} \\
& \hspace{3em} \text{reject the last explored layer} \\
& \hspace{3em} \text{switch to a finished cluster} \\
& \hspace{3em} \text{continue} \leftarrow \text{False} \\
& \hspace{1em} \textbf{end if} \\
& \hspace{1em} \textbf{else} \\
& \hspace{2em} \text{execute the} \hspace{1em} \textit{Battle Rule} \\
& \hspace{1em} \textbf{end if} \\
\textbf{end while}
\end{align*}
\]
Algorithm $Dist\ Part$ : Implementation

- A node can be in five states
  - $root$ : takes global decisions for its cluster
  - $relay$ : forwards informations
  - $leaf$ : fights for new regions
  - $orphan$ : single node cluster
  - $final$ : belongs to a final cluster

- A BFS tree is constructed for each cluster
  - An efficient structure for node communications
Algorithm $Dist\_Part : Implementation$

- Broadcast from the root
  - $NEW$ message
  - $BACK$ and $STOP$ messages
  - $UP$ and $FAILURE$ messages
  - $OK$ and $DOWN$ messages

- convergecast from the leaf to the root
  - $YES$, $NO$ and $STOPPED$ messages
  - $BYE$, $OK$ messages
  - Computation of the sparsity condition

- Election technique in a ball of radius 2 to execute a new exploration
Algorithm $Dist_{Part}$: Correctness

Correctness

- A cluster grows in a layered fashion
- A cluster loses the whole last layer in case of neighbor invasions
- The sparsity condition is verified by the root at each step
Algorithm \textit{Dist\_Part} : Correctness

Correctness

- A cluster grows in a layered fashion
- A cluster loses the whole last layer in case of neighbor invasions
- The sparsity condition is verified by the root at each step

Termination

- A cluster can grow only up to radius $k$
- The cluster having the biggest identity always succeeds a new exploration
- No deadlocks
- The number of active nodes decreases
Algorithm $Dist\_Part$ : Complexity

$O(n)$ in the worst case

- The winner cluster have the biggest (lexicographical order) (Radius,ID)
Algorithm $Dist\_Part$ : Complexity

$O(n)$ in the worst case

- The winner cluster have the biggest (lexicographical order) (Radius,ID)

Remarks

- No election step
- No pre-processing
- How many clusters are constructed in parallel ?
  - $> 1$ in practice !!
- Experimentation with ViSiDiA
Algorithm \textit{Fast Part} : Main idea

- Nodes having low degrees will always form a cluster of radius 0
- Give a high priority to nodes with high degree
- Two types of nodes
  - \( v \) is dense, if \( d(v) > n^{\frac{1}{k}} \)
  - \( v \) is sparse, if \( d(v) \leq n^{\frac{1}{k}} \)
Algorithm *Fast Part*: Main idea

### New Rules

- A sparse node looses against a dense node
- A sparse node with a sparse neighborhood becomes finished
- A sparse node which is not invaded becomes finished

### Implementation:

- We use a couple (ID,bool)

  *If* a sparse node becomes finished, *then* it informs its neighbors.
**Algorithm Fast Part**: analysis

- **Correctness**:
  - A cluster can not add more than \( k - 1 \) layers
  - A node is explored by at most one cluster
  - The sparsity condition is always verified

- **Time complexity**:
  \[ O(k^2 \Lambda) = O(k^2 n^{1 - \frac{1}{k}}) \]
  - \( \Lambda \): the number of clusters with radius \( \geq 1 \)
  - In the worst case, the construction of a cluster of radius \( \geq 1 \) is \( O(k^2) \)
  - In the worst case, the clusters are constructed one by one
  - The cluster with radius 0 are 0 time consuming
Algorithm $Fast_{Part}$: Remarks

- Time complexity: $O(n^{1-\frac{1}{k}})$
  - A cluster of radius $l \rightarrow n^{\frac{l}{k}}$ nodes
  - A cluster of radius $l \rightarrow O(l^2)$ time consuming
  - $O(1)$ time $\rightarrow n^{\frac{1}{k}}$ nodes

- remarks:
  - The algorithm privilege the decomposition of dense regions
  - The complexity is better when dense regions are far away each others
  - For particular graphs ($\Lambda$), we obtain better performances
  - The analysis is still sequential.
Algorithm $Elect_{Part}$: preliminary

- randomized procedure [MSZ02]
- LE2: relabeling of disjoint closed stars

- degree of parallelism of the algorithm

In the case $k = 2$, if a node is elected, then
1. It computes the number of its active neighbors,
2. It decides to be a finished cluster with radius 0 or 1
3. It terminates
Algorithm \textit{Elect\_Part} : Algorithm $LE_k$

Generalization for $k \geq 2$

\begin{verbatim}
Round ← 0;
while Round < k do
    execute the \textit{Exploration Rule};
    Round ← Round + 1;
    if Non Success of the \textit{Exploration Rule}
    then
        execute the \textit{Battle Rule};
    end if
end while
\end{verbatim}
while There exist nodes not in a finished cluster do
    (0.) each node selects randomly an identity from a big set of integers.

Stage 1 : local election in balls of radius $k$
    (1.a) Each node $v$ not in a finished cluster runs algorithm $LE_k$.

Stage 2 : reinitialization
    (2.a) Each formed cluster $S$ computes independently the sparsity condition for each layer $j \leq k$,
        if $S$ contains a layer $j$ violating the sparsity condition then
            (2.b) $S$ releases all layers $l \geq j$ and becomes a finished cluster,
            (2.c) nodes in released layers become single-node clusters.
        else
            if all neighbors are finished then
                (2.d) $S$ becomes finished.
            end if
        end if
    end if
(2.e) Break all non finished clusters and form new single-node clusters.
end while
Algorithm *Elect_Part* : analysis

Correctness : OK

Complexity

- we define $K$ such that: $\forall v \in V, |N_{2k}(v)| \leq K$

- The expected number of $k$-elected nodes is upper bounded by $|V|/K$

- By induction, we prove that the time complexity $T$ verifies:

$$E(T) = \mathcal{O} \left( k^2 \frac{\log(n)}{\log\left(\frac{K}{K-1}\right)} \right)$$
Case study: Circulant graph

We consider a circulant graph $C_n(1, 2, \ldots, \left\lfloor \frac{n^c}{2} \right\rfloor)$.

Example of circulant graphs
- $C_8(1)$, $C_8(1, 2)$, $C_8(1, 2, 3)$
Case study: **Circulant graph**

- We consider a circulant graph $C_n(1, 2, \ldots, \lfloor \frac{n^\epsilon}{2} \rfloor)$.

**Easy consequence**

- Complexity of $Fast\_Part : T = \mathcal{O}(n^{1-\epsilon})$
- Complexity of $Elect\_Part : E(T) = \mathcal{O}(k^3 \log(n) n^\epsilon)$

**Detailed analysis**

- Complexity of $Elect\_Part :$
  \[
  E(T) = \mathcal{O}(k^3 \log(n) + kn^\frac{1}{k})
  \]

**Box**

\[
E(T) = \mathcal{O}(k^3 \log(n))
\]
Application to graph spanners

- A $(\alpha, \beta)$-spanner of $G$ is a subgraph $H$ such that
  \[ \forall u, v \in V, d_H(u, v) \leq \alpha.d_G(u, v) + \beta \]

- Quality of a spanner
  - size and stretch
  - time: *the locality of the problem*
Application to graph spanners

Using algorithm $Basic\_Part$, a $(4k - 3, 0)$-spanner with $\mathcal{O}(n^{1+\frac{1}{k}})$ edges can be constructed deterministically.

- construct a BFS tree for each cluster
- add an intercluster edge between each two neighboring clusters
Application to graph spanners

- Using algorithm $Basic\_Part$, a $(4k - 3, 0)$-spanner with $O(n^{1 + \frac{1}{k}})$ edges can be constructed deterministically.
- Construct a BFS tree for each cluster.
- Add an intercluster edge between each two neighboring clusters.
Application to graph spanners

- Using algorithm \( Basic\_Part \), a \( (4k - 3, 0) \)-spanner with \( O(n^{1 + \frac{1}{k}}) \) edges can be constructed \textit{deterministically}
- construct a BFS tree for each cluster
- add an intercluster edge between each two neighboring clusters

![Diagram](image)
Application to graph spanners

- Each cluster constructs a BFS tree
- The last rejected layer is also spanned
  - the last rejected layer contains at most $n^{1/k} \cdot |S|$ nodes
Application to graph spanners

- each cluster constructs a BFS tree
- the last rejected layer is also spanned
- the last rejected layer contains at most $n^{1/k} \cdot |S|$ nodes
Application to graph spanners

- Using algorithm $Fast_Part$, a $(2k - 1, 0)$-spanner with $O(n^{1+\frac{1}{k}})$ edges can be constructed $\textit{deterministically}$ in $O(n^{1-\frac{1}{k}})$ time.

$$s \leq (k - 1) + (k - 1) + 1$$
Application to synchronizers:

Synchronizers

- Simulate synchronous distributed algorithms in an asynchronous setting.
- Synchronizers $\alpha$, $\beta$ and $\gamma$
Application to synchronizers:

Synchronizers

- Simulate synchronous distributed algorithms in an asynchronous setting.
- Synchronizers $\alpha$, $\beta$ and $\gamma$
Conclusion

- Efficient distributed algorithms
- More efficient in practice
- Can we improve the time complexity?
  - Improved randomized solutions?
- Lower bound?
  - Assume unlimited message size
  - Break the symmetry: What information is necessary?
Thank You!

Questions?