Efficient Parallel Graph Algorithms for Coarse-Grained Multicomputers and BSP

E. Dehne, A. Ferreira, E. Caceres, S.W. Swong, and A. Roncato

Topics for Discussion

- Parallel Computing Models (PRAM, BSP, CGM)
- Developing Deterministic parallel algorithms under the CGM & BSP models

Goal: Solving well-known graph problems such as,

1. List ranking
2. Euler tour construction in a tree
3. computing the connected components and spanning forest
4. Lowest common ancestor preprocessing
5. Tree contraction and expression tree evaluation
6. Computing an ear decomposition or open ear decomposition
7. 3-edge connectivity and bioconnectivity

Parallel Random Access Machine (PRAM) Model

- A **Parallel Random Access Machine** (**PRAM**) is an abstract machine for designing the algorithms applicable to parallel computers. It eliminates the focus on miscellaneous issues such as synchronization and communication, but lets designer think explicitly about the exploitation of concurrency. In terms of Flynn's taxonomy, PRAMs are *multiple instruction multiple data (MIMD)* computers.
- MIMD: Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space.

Bulk Synchronous Parallel Model (BSP model)

- Introduced by Valliant in 1990; Its initial goal was to simulate PRAM algorithms on the BSP model.
- It was one of the first models that took communications issues into account and abstracted the features of a parallel machine in a few parameters.

Bulk Synchronous Parallel Model (BSP model)

- A BSP machine consists of p processors/memory connected by a router that can deliver messages in a point to point fashion between the processors.

- A BSP computer has the following parameters:

1. *N* refers to the problem size,
2. *p* is the number of processors,
3. *L* is the minimum time between synchronization steps (measured in basic computation units), and
4. *g* is the ratio of overall system computational capacity (number of computation operations) per unit time divided by the overall system communication capacity (number of messages of unit size that can be delivered by the router) per unit time.

Bulk Synchronous Parallel Model (BSP model)

- The BSP approach requires that *g* (= local computation speed / router bandwidth) is low, or fixed, even for increasing number of processors.
- Valiant described circumstances where PRAM simulations cannot be performed efficiently, if the factor *g* is high.
- Unfortunately, this is true for most currently available multiprocessors.

Bulk Synchronous Parallel Model (BSP model)

- BSP algorithms consists of a sequence of supersteps. In each superstep the processors operate independently performing local computations and global communications by send and receive operations.

- The sent messages can be assumed to be received in the next superstep.

- At the end of a superstep a barrier synchronization is realized. The time for a BSP algorithm is the sum of the times for the supersteps.

- Each superstep is charged by max{L,x+hg} where x is the maximal number of local computations on every processor and h is the maximal number of packets that any processor sends or receives in the superstep.

Bulk Synchronous Parallel Model (BSP model)

- For clarity a superstep is often divided into a computation and a communication superstep.
- In *computation supersteps* the processors perform computations on data that was present locally at the beginning of the superstep.
- In *communication supersteps* data is exchanged among the processors via the router.
- This postponing of communications to the end of a superstep is the key idea in BSP. It removes the need to support *non*-barrier synchronizations between processes, and guarantee that processes within a superstep are *mutually independent*.

Coarse-grained multicomputer (CGM) model

- The Coarse Grained Multicomputer CGM model was proposed by Dehne et. al in 1993.
- It is comprised of a set of *p* processors, $P\_1,\ldots,P\_p$ with $O(N/p)$ local memory per processor and an arbitrary communication network (or shared memory).
- (*p* = no. processors, *N* = total input size).

Coarse-grained multicomputer (CGM) model

- The local memory will typically be considerably lager than O(1), (i.e. N/p >= p).
- In practice, we assume that N>>p. More precisely, we assume that $N / p \geq p \wedge \varepsilon$, for some fixed e>0.
- This feature gives the model its name “coarse grained”.

Coarse-grained multicomputer (CGM) model

- A CGM(N,p) uses only two parameters, N and p, and assumes a collection of p processors with N/p local memory each connected by a router that can deliver messages in a point to point fashion.
- A CGM algorithm consists of an alternating sequence of *computation rounds* and *communication rounds* separated by barrier synchronizations.

Coarse-grained multicomputer (CGM) model

- Each communication round consists of routing a single *h*-relation with h=O(N/p), i.e., each processor sends *O(N/p)* data and receives *O(N/p)* data.
- We require that all information sent from a given processor to another processor in one communication round is packed into one long message, thereby minimizing the message overhead ("packing requirement").
- The time for a CGM algorithm is the sum of the times for the computation and communication rounds.

Relation between the BSP and CGM models

- Compared to the BSP model a computation round in the CGM model is equivalent to a superstep in the BSP model with L=g (N/p).
- Therefore the CGM model is slightly more powerful than the BSP model.
- A communication round consists of a single h-relation with h < N/p.
- The cost of each communication round has the same value referred as H_N,p.
- Therefore, the total communication cost is simply T_comm=L * H_N,p.

Relation between the BSP and CGM models

- The main difference between the BSP and CGM models is that the latter allows only one single type of communication operation, the *h*-relation, and simply counts the number of *h*-relations as its main measure of communication cost.

- Note that every CGM algorithm is also a BSP algorithm but not vice versa.

- The main advantage of the CGM model is that it allows us to model the communication cost of a parallel algorithm by one single parameter, the number of communication rounds, L.

Relation between the BSP and CGM models

- A few extremely large messages may require more time than a larger number of very small messages. Therefore, it is also useful to study the total message size per processor over all rounds, i.e., the sum of the sizes of all message sent by a processor during the entire computation.
- For the algorithms that follow, the total message size per processor over all rounds is O(log(p)(N/p)).

- **Note:** *A CGM algorithm with L rounds and computation cost T_*comp *corresponds to a BSP algorithm with L supersteps*, *communication cost O(gL(N/p))*, *and the same computation cost T_*comp.

List Ranking

- Let *L* be a linked list of length *n* represented by a vector s[1,…,n]. For each i in {1,…,n}, s[i] is the pointer to the list element following i in the list L. We refer to i and s[i] as s-neighbors. The last element λ of the list L is the one with s[λ]=λ. The distance between i and j, d_L(i,j), is the number of nodes in L between i and j plus one (i.e., the distance is zero if and only if *i* = *j* , and it is one if and only if one node follows the other).

- The ***list ranking problem*** consists of computing for each i in L the distance between i and λ, referred to as rank_L(i)=d_L(i,λ).

List Ranking

- On a CGM(n,p), each processor initially stores n/p elements of L with their respective pointers.
- Figure 1, shows an example of a linked list with n=24 elements stored on a CGM with p=4 processors, n/p=6 elements per processor.

List Ranking

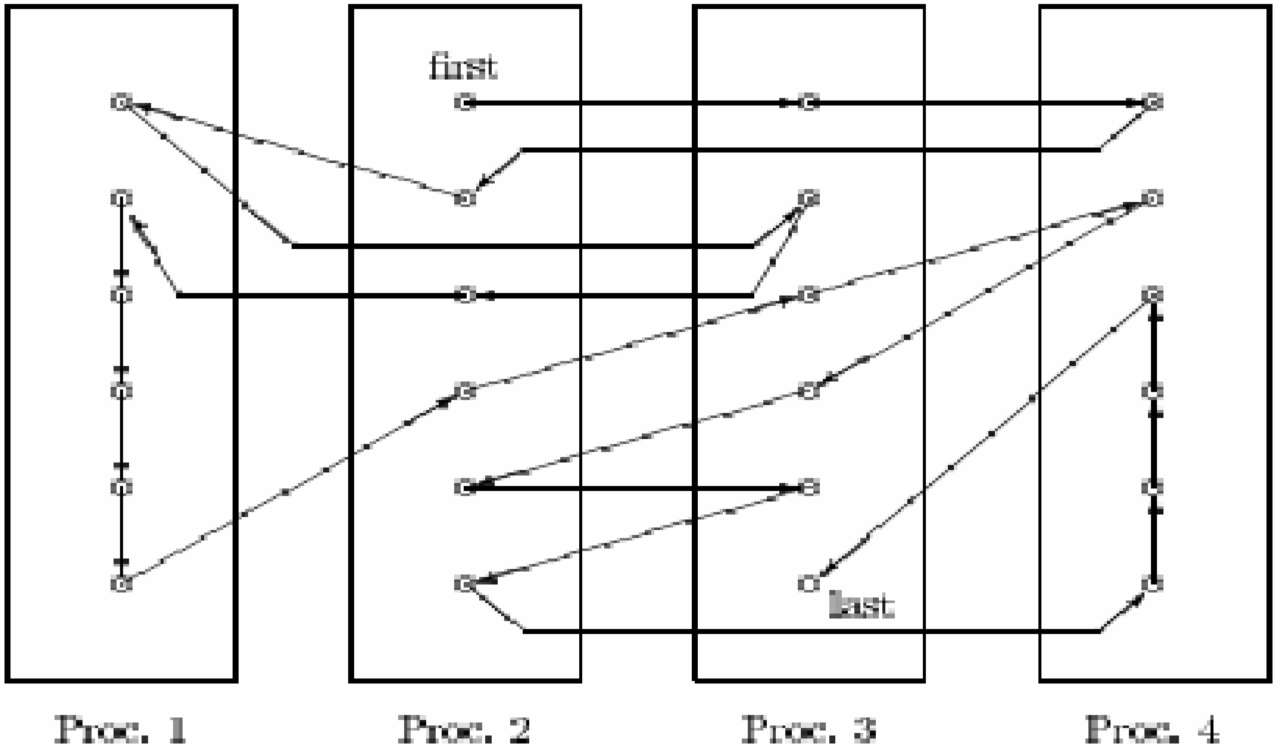


Fig. 1. A linked list L stored on a $CGM(n, p)$.

List Ranking

- The following definition is needed;

- Definition: An *r -ruling set L′* of *L* is defined as a subset of *selected* list elements of *L* that has the following properties:

1. No two neighboring elements.
2. The distance of any unselected element to the next selected element is at most r. For each i in L let s_L′[i] be the next selected element j in L′ with respect to the order implied by L.

List Ranking

- We represent an r-ruling set L’ again as a linked list where each element i in L’ is assigned a pointer to s_L’[i] and a value w(i)=d_L(i, s_L’[i]).

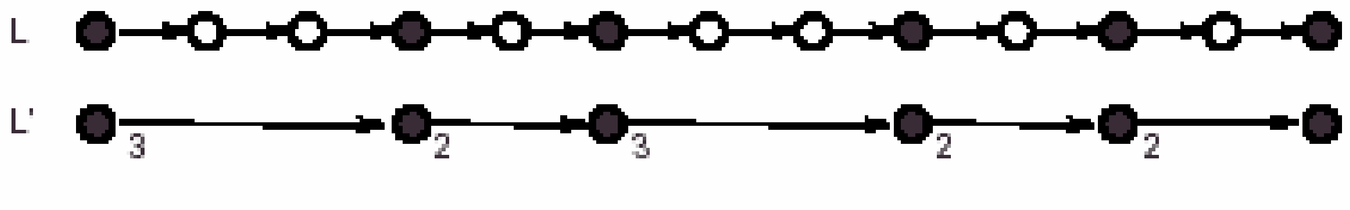


Fig. 2. A list L and a 3-ruling set L'.

CGM List Ranking

- We give a deterministic CGM list ranking algorithm which requires O(logp) rounds.
- It is important to note, that the rounds of this parallel algorithm are **independent** of the problem size, and grows only logarithmically with respect to *p*.

CGM List Ranking

- **Algorithm 1.** CGM List Ranking

- **Input**: A linked list L of length n stored on a CGM(n,p), $N / p \geq p \wedge \varepsilon$
  Each processor stores n/p list elements i in L and their respective pointers s[i].

- **Output**: For each list element i its rank rank_L(i) in L.

CGM List Ranking

- **Implementation.**
- **Step 1**: The CGM computes an O(p^2)-ruling set R of size |R|=O(n/p). (Given via algorithm 2).
- **Step 2**: R is broadcast to all processors. This broadcast is implemented as O(logp) communication rounds where the number of processors storing R is initially one and then doubled in each communication round.

CGM List Ranking

- **Step 3**: Each processor, sequentially performs weighted list ranking on R with weights w( ), thereby computing for each j in R its rank rank_L(j) in L.
- **Step 4**: Each list element i in L – R has at most distance O(p^2), with respect to L, to its next element s_R[i]. Using O(logp) sorting steps, the CGM simulates O(logp) pointer jumping steps of the standard PRAM list ranking algorithm, thereby computing for each i in L – R its distance d_L(i,s_R[i]) to its next element s_R[i] in R.
- **Step 5**: Each processor locally computes the ranks of its list elements i in L – R as follows: rank_L(i)=d_L(i,s_R[i])+rank_L(s_R[i]).

CGM List Ranking

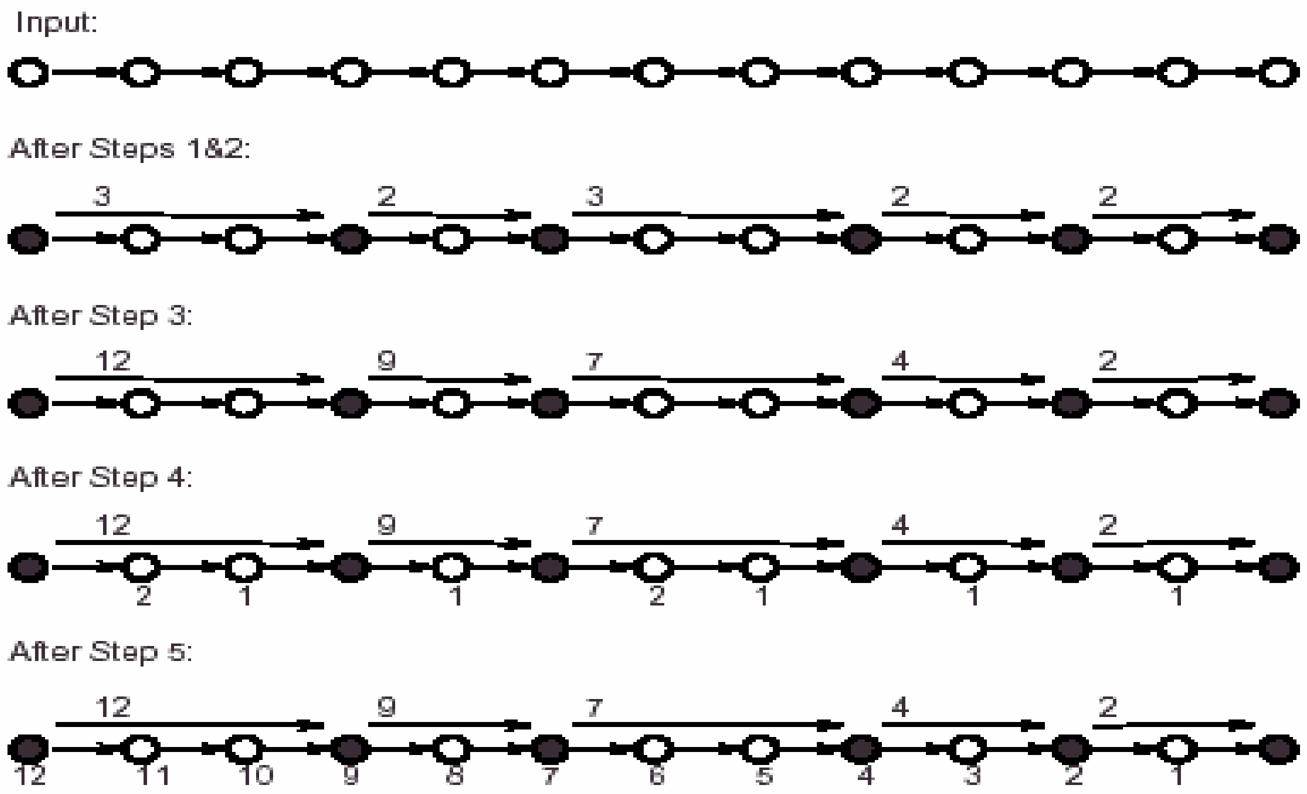


Fig. 3. Illustration of Algorithm 1.

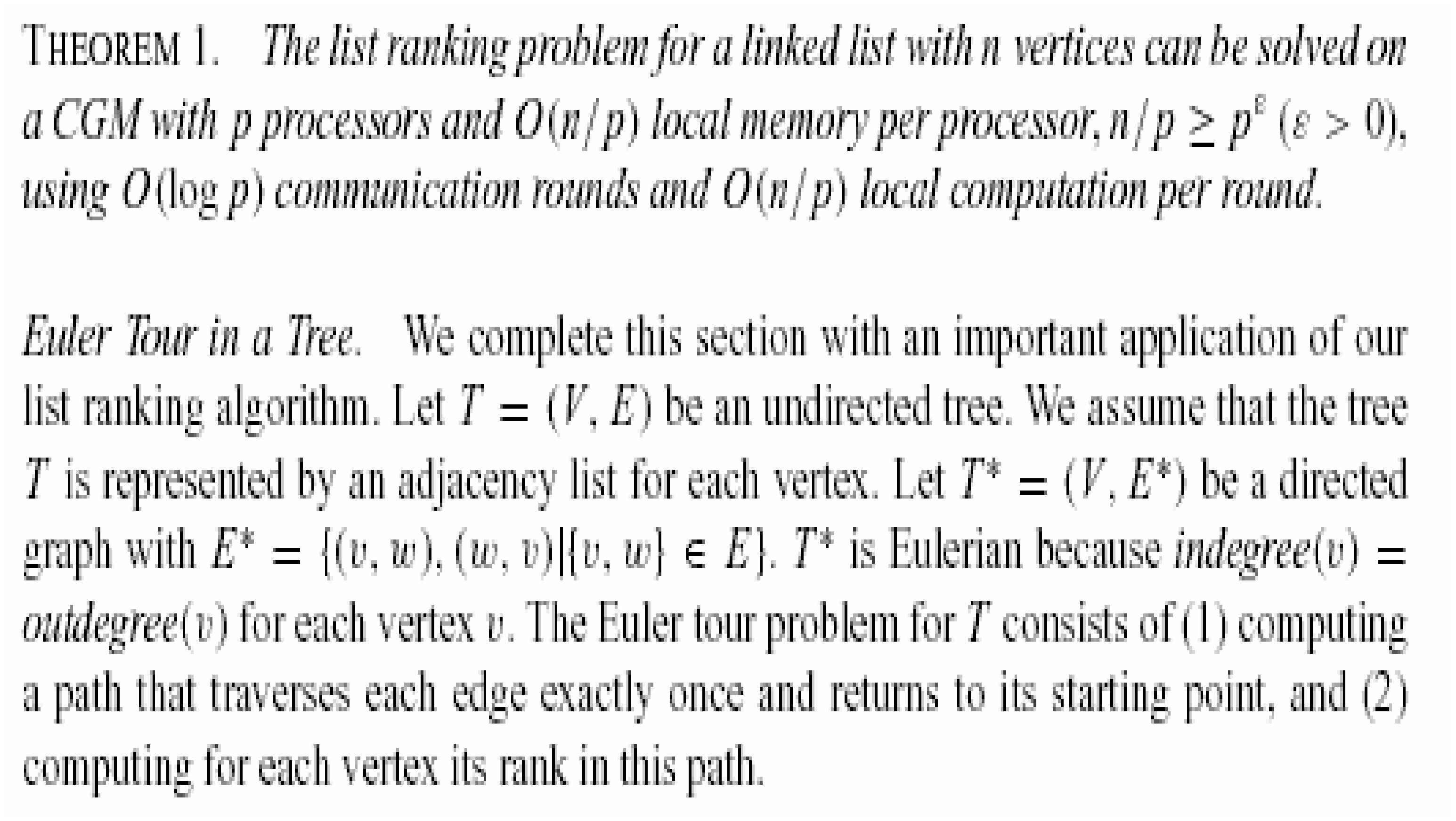
CGM List Ranking

- Steps 2-5 can be easily implemented on a CGM in O(logp) communication rounds with O(n/p) local computation per round.
- In Step 4 we are simulating PRAM pointer jumping steps on the CGM. Each such step can be implemented in O(1) rounds by applying Goodrich’s sorting algorithm (1996).

CGM List Ranking

- In order to compute an O(p^2)-ruling set in O(logp) communication rounds, a technique, called deterministic list compression is used.
- The basic idea behind *deterministic list compression* is to have an alternating sequenceof *compress* and *concatenate* phases. In a compress phase we *select* a subset of list elements, and in a *concatenate* phase we use pointer jumping to work our way towards building a linked list of selected elements.

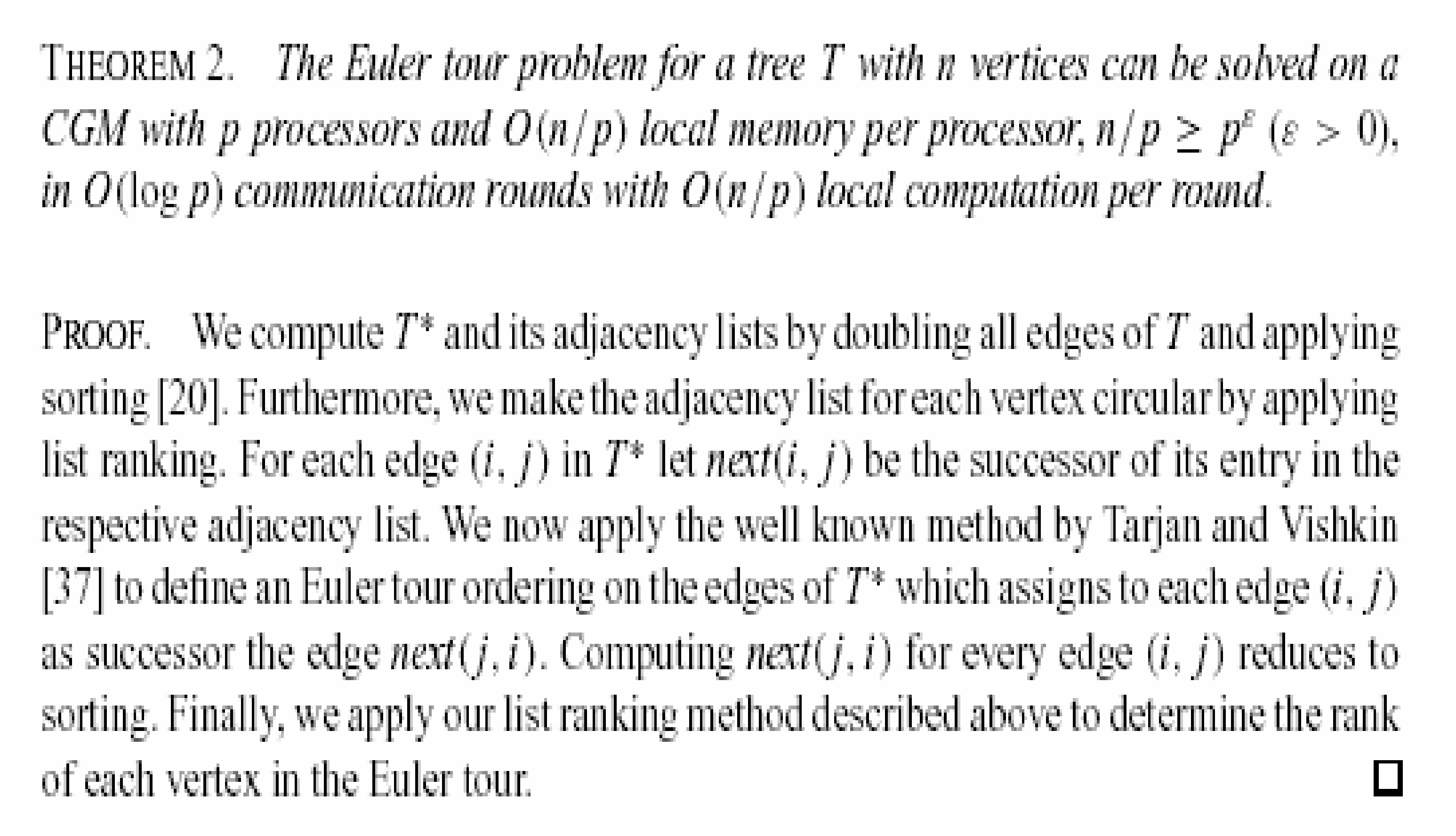
CGM List Ranking



THEOREM 1. *The list ranking problem for a linked list with n vertices can be solved on a CGM with p processors and $O(n/p)$ local memory per processor, $n/p \geq p^{\varepsilon}$ $(\varepsilon > 0)$, using $O(\log p)$ communication rounds and $O(n/p)$ local computation per round.*

Euler Tour in a Tree. We complete this section with an important application of our list ranking algorithm. Let $T = (V, E)$ be an undirected tree. We assume that the tree T is represented by an adjacency list for each vertex. Let $T^* = (V, E^*)$ be a directed graph with $E^* = \{(v, w), (w, v) | \{v, w\} \in E\}$. T^* is Eulerian because *indegree*(v) = *outdegree*(v) for each vertex v. The Euler tour problem for T consists of (1) computing a path that traverses each edge exactly once and returns to its starting point, and (2) computing for each vertex its rank in this path.

CGM List Ranking



THEOREM 2. *The Euler tour problem for a tree T with n vertices can be solved on a CGM with p processors and $O(n/p)$ local memory per processor, $n/p \geq p^{\varepsilon}$ $(\varepsilon > 0)$, in $O(\log p)$ communication rounds with $O(n/p)$ local computation per round.*

PROOF. We compute T^* and its adjacency lists by doubling all edges of T and applying sorting [20]. Furthermore, we make the adjacency list for each vertex circular by applying list ranking. For each edge (i, j) in T^* let *next*(i, j) be the successor of its entry in the respective adjacency list. We now apply the well known method by Tarjan and Vishkin [37] to define an Euler tour ordering on the edges of T^* which assigns to each edge (i, j) as successor the edge *next*(j, i). Computing *next*(j, i) for every edge (i, j) reduces to sorting. Finally, we apply our list ranking method described above to determine the rank of each vertex in the Euler tour. □