Parallel Algorithms for Two Processors Precedence Constraint Scheduling (2003; Jung, Serna, Spirakis)

Maria Serna, Technical University of Catalonia, www.cs.upc.edu/~mjserna

INDEX TERMS: Scheduling, multiprocessor, parallel algorithms, NC class, PRAM algorithm.
SYNONIMS: Optimal scheduling for two processors.

1 PROBLEM DEFINITION

In the general form of *multiprocessor precedence scheduling problems* a set of n tasks to be executed on m processors is given. Each task requires exactly one unit of execution time and can run on any processor. A directed acyclic graph specifies the precedence constraints where an edge from task x to task y means task x must be completed before task y begins. A solution to the problem is a schedule of shortest length indicating when each task is started. The work of Jung, Serna, and Spirakis provides a parallel algorithm (on a PRAM machine) that solves the above problem for the particular case that $m = 2$, that is where there are two parallel processors.

The *two processor precedence constraint scheduling problem* is defined by a directed acyclic graph (dag) $G = (V, E)$. The vertices of the graph represent unit time tasks, and the edges specify precedence constraints among the tasks. If there is an edge from node x to node y then x is an *immediate predecessor* of y. *Predecessor* is the transitive closure of the relation immediate predecessor, and *successor* is its symmetric counterpart. A *two processor schedule* is an assignment of the tasks to time units $1, \ldots, t$ so that each task is assigned exactly one time unit, at most two tasks are assigned to the same time unit, and if x is a predecessor of y then x is assigned to a lower time unit than y. The length of the schedule is t. A schedule having minimum length is an *optimal* schedule. Thus the problem is the following:

Name Two processor precedence constraint scheduling
Input A directed acyclic graph
Output A minimum length schedule preserving the precedence constraints.

Preliminaries The algorithm assume that tasks are partitioned into levels as follows:

(i) Every task will be assigned to only one level

(ii) Tasks having no successors will be assigned to level 1 and

(iii) For each level i, all tasks which are immediate predecessors of tasks in level i will be assigned to level $i + 1$.

Clearly topological sort will accomplish the above partition, and this can be done by an NC algorithm that uses $O(n^3)$ processors and $O(\log n)$ time, see [3]. Thus, from now on, it is assumed that a level partition is given as part of the input. For sake of convenience two special tasks, t_0 and t^* are added, in such a way that the original graph could be taught as the graph formed by all tasks that are successors of t_0 and predecessors of t^*. Thus t_0 is a predecessor of all tasks in the system

(actually an immediate predecessor of tasks in level the highest level $L(G)$) and t^* is a successor of all tasks in the system (an immediate successor of level 1 tasks).

Notice that if two tasks are at the same level they can be paired. But when x and y are at different levels, they can be paired only when neither of them is a predecessor of the other. Let $L(G)$ denote the number of levels in a given precedence graph G. A *level schedule* schedules tasks level by level. More precisely, suppose levels $L(G), \ldots, i+1$ have already been scheduled and there are k unscheduled tasks remaining on level i. When k is even, those tasks with are paired whith each other. When k is odd, $k-1$ of the tasks are paired with each other, while the remaining task may (but not necessarily) be paired with a task from a lower level.

Given a level schedule level i *jumps to level* i' ($i' < i$) if the last time step containing a task from level i also contains a task from level i'. If the last task from level i is scheduled with an empty slot, it is said that level i *jumps to level* 0. The *jump sequence* of a level schedule is the list of levels jumped to. A *lexicographically first jump schedule* is a level schedule whose jump sequence is lexicographically greater than any other jump sequence resulting from a level schedule.

Given a graph G a *level partition* of G is a partition of the nodes in G into two sets in such a way that levels $0, \ldots, k$ are contained in one set (the upper part) denoted by U, and levels $k+1, \ldots, L$ in the other (the lower part) denoted by L. Given a graph G and a level i, the *i-partition* of G (or the partition at level i) is formed by the graphs U_i and L_i defined as U_i contains all nodes x such that level$(x) < i$ and L_i contains all nodes x with level$(x) > i$. Note that each i-partition determines two different level partitions depending on whether level i nodes are assigned to the upper or the lower part. A task $x \in U_i$ is called *free* with respect to a partition at level i if x has no predecessors in L_i.

Auxiliary problems The algorithm for the two processors precedence constraint scheduling problem uses as a building block an algorithm for solving a matching problem in a particular graph class.

A *full convex bipartite graph* G is a triple (V, W, E), where $V = \{v_1, \ldots, v_k\}$ and $W = \{w_1, \ldots, w_{k'}\}$ are disjoint sets of vertices. Furthermore the edge set E satisfies the following property: If $(v_i, w_j) \in E$ then $(v_q, w_j) \in E$ for all $q \geq i$. Thus, from now on it is assumed that the graph is connected.

A set $F \subseteq E$ is a *matching* in the graph $G = (V, W, E)$ iff no two edges in F have a common endpoint. A *maximal matching* is a matching that cannot be extended by the addition of any edge in G. A *lexicographically first maximal matching* is a maximal matching whose sorted list of edges is lexicographically first among all maximal matchings in G.

2 KEY RESULTS

When the number of processors m is arbitrary the problem is known to be NP-complete [8]. For any $m \geq 3$, the complexity is open [6]. Here the case of interest has been $m = 2$. For two processors a number of efficient algorithms has been given. For sequential algorithms see [2, 4, 5] among others. The first deterministic parallel algorithm was given by Helmbold and Mayr [7], thus establishing membership in the class NC. Previously [9] gave a randomized NC algorithm for the problem. Jung, Serna and Spirakis present a new parallel algorithm for the two processors scheduling problem that takes time $O(\log^2 n)$ and uses $O(n^3)$ processors on a CREW PRAM. The algorithm improves the number of processors of the algorithm given in [7] from $O(n^7 L(G)^2)$, where $L(G)$ is the number of levels in the precedence graph, to $O(n^3)$. Both algorithms compute a level schedule that has a lexicographically first jump sequence.

To match jumps with tasks it is used a solution to the problem of computing the lexicographically first matching for a special type of convex bipartite graphs, here called *full convex bipartite graphs*.

A geometric interpretation of this problem leads to the discovery of an efficient parallel algorithm to solve it.

Theorem 1. *The lexicographically first maximal matching of full convex bipartite graphs can be computed in time $O(\log n)$ on a CREW PRAM with $O(\frac{n^3}{\log n})$ processors, where n is the number of nodes.*

The previous algorithm is used to solve efficiently in parallel two related problems.

Theorem 2. *Given a precedence graph G, there is a PRAM parallel algorithm that computes all levels that jump to level 0 in the graph L_i and all tasks in level $i-1$ that can be scheduled together with a task in level i, for $i = 1, \ldots, L(G)$, using $O(n^3)$ processors and $O(\log^2 n)$ time.*

Theorem 3. *Given a level partition of a graph G together with the levels in the lower part in which one task remains to be matched with some other task in the upper part of the graph. There is a PRAM parallel algorithm that computes the corresponding tasks in time $O(\log n)$ using $\frac{n^3}{\log n}$ processors.*

With those building blocks the algorithm for two processor precedence constraint scheduling starts by doing some preprocessing and after that an adequate decomposition that insure that at each recursive call a number of problems of half size are solved in parallel. This recursive schema is the following:

Algorithm Schedule

0. Preprocessing
1. Find a level i such that $|U_i| \leq n/2$ and $|L_i| \leq n/2$
2. Match levels that jump to free tasks in level i.
3. Match levels that jump to free tasks in U_i.
4. If level i (or $i+1$) remain unmatched try to match it with a non free task.
5. Delete all tasks used to match jumps.
6. Apply (1)–(5) in parallel to L_i and the modified U_i.

Algorithm **Schedule** stops whenever the corresponding graph has only one level.

The correction an complexity bounds for algorithm **Schedule** follows from the previous results, leading to:

Theorem 4. *There is an NC algorithm which finds an optimal two processors schedule for any precedence graph in time $O(\log^2 n)$ using $O(n^3)$ processors.*

3 APPLICATIONS

A fundamental problem in many applications is to devise a proper schedule to satisfy a set of constrains. Assigning people to jobs, meetings to rooms, or courses to final exam periods are all different examples of scheduling problems. A key and critical algorithm in parallel processing is the one mapping tasks to processors. In the performance of such an algorithm relies many properties of the system, like load balancing, total execution time, etc. Scheduling problems differ widely in the nature of the constraints that must be satisfied, the type of processors, and the type of schedule desired.

The focus on precedence-constrained scheduling problems for directed acyclic graphs has a most direct practical application in problems arising in parallel processing. In particular in systems where computations are decomposed, prior to scheduling into approximately equal sized tasks and the corresponding partial ordering among them is computed. These constraints must define a directed acyclic graph, acyclic because a cycle in the precedence constraints represents a Catch situation that can never be resolved.

4 OPEN PROBLEMS

The parallel deterministic algorithm for the two processors scheduling problem presented here improves the number of processors of the Helmbold and Mayr algorithm for the problem [7]. However, the complexity bounds are far from optimal: recall that the sequential algorithm given in [5] uses time $O(e + n\alpha(n))$, where e is the number of edges in the precedence graph and $\alpha(n)$ is an inverse Ackermann's function. Such an optimal algorithm might have a quite different approach, in which the levelling algorithm is not used.

Interestingly enough computing the lexicographically first matching for full convex bipartite graphs is in NC, in contraposition with the results given in [1] which show that many problems defined through a lexicographically first procedure in the plane are P-complete. It is an interesting problem to show whether all these problems fall in NC when they are convex.

5 EXPERIMENTAL RESULTS

None is reported.

6 DATA SETS

None is reported.

7 URL to CODE

None is reported.

8 CROSS REFERENCES

First Come First Served Scheduling. List Scheduling. Maximum Matching. Minimum Makespan on Unrelated Machines. Parallel Connectivity and Minimum Spanning trees. Scheduling: Define alpha | beta | gamma Notation, Online Scheduling, Stocastic Scheduling, Resouce Augmentation Analysis, Makespan etc. Shortest Elapsed Time First Scheduling. Stochastic Scheduling. Voltage Scheduling.

9 RECOMMENDED READING

[1] M. Attallah, P. Callahan and M. Goodrich. P-complete geometric problems. *Internat. J. Comput. Geom. Appl.* Vol 3 (4) pp 443-462 1993.

[2] E. G. Coffman, and R. L. Graham. Optimal scheduling for two processors systems. *Acta Informatica*, 1:200–213, 1972.

[3] E. Dekel, D. Nassimi, and S. Sahni. Parallel matrix and graph algorithms. *SIAM Journal of computing*, 10:657–675, 1981.

[4] M. Fujii, T. Kasami, and K. Ninomiya. Optimal sequencing of two equivalent processors. *SIAM Journal of computing*, 17:784–789, 1969.

[5] H. N. Gabow. An almost linear time algorithm for two processors scheduling. *Journal of the ACM*, 29(3):766–780, 1982.

[6] M. R. Garey, and D. S. Johnson. *Computers and Intractability: A Guide to the theory of NP completeness*. Freeman, San Francisco, 1979.

[7] D. Helmbold and E. Mayr. Two processor scheduling is in NC. *SIAM J. Comput.* Vol 16 (4) 1987.

[8] J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10:384–393, 1975.

[9] U. Vazirani and V. Vazirani. Two-processor scheduling problem is in random NC. *SIAM J. Comput.* Vol 18: (4) pp 1140-1148 1989.