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 $O(n \log(n))$.

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.