COMPUTING MINIMUM SPANNING FORESTS ON 1- AND 2-DIMENSIONAL PROCESSOR ARRAYS

(Extended Abstract)

Friedhelm Meyer auf der Heide *
Informatik II, Universität Dortmund
D - 4600 Dortmund 50, Fed. Rep. of Germany

Abstract: Minimum spanning forests (MSFs) can be computed in time $O(n^2/l)$ on a 1-dimensional processor array of length $l \leq n$. For this result we apply a new algorithmic approach different from e.g. Sollin's. It holds for arbitrary input conventions if we only count communication rounds. If we also take internal computation into account it still holds for a wide class of input conventions, generalizing a result by Doshi and Varman. For $l \times l$-meshes, $\sqrt{n} \leq l \leq n$, we present two input conventions for which computing MSFs needs different numbers of communication rounds. For one of them we prove the interesting phenomenon that the complexity is not monotone in l: It is $\approx n$ for $l = \sqrt{n}$ and $l = n$, but takes its minimum, $\approx n^{3/4}$, for $l = n^{3/4}$.

1. Introduction

In this paper we present upper and lower bounds for computing minimum spanning forests (MSFs) of weighted graphs on simple parallel computation models, namely the 1-dimensional processor-array with l processors (l-array), and the 2-dimensional processor-array with l^2 processors ($l \times l$-mesh).

There is a variety of approaches to compute MSFs sequentially due to Sollin, Prim-Dijkstra, and Kruskal (compare e.g. [M]). Most known parallel algorithms parallelize Sollin's approach (for an overview see [QD]).

Therefore we give here a recursive, high-level description of this algorithm. (For simplicity we assume all edge weights to be different.)

ALGO 0 (Input: Weighted graph G with n nodes)
Step 1: Compute the min-subforest F of G, i.e. the subforest of G consisting of the cheapest edges incident to each node. (F is part of an MSF of G.)
Step 2: Compute the connected components $A = (A_1, \ldots, A_s)$ of F ($s \leq n/2$).
Step 3: Compute the A-contraction G' of G. I.e.: Contract the nodes from each A_p to one node p, join p and q with the cheapest edge in G between A_p and A_q.
Step 4: Compute recursively an MSF F' of G'.
Step 5: Compute the subforest $\tilde{F}$ of G consisting of those edges the edges in F' descend from ($F \cup \tilde{F}$ is an MSF of G.)

The algorithm has recursion depth $\log(n)$.

* Supported in part by DFG-Grants ME-872/1-2 and WE 1066/2-1

There are many papers presenting implementations of this algorithm on several parallel computation models. For shared memory machines (PRAMs) the optimal time-processor product $\Theta(n^2)$ (which is the sequential time) can be achieved for up to $n^2/\log(n)^2$ processors, as shown by Chin,Lam,Chen in [CLC]. Clearly, on an l-array, a lower bound $\Omega(l)$ holds, if the input is distributed evenly among the processors. Thus an optimal time-processor product is impossible for $l = \omega(n)$.

Doshi and Varman have shown in [DV] that the optimal time bound can indeed be achieved for $l \leq n$. They designed an algorithm on l-arrays, $l \leq n$, that computes MSFs in time $O(n^2/l)$. They implement Sollin's algorithm. The disadvantage of their solution is that it is heavily based on their choice of the input convention. They assume that each processor gets an $(l/\sqrt{n}) \times (l/\sqrt{n})$-submatrix of the adjacency matrix, and that these submatrices are distributed over the processors in "shuffled row and column" order. This makes it possible that the critical step 3 of Sollin's algorithm becomes cheaper on lower recursion levels, because communication only has to take place over short distances. This is no longer true if we change the input convention. Then, e.g. for $l = n$, each recursion level needs $\Theta(n)$ steps (the diameter of the n-array). Thus the runtime becomes $\Theta(n \log(n))$.

A different, simpler approach due to Preilowski [P] computes (not necessarily minimum) spanning forests in $O(n)$ time for a large variety of input conventions.

In this paper we present a new algorithmic approach which finds a way around the high amount of communication for step 3. We allow that a processor does not necessarily know the correct weight of the cheapest edge between nodes in order to start the next recursion level, i.e. we allow wrong edge-weights. Our approach leads to the following result for l-arrays, $1 \leq l \leq n$:

- *The number of communication rounds to compute an MSF on an l-array is $\Theta(n^2/l)$, independent of the input convention.*

Here we assume that internal computation is free, and that, in order to be fair, only names of nodes or names and weights of edges can be communicated to neighbouring processors in one round. If we take internal computation into account, we have to put some restrictions on the input conventions in order to achieve $O(n^2/l)$ runtime. These restrictions still allow a variety of "reasonable" input conventions including that from Doshi and Varman as well as the "most natural" convention where processors get adjacency lists of nodes.

Furthermore we consider $l \times l$-meshes, $\sqrt{n} \leq l \leq n$. Here we only count the number of communication rounds, because this seems to mirror the inherent parallelism of computing MSFs. It turns out that the time-processor product can become as small as $n^{3/2}$ instead of n^2, if we measure time by number of communication rounds.

Atallah and Kosaraju [AK] have designed an algorithm that takes $O(n)$ steps on an $n \times n$-mesh. Thus, for a suitable input convention, we can emulate the $n \times n$-mesh on an $l \times l$-mesh with optimal timeloss achieving $O(n^3/l^2)$ runtime. This is also the best bound for the number of communication rounds known in the literature.

We shall see in this paper that, in contrast to l-arrays, the number of communication rounds in $l \times l$-meshes depends on the choice of the input convention, even if we demand that each processor gets the same number of edges.

To illustrate this we consider two input conventions.

Convention A: The $l \times l$-mesh is partitioned into n $l' \times l'$-meshes, $l' = l/\sqrt{n}$, each of which stores an adjacency list of a node.

Convention B: The adjacency matrix is partitioned into l^2 submatrices of size $n/l \times n/l$. Each processor gets the respective submatrix.

We show that convention B allows substantially faster algorithms than convention A.

For the lower bounds, we assume that each processor can only communicate one bit per round, and that at the end of the computation each processor knows those edges from its input that belong to the MSF. On the other hand we prove the lower bounds even for the case that only weights 1 and 2 are allowed. Thus, messages allowed for the upper bounds have length $O(\log(n))$, and our lower bounds need only be divided by $\log(n)$ in order to be directly comparable to the upper bounds.

We achieve the following upper and lower bounds for our two input conventions.

- *The number of rounds needed by an $l \times l$-mesh ($\sqrt{n} \leq l \leq n$) to compute an MSF of an n-node graph given by convention B is $\Omega(l)$. This is tight within a logarithmic factor.*

- *For convention A, the number of rounds is $\Omega(n^{3/2}/l + l)$. Also this is tight within a logarithmic factor.*

The result for convention A proves an interesting phenomenon: The number of rounds is not a monotone function in l. Instead, within logarithmic factors, it is $\Theta(n)$ for $l = \sqrt{n}$ and $l = n$ and takes its minimum, $\Theta(n^{3/4})$, for $l = n^{3/4}$.

The paper is organized as follows. In Chapter 2 we introduce graph-theoretical notions and properties of MSFs needed for the algorithms. In Chapter 3 we define our computation models and state the results. Chapter 4 presents the algorithms for l-arrays, Chapter 5 the lower bounds and Chapter 6 the upper bounds for $l \times l$-meshes.

2. Properties of MSTs

In this paper, G always denotes a weighted graph $G = (V, E, w)$ with vertex set $V = \{1, \ldots, n\}$, edge set $E \subseteq \binom{V}{2}$ and weight function $w : E \to I\!N$. The *min-subforest of G* is the forest on vertex set V that contains, for each non-isolated node $i \in V$, an edge $\{i, j\} \in E$ with smallest weight. Ties are broken in favour of the smallest j. (It is easily checked that these edges form a forest.)

Let $A = \{A_1, \ldots, A_s\}$ be a disjoint partition of V. The *A-contraction of G*, $G/_A$, is the graph with vertex set $V' = \{1, \ldots, s\}$, edge set E' with $\{\alpha, \beta\} \in E'$ iff there are

$i \in A_\alpha$, $j \in A_\beta$ with $\{i,j\} \in E$, and weight function $w' : E' \to \mathbb{N}$ with $w'(\alpha,\beta) := \min\{w(i,j), i \in A_\alpha,\ j \in A_\beta, \{i,j\} \in E\}$. If $w(i,j)$ is this minimum, then we say that $\{\alpha,\beta\}$ decends from $\{i,j\}$.

The following lemma collects properties of MSFs which are easy implications of the "matroid property" of spanning forests.

Lemma (The MSF-lemma):

a) There is an MSF of G which contains the min-subforest of G.

b) Let $A = \{A_1, \ldots, A_s\}$ be the partition of V given by the connected components of its min-subforest. Let F be an MSF of $G/_A$, and $\tilde{E}$ be the subset of E containing exactly those edges, the edges of F descend from. Then $\tilde{E}$ together with the min-subforest of G forms an MSF of G.

c) Let $G = (V,E,w)$, $G_1 = (V,E_1,w_1)$, and $G_2 = (V,E_2,w_2)$ be weighted graphs, $E = E_1 \cup E_2$, $w(i,j) = min\{w_1(i,j), w_2(i,j)\}$. Let F_1, F_2 be MSFs of G_1, G_2. Then $F_1 \cup F_2$ contains an MSF of G.

d) Let $G' = (V,E',w)$ be a subgraph of G, $F' = (V,E'',w)$ an MSF of G', $G^* = (V,(E \setminus E') \cup E'',w)$. Then each MSF of G^* is an MSF of G.

a) and b) form a correctness proof of Sollin's algorithm mentioned in the introduction. c) is the heart of our new linear time algorithm on n-arrays described in chapter 4.

3. Definitions and Results

In this paper we deal with two well-known parallel computation models, the l-array and the $l \times l$-mesh.

An *l-array* consists of l processors $P_1, \ldots, P_l$. Communication links only exist between P_k and P_{k+1}, $1 \le k \le l-1$.

An $l \times l$-mesh consists of l^2 processors $P_{h,k}, 1 \le h,k \le l$. Communication links from $P_{h,k}$ go to $P_{h\pm1,k}$ and $P_{h,k\pm1}$ (if these processors exist). Communication links can be used in both directions.

In one communication round, each processor can execute an arbitrary amount of internal computation and can send one message to one of its neighbours. In order to be fair we do not allow that a processor encodes much information in one message, because this would make it capable of communicating e.g. its whole configuration in one step. Instead, in case we want to compute an MSF of G, messages may only consist of a weight of some input edge, or the name of a node.

We consider two complexity measures:

1) number of communication rounds
2) runtime (Here we also take internal computation into account, assuming the usual uniform cost measure.)

As we shall prove later, it is crucial for the efficiency of an algorithm on l-arrays or $l \times l$ meshes how the input is represented, i.e. which processor initially knows which input variables. Thus we have to consider input-conventions.

An *input convention* on l-arrays (for graphs G) is a disjoint partition $(E_1, \ldots, E_l)$ of $\binom{V}{2}$. The interpretation is that P_k knows the weighted edges of G that are in $E_k \cap E$. Input conventions $(E_{h,k}, 1 \leq h, k \leq l)$ on $l \times l$-meshes are defined analogously.

Now we are ready to state our results.

Theorem 1: *For each input convention on l-arrays, $1 \leq l \leq n$, there is an algorithm on l-arrays that computes an MSF of a graph G in $O(n^2/l)$ communication rounds.*

We shall see later that this algorithm is conceptually different from all known algorithms, e.g. from Sollin's algorithm.

In order to achieve linear *runtime*, we have to put some restrictions on the allowed input conventions.

An input-convention $(E_1, \ldots, E_n)$ on n-arrays is *nice*, if the following holds for each input $G = (V, E, w)$:

a) For each $1 \leq k \leq n$, an MSF T_k of $G_k = (V, E_k, w)$ can sequentially be computed in time $O(n)$,
b) For each $1 \leq k \leq n$, and for each disjoint partition $A = (A_1, \ldots, A_s)$ of V, an MSF of $T_k/_A$ can sequentially be computed in time $O(s + n/\log(n))$.

$(E_1, \ldots, E_l)$ is *nice on l-arrays*, $l \leq n$, if there is $(E'_1, \ldots, E'_n)$ nice on n-arrays such that $E_i = E'_a \cup \ldots \cup E'_b$, $a = (i-1)n/l + 1$, $b = i \cdot n/l$, for $1 \leq i \leq l$.

Examples for nice input conventions on n-arrays:

(i) $E_k = \{\{k, j\}, j \neq k\}$. (This is the appearently "most natural" input convention : Each processor gets a weighted adjacency list of one node.)
(ii) The weighted adjacency matrix of G is partitioned into n $\sqrt{n} \times \sqrt{n}$-submatrices $B_{a,b}$, $1 \leq a, b \leq \sqrt{n}$. Each processor gets one of them. (This is — from the algorithmic point of view — the simplest input convention. For one special distribution of the $B_{a,b}$'s over the processors according to the "shuffled row and column assignment scheme", Doshi and Varman have presented an algorithm with linear runtime in [DV]. Their algorithm only works for this input convention.)
(iii) E_k is a subset of $\binom{V}{2}$, $\#E_k \leq n$, which only touches at most $n/\log(n)$ nodes.
(iv) If $(E_1, \ldots, E_n)$ is nice, then also $(E_{\pi(1)}, \ldots, E_{\pi(n)})$ for every permutation π on $\{1, \ldots, n\}$.

(i) and (iv) are clear, (ii) and (iii) are easily checked using well known sequential algorithms for MSFs (see [M]).

Theorem 2: *For each nice input convention on l-arrays, $l \leq n$, there is an algorithm on l-arrays that computes an MSF of a graph G in $O(n^2/l)$ runtime.*

This result generalizes the result from [DV] mentioned in (ii) above. As n^2, the number of input variables, is clearly a sequential lower bound, the algorithms from the above two theorems are asymptotically optimal.

Remark: *It suffices to prove the upper bounds of Theorems 1 and 2 for n-arrays.*

The generalization to l-arrays, $l \leq n$, is simply done by the following simulation: Each processor of the l-array executes the work of n/l consecutive processors of the n-array. This slows down the runtime and the number of communication rounds only by a factor $O(n/l)$, which yields the theorems. (Note that the definition of nice input conventions on l-arrays is consistent with this simulation.)

We now turn to $l \times l$-meshes, $\sqrt{n} \leq l \leq n$. Here we only consider communication rounds. It turns out that, in contrast to l-arrays, the input convention is crucial for the efficiency of the algorithms. To illustrate this we consider two input conventions:

Convention A: The $l \times l$-mesh is partitioned into n $l' \times l'$-submeshes, $l' = l/\sqrt{n}$, each of which knows the adjacency list of one node of G, distributed evenly among the processors of the submesh.
Convention B: The adjacency matrix of G is partitioned into l^2 $n' \times n'$-submatrices, $n' = n/l$. The h-th submatrix in the k-th row of submatrices is known to $P_{h,k}$.

These conventions correspond to the conventions (i) and (ii), resp., for l-arrays described earlier.

Theorem 3: *Let* $\sqrt{n} \leq l \leq n$.
a) Each $l \times l$*-mesh that computes an MSF of a graph* G *given by convention A needs* $\Omega(l + n^{3/2}/l)$ *communication rounds.*
b) The above lower bound is tight within a $\log(n)$ *factor.*

This result shows that, for convention A, the best choice of l is $n^{3/4}$. In this case the number of communication rounds is $\Theta(n^{3/4})$ (up to a $\log(n)$ factor), whereas both for $l = \sqrt{n}$ and $l = n$ we get a linear number of rounds.

For convention B, we obtain different bounds.

Theorem 4: *Let* $\sqrt{n} \leq l \leq n$.
a) Each $l \times l$*-mesh that computes an MSF of a graph* G *given by convention B needs* $\Omega(l)$ *communication rounds.*
b) The above lower bound is tight within a $\log(n)$ *factor.*

In this case it turns out that the (trivial) lower bound l ($\approx$ diameter of the $l \times l$-mesh) can, up to a $\log(n)$ factor, be reached in the whole range of l between $\sqrt{n}$ and n, not only between $n^{3/4}$ and n as for convention B.

4. Upper bounds on n-arrays

In this chapter we prove Theorems 1 and 2. By the remark from Chapter 3 we only have to consider n-arrays. We start with an algorithm that only takes communication rounds into account.

In order to make our recursive approach work, we have to define an even more general type of input convention, where we allow that processors know wrong weights of edges.

A representation of $G = (V, E, w)$ is a tuple $((E_1, w_1), \ldots, (E_n, w_n))$, where $E_k \subseteq E$, and $w_k : E_k \to I\!N$ is a weight function on E_k, $1 \leq k \leq n$, with the properties:
(i) $E = \bigcup_{k=1}^{n} E_k$ (The E_k's are not necessarily disjoint)
(ii) For all $\{i,j\} \in E$, $w(i,j) = min\{w_k(i,j), \{i,j\} \in E_k\}$.

ALGO 1 (Input: a representation of a graph G)
Step 1: Compute the min-subforest F of G.
Remark: Using pipelining, this can be done as follows: P_1 sends, one after another, packets $(i, v(i), j), 1 \leq i \leq n$, along the array to P_n. Initially, $j = i$, $v(i) = \infty$. If $(i, v(i), j)$ arrives at P_k, P_k checks whether it knows an edge $\{i, j'\}$ incident to i with $w_k(i,j') < v(i)$ or $w_k(i,j') = v(i)$ and $j' < j$. If yes, it chooses such an edge (i,j') with smallest weight, or with smallest j', resp., and replaces $(i, v(i), j)$ by $(i, w_k(i,j'), j')$. Thus, after $O(n)$ rounds, P_n knows the min-subforest F of G.

Step 2: P_n computes the connected components $A = (A_1, \ldots, A_s)$ of F and broadcasts them to $P_1, \ldots, P_n$.
Remark: This takes $O(n)$ rounds. (We do not count for the internal computation of P_n.) Furthermore, $s \leq n/2$.

Step 3: For each $1 \leq k \leq n$, P_k computes the A-contraction $G'_k = (V', E'_k, w'_k)$ of $G_k = (V, E_k, w_k)$ (with $V' = \{1, \ldots, s\}$).
Remark: This step needs no communication. As a result we get the following: Let $G_1 = (V', E^1, w^1)$ and $G_2 = (V', E^2, w^2)$ be the graphs represented by $((E'_1, w'_1), \ldots, (E'_{n/2}, w'_{n/2}))$ and $((E'_{n/2+1}, w'_{n/2+1}), \ldots, (E'_n, w'_n))$, resp. Let $G^* = (V', E^*, w^*)$ be the A-contraction of G. Then $E^* = E^1 \cup E^2$ and for each $\{i,j\} \in E^*$, $w^*(i,j) = min\{w^1(i,j), w^2(i,j)\}$. Thus the MSF-lemma suggests the following two steps to complete the algorithm.

Step 4: In parallel, recursively, $P_1, \ldots, P_{n/2}$ and $P_{n/2+1}, \ldots P_n$ compute MSFs F_1 and F_2 of G_1 and G_2, resp. (Note that G_1 and G_2 have $s \leq n/2$ vertices.)

Step 5: Send F_1 and F_2 to P_1. P_1 computes an MSF F^* of $F_1 \cup F_2$ and, from that, the subset $\tilde{F} \subseteq E$ the edges in F^* descend from.
Remark: This step needs $O(n)$ communication rounds, because $F_1 \cup F_2$ only has $O(n)$ edges. It follows directly from the MSF-lemma that $F \cup F^*$ is an MSF of G. (F denotes the min-subforest of G.)

By the remarks, the algorithm is shown to be correct. Let $T(n)$ denote its worst case number of communication rounds for graphs with n nodes on an n-array. As steps 1,2,3,5 need $O(n)$ rounds together, and Step 4 needs at most $T(n/2)$ rounds, we obtain $T(1) = O(1)$, and for $n > 1$, $T(n) = T(n/2) + O(n)$. Thus $T(n) = O(n)$, which proves Theorem 1.

In order to prove Theorem 2 we have to define *nice representations* of graphs $G = (V, E, w)$. A representation $((E_1, w_1), \ldots, (E_n, w_n))$ of G is nice if the following holds:

a) For each $1 \leq k \leq n$, an MSF T_k of $G_k = (V, E_k, w_k)$ can sequentially be computed in time $O(n)$,

b) For each $1 \leq k \leq n$, and for each disjoint partition $A = (A_1, \ldots, A_s)$ of V, an MSF of $T_k/_A$ can sequentially be computed in time $O(s + n/\log(n))$.

We always assume that the graph G_k stored in P_k is represented as adjacency lists, where each list is ordered according to the weights, and ties are broken in favour of the smaller node.

ALGO 2 (Input: a nice representation of a graph G)
Step 1: Compute the min-subforest F of G.
Remark: This is done as described in Step 1 in Algo 1. Each processor only needs constant time to find the edge $\{i, j'\}$, because of the internal representation of G_k in P_k mentioned above. Thus Step 1 needs $O(n)$ steps.

Step 2: P_n computes the connected components $A = (A_1, \ldots, A_s)$ of F and broadcasts them to $P_1, \ldots, P_n$.
Remark: This can be done in $O(n)$ steps sequentially using e.g. depth first search. The broadcast clearly needs also $O(n)$ steps.

Step 3a: For each $1 \leq k \leq n$, P_k computes an MSF F_k of $G_k = (V, E_k, w_k)$.
Remark: This can be done in time $O(s+n/\log(n))$ by definition of nice representations. This step is necessary as we cannot compute $G_k/_A$ in time $O(n)$, because G_k can have $\omega(n)$ edges, whereas F_k has at most $n-1$ edges.

Step 3b: For each $1 \leq k \leq n$, P_k computes the A-contraction $G'_k = (V', E'_k, w'_k)$ of F_k.
Remark: This can be done in time $O(n)$, because F_k has at most $(n-1)$ edges. The MSF-lemma d) guarantees that we obtain a correct algorithm if we now go on as described in Algo 1 following the remark for step 3 (with our new interpretation of G'_k). We now use the notations from Algo 1.

Step 4: In parallel, recursively, $P_1, \ldots, P_{n/2}$ and $P_{n/2+1}, \ldots, P_n$ compute an MSF F_1 and F_2 of G_1 and G_2, resp.

Step 5: Compute an MSF F^* of $F_1 \cup F_2$ and the set $\tilde{F} \subseteq E$ of edges from which the edges of F^* descend.
Remark: We cannot, as in Algo 1, compute F^* sequentially, because this would take $\omega(n)$ steps. There are many ways of computing F^* in $O(n)$ time. One way is as follows: Broadcast $F_1 \cup F_2$ to all processors. This takes $O(n)$ time, because $F_1 \cup F_2$ has $O(n)$ edges. As a result, we are ready to apply any $O(n)$ algorithms for an arbitrary input convention. Thus we may use the algorithm by Doshi and Varman from [DV].

The correctness of this algorithm follows directly from the remarks in Algo 1 and Algo 2. In order to compute the runtime, we have to be careful because of Step 3a. First let us assume that Step 3a takes $O(s)$ steps for graphs with s nodes on s-arrays.

Then we would get the same recursion as in Algo 1, i.e. Algo 2 would have runtime $O(n)$. In order to also take the additional $O(n/\log(n))$ steps in each of the d iterations of step 3a into account, we have to add $O(d \cdot n/\log(n))$ steps. But as $d = \log(n)$ we obtain $O(n)$ runtime.

5. Lower bounds on l×l-meshes

In this chapter we prove the lower bounds from Theorems 3 and 4. The $\Omega(l)$ lower bound in both theorems is clear because this is the time needed to send a message through an $l \times l$-mesh.

Thus it remains to prove the $\Omega(n^{3/2}/l)$ lower bound from Theorem 3 under convention A.

Let M be the upper left $l' \times l'$-submesh of the $l \times l$-mesh, $l' = l/\sqrt{n}$. By the definition of convention A, it contains all the weighted edges incident to one node, e.g. to node 1.

The outline of the proof is as follows: We show that (almost) each subset of the edges $\Gamma(1) = \{\{1,2\},\ldots,\{1,n\}\}$, known in M can belong to the MSF. Thus M needs to get $\approx n$ bits of information in order to distinguish beween these $\approx 2^n$ situations. In one round, M can only get $2l/\sqrt{n}$ bits of information (this is the number of links leaving M in the $l \times l$-mesh). Thus, at least $\approx n/(2l/\sqrt{n}) = \frac{1}{2}n^{3/2}/l$ rounds are necessary. In order to make this precise, we first show:

Claim: *For each $D \subseteq \Gamma(1) \setminus \{\{1,n\}\}$, there is a graph $G = (V,E,w)$ with $\Gamma(1) \subseteq E$, $w(\Gamma(1) \setminus \{\{1,n\}\}) = \{2\}$, $w(E) = \{1,2\}$ such that for each MSF F of G, $F \cap \Gamma(1) = D \cup \{\{1,n\}\}$ holds.*

Proof: Let $D \subseteq \Gamma(1) \setminus \{\{1,n\}\}$ be fixed, $V' = \{j, \{1,j\} \in D\}$, $V'' = V \setminus (V' \cup \{1\})$. Clearly, the following graph G fulfils the claim.

- V'' is connected in G,
- V' is independent in G,
- the edges in $\Gamma(1) \setminus \{\{1,n\}\}$ have weight 2, all others weight 1.

Now consider as inputs all graphs with edge weights 1 or 2, which contain all edges from $\Gamma(1)$, weighted as in the claim. Restricted to these graphs, the part of the input known to M, the upper left $l' \times l'$-submesh, is constant. Therefore, in order to distinguish between the possible outputs $D \subseteq \Gamma(1)$ for M, M needs $n-2$ bits of information from outside M. This is true because, by the claim, 2^{n-2} choices of D are possible. Thus, the argument before the claim proves the lower bound.

6. Upper Bounds on l×l-meshes

In this chapter we present the upper bounds from Theorems 3 and 4.

First we consider Theorem 4, where the input is given by convention B. We again have to generalize this convention in order to make our recursive approach work.

A *B-representation* of a graph $G = (V, E, w)$ is a tuple $((E_{h,k}, w_{h,k}),\ 1 \leq h, k \leq l)$, where $E_{h,k} \subseteq E$, $w_{h,k} : E_{h,k} \to I\!N$ with the properties:
(i) $\bigcup_{1 \leq h,k \leq l} E_{h,k} = E$
(ii) For each $e \in E$, $w(e) = \min\{w_{h,k}(e),\ e \in E_{h,k}\}$
(iii) For each $1 \leq h, k \leq l$, there are sets $I_h, J_k \subseteq V$, $\#I_h, \#J_k = O(n/l)$, such that all nodes incident to edges from $E_{h,k}$ belong to $I_h \cup J_k$.

Clearly, convention B is a B-representation.

ALGO 3 (Input: a B-representation of a graph G)
Step 1: Compute the min-subforest F of G as follows:
a) For each $1 \leq h \leq l$, compute, for each $i \in I_h$, the edge $\{i, j_{i,h}\}$ incident to i with smallest weight $v_h(i)$ among the weights known for edges incident to i by processors in column h. Ties are broken in favor of smaller nodes.
Remark : As each column can be looked upon as an l-array this can be done in $O(n/l + l) = O(l)$ rounds (note that $\sqrt{n} \leq l \leq n$), similar to Step 1 of Algo 1. As a result, we have computed $\#I_h = O(n/l)$ many triples $(i, v_h(i), j_{i,h})$ in each column, thus $O(n)$ triples altogether.
b) Do the same as in a) for each row k, for each $i \in J_k$.
Remark: As a result, analogously to a), we need $O(l)$ rounds and get $O(n)$ triples $(i, v'_k(i), j'_{i,k})$. Thus we have $O(n)$ triples $(i, v_h(i), j_{i,h})$ and $(i, v'_k(i), j'_{i,k})$. The edge incident to i that belongs to the min-subforest is given by the third component of the triple with first component i and lexicographically minimum pair (second component, third component).
c) Sort the $O(n)$ triples lexicographically (first priority i, second priority $v_h(i)$ (resp. $v'_k(i)$) third priority $j_{i,h}$ (resp. $j'_{i,k}$)
Remark: This can be done in $O(l)$ steps, because the number of elements to be sorted is linear in the number of processors (Compare [SS]). As a result, the first triple in the ordered sequence with first component i represents the edge incident to i in F. Thus we have computed F in time $O(l)$.

Step 2: Compute the connected components $A = (A_1, \ldots, A_s)$ of F as follows:
a) Expand each node with degree d in F to d nodes of degree 3.
Remark: This can easily be done in time $O(l)$. The resulting forest F' still has $O(n)$ vertices, because F only has $O(n)$ edges.
b) Distribute the $O(n)$ edges of F' evenly among the processors of the upper left $\sqrt{n} \times \sqrt{n}$-mesh.
c) Compute the connected components $A'_1, \ldots A'_s$ of F' in this $\sqrt{n} \times \sqrt{n}$-mesh.
Remark: This can be done with an algorithm described by Nassimi and Sahni in [NS] in time $O(\sqrt{n \log(n)})$.
d) Compute $A_1, \ldots, A_s$ from $A'_1, \ldots, A'_s$.
Remark: This is done in $O(l)$ rounds by reversing the expansion from a). Thus Step 2 takes $O(l + \sqrt{n \log(n)})$ rounds.

Step 3: For each $p \in \{1,\ldots,s\}$, $j \in A_p$, send (p,j) (= "node j of G belongs to A_p") to each processor $P_{h,k}$ with $j \in I_h \cup J_k$ as follows:

a) For each $1 \leq h \leq l$, send each (p,j), $j \in I_h$, to column h.
Remark: Initially, each processor in the upper left $\sqrt{n} \times \sqrt{n}$-mesh knows $O(1)$ many pairs. These pairs are pipelined within their rows over the whole mesh. This takes time $O(l+\sqrt{n}) = O(l)$.

b) For each $1 \leq h \leq l$, broadcast each pair in column h to all processors in column h.
Remark: As there are only $O(n/l)$ pairs in column h, this takes $O(n/l+l) = O(l)$ rounds.

c) Do the analogue to a),b) for each row.
Remark: After this, each processor $P_{h,k}$ knows, for each node i from $I_h \cup J_k$, the component A_p to which i belongs. Thus $P_{h,k}$ can internally compute the A-contraction $G'_{h,k} = (V', E'_{h,k}, w'_{h,k})$ of the graph $G_{h,k} = (V, E_{h,k}, w_{h,k})$ ($V' = \{1,\ldots,s\}$). Let $G' = (V', E', w')$ be the graph with B-representation $((E'_{h,k}, w'_{h,k})$, $1 \leq h,k \leq l)$. G' is the A-contraction of G. Thus the MSF-lemma suggests the following steps to complete the algorithm.

Step 4: Recursively compute an MSF F^* of G'. (The recursion terminates if G' has one vertex, i.e. after $\log(n)$ iterations.)

Step 5: Compute $\tilde{F}$, the set of edges in E from which the edges in F^* descend.
Remark: This step needs no communication. The MSF-lemma shows that $F \cup \tilde{F}$ is an MSF of G. (F is the min-subforest, compare step 1.)

By the remarks, the algorithm is shown to be correct. The number of rounds needed for each of the $\log(n)$ recursive calls is $O(l + \sqrt{n\log(n)})$. (Note that deeper recursive calls do not become cheaper, because they still work on the whole $l \times l$-mesh.) Thus the algorithm needs $O(l\log(n) + \sqrt{n}\log(n)^{3/2})$ rounds.

Also for the algorithm for Theorem 3 we need a generalization of the input convention. For $1 \leq r,q \leq \sqrt{n}$ let $B_{r,q}$ denote the $l' \times l'$-mesh, $l' = l/\sqrt{n}$, consisting of the processors $P_{h,k}$ with $(r-1)l'+1 \leq h \leq rl'$, $(q-1)l'+1 \leq h \leq ql'$. An *A-representation* of a graph $G = (V,E,w)$ is defined like a B-representation, except for (iii), which has to be replaced by

(iii)': $\#E_{h,k} = O((n/l)^2)$ for $1 \leq h,k \leq l$, and each $B_{r,q}$ contains a subset of $\Gamma(i) \cap E$ for some node $i \in V$.

ALGO 4: (Input: an A-representation of a graph G)
Step 1 is easier than in Algo 3: First each $B_{r,q}$ computes its edge with smallest weight in $O(l/\sqrt{n})$ rounds. Then, for all $1 \leq i \leq n$, among the edges chosen in those $B_{r,q}$'s that know weights of edges in $\Gamma(i) \cap E$, the smallest can be computed in $O(l)$ rounds, which finishes the computation of F.

Step 2 can be done as in Algo 3.

Step 3 takes longer than in Algo 3. It takes at least $n/(4 \cdot l/\sqrt{n}) = \frac{1}{4}n^{3/2}/l$ rounds, because each $B_{r,q}$ has to obtain up to n pairs (p,j), but can only obtain 4 $l/\sqrt{n}$ such pairs per round. (Compare the lower bound argument from Chapter 5.) We shall show how this time bound can be achieved. (Note that, after Step 2, each of the first $\sqrt{n}$ rows know $O(\sqrt{n})$ pairs, $O(1)$ pair per processor.

a) For $1 \leq h \leq \sqrt{n}$, broadcast the pairs known in row h to all processors in row h.
Remark: This takes $O(\sqrt{n}+l) = O(l)$ rounds. Now in each column all pairs are known, $O(\sqrt{n})$ per processor. Assume that we have partitioned the set of pairs in l' sets $L_1, \ldots, L_{l'}$, $\#L_j = O(n/l')$.

b) For each $1 \leq r \leq \sqrt{n}$, for each $1 \leq j \leq l'$, broadcast L_j to all processors in the j-th row of the r-th column of $B_{r,q}$'s.
Remark: This takes $O(n/l'+l) = O(n^{3/2}/l+l)$ rounds. As a result, the connected components $A = \{A_1, \ldots, A_s\}$ of F are known in each $B_{r,q}$.

Step 4, Step 5 are as in Algo 3.

The correctness of the algorithm follows from the correctness of Algo 3. The number of rounds in each of the $\log(n)$ levels of recursion is

$$O(n^{3/2} + l + \sqrt{n\log(n)}) = O(n^{3/2}/l + l).$$

Thus the algorithm needs

$$O((n^{3/2}/l + l) \cdot \log(n))$$

rounds.

Acknowledgement: I would like to thank Martin Dietzfelbinger for many helpful discussions.

References

[AK] M.J.Atallah, S.R.Kosaraju: Graph problems on a mesh-connected processor array, Proc. 14th ACM-STOC, 345-353, 1982.

[CLC] F.Y.Chin, J.Lam,L.N.Chen: Efficient parallel algorithms for some graph problems, Comm. ACM 25(9), 659-665, 1982.

[DV] K.Doshi, P.Varman: Efficient graph algorithms using limited communication on a fixed size array of processors, Proc. 4th STACS, Passau, Fed. Rep. Germany, 76-87, 1987.

[M] K.Mehlhorn: Data structures and algorithms, Vol. 2, Springer-Verlag, Berlin Heidelberg New York, 1984.

[NS] D.Nassimi, S.Sahni: Finding connected components and connected ones on a mesh-connected parallel computer, SIAM J. Comp. 9(4), 744-757, 1980.

[P] W.Preilowski: Parallele Algorithmen für lineare Prozessorenarrays, Diplomarbeit, Paderborn, 1987.

[QD] M.J.Quinn, N.Deo: Parallel graph algorithms, Computing Surveys 16(3), 319-348, 1984.

[SS] C.P.Schnorr, A.Shamir: An optimal sorting algorithm for mesh connected computers, Proc. 18th ACM-STOC, 255-263, 1986.