The Two-Processor Scheduling Problem

Umesh V. Vazirani*
Vijay V. Vazirani**

TR 89-1000
May 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Currently supported by a PYI Award. Part of this research was supported by NSF Grant 82-04506 and an IBM Doctoral Fellowship.

**Currently supported by a PYI Award, with matching funds from AT&T Bell Laboratories and Sun Microsystems Inc. Part of this research was supported by an IBM Faculty Development Award.

***Part of this research was done while the authors were visiting the IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, in the summer of 1984.

The Two-Processor Scheduling Problem is in Random NC

Umesh V. Vazirani *

University of California, Berkeley

Vijay V. Vazirani **

Cornell University

Abstract:

An efficient parallel algorithm (RNC^2) for the two-processor scheduling problem is presented. An interesting feature of this algorithm is that it finds a highest level first schedule : such a schedule defines a lexicographically first solution to this problem in a natural way. A key ingredient of the algorithm is a generalization of a theorem of Tutte which establishes a one-to-one correspondence between the bases of the Tutte matrix of a graph and the sets of matched nodes in maximum matchings in the graph.

Part of this research was done while the authors were visiting the IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, in the summer of 1984.

* Currently supported by a PYI Award. Part of this research was supported by NSF Grant 82-04506 and an IBM Doctoral Fellowship.

** Currently supported by a PYI Award, with matching funds from AT&T Bell Laboratories and Sun Microsystems Inc. Part of this research was supported by an IBM Faculty Development Award.

1. Introduction

We present an efficient parallel RNC^2 algorithm for the following well studied problem: schedule n unit length jobs subject to given precedence constraints on two identical processors so as to minimize the finish time. Our algorithm uses $O(n^2M(n))$ processors, where $M(n)$ is the number of arithmetic operations required to multiply two $n \times n$ matrices.

There are close connections between two-processor scheduling and matching. The first polynomial time sequential algorithm for two processor scheduling relied on this connection [FKN], and in fact techniques for finding matchings in parallel play an important role in the algorithm presented in this paper. The recent parallel matching algorithms [KUW], [MVV] are based on an algebraic connection established by a theorem of Tutte [Tu]. We present a generalization of this theorem, establishing a one-to-one correspondence between the bases of the Tutte matrix of a graph and the sets of matched nodes in maximum matchings in the graph. This yields an RNC^2 algorithm for computing the lexicographically largest such node-set. This subroutine is central to the parallel scheduling algorithm. The generalization is also of independent interest, and has been used in [MVV] to obtain an RNC^2 algorithm for maximum vertex-weighted matching.

The parallel scheduling algorithm finds a special type of schedule called a highest-level-first (HLF) schedule; a basic theorem of Gabow [Ga1][Ga2] establishes the optimality of such a schedule. In section 5 we show that HLF schedules are "well-behaved" by proving two new properties of such schedules.

Using these properties, the task of computing an HLF schedule is shown to be reducible to parallel invocations of the lexicographically first node-set subroutine.

It is pointed out in section 4 that an HLF schedule is the lexicographically first schedule under a certain natural ordering on the jobs. In general the parallel complexity of a problem is very sensitive to whether an arbitrary solution is acceptable or whether the lexicographically first solution is sought. Two important cases illustrate this point: maximal independent set and depth first search. The problems of finding the lexicographically first maximal independent set or the lexicographically first depth-first search tree are P-complete. Nevertheless, if an arbitrary solution is acceptable, in each case fast parallel algorithms exist. In the case of the maximum matching problem, fast parallel algorithms are known when an arbitrary solution suffices, but the parallel complexity of finding the lexicographically first maximum matching is still unresolved. Using the connection between two-processor scheduling and matching, the parallel algorithm presented in this paper can be interpreted as an algorithm for computing lexicographically first matchings for incomparability graphs under a certain natural ordering on the edges.

The algorithm presented here is randomized in the Las Vegas sense: it always produces the correct answer, but the guarantee on the running time is probabilistic. A preliminary report on this algorithm appeared in [VV]. Subsequently Helmboldt and Mayr [HM] devised an algorithm that avoids the use of randomization. Their algorithm runs in $O(\log^2 n)$ time and uses $O(n^7 L^2)$ processors, where L is the length of the optimal schedule. Their algorithm shares some common features with ours; the proof of correctness of their algorithm relies on the "well-behavedness" properties of HLF schedules that are implicitly assumed in their paper, and explicitly proved here.

There is extensive literature on the two-processor scheduling problem: see [LR] for a general reference. We shall use results from [FKN], [Ga1] and [Ga2].

2. Connection with Matchings

We shall denote the set of jobs by V, and the precedence relations among the jobs by a directed acyclic graph $G(V,E)$.

Definition: A *schedule* is a sequence of sets $S_1, ..., S_t$ that partition V, with $|S_i| \leq 2$ and such that if x precedes y, $x \in S_i$ and $y \in S_j$ then $i<j$. The length of the schedule is t.

It is useful to associate a compatibility graph $H(V,E')$ with the given precedence graph $G(V,E)$. $H(V,E')$ is an undirected graph with an edge between a pair of vertices, x and y, if they are compatible - i.e. there is no directed path from x to y or y to x in G. The following theorem gives the fundamental relationship between matchings and schedules.

Theorem 1 (Fujii, Kasami, and Ninomiya [FKN]): For G and H as defined above, the paired jobs in any *optimal* schedule for G form a *maximum* matching in H. Moreover, any maximum matching in H can be transformed into an optimal schedule for G, leaving the set of unpaired jobs unchanged.

Note that not all maximum matchings in H correspond to valid schedules for G; in order for a maximum matching to correspond to a valid schedule, it should be possible to linearly order the pairs so that all precedence constraints are satisfied. The content of the above theorem of [FKN] is a sequential procedure to transform any maximum matching into a valid schedule without changing the set of unpaired jobs.

The connection between matchings and schedules goes further: in section 5 we show that the lexicographically first maximum matching (under a natural ordering of the edges of H) always corresponds to an optimal schedule. The parallel algorithm presented in this paper finds an optimal schedule by constructing precisely such a lexicographically first matching.

3. The Algebraic Connection

The results of this section are of independent interest; in this section, we will let $G(V,E)$ denote an undirected graph, with $|V|=n$.

Definition: The determinant of matrix A will be denoted by $det(A)$, and its $(i,j)^{th}$ entry will be denoted by a_{ij}. The adjacency matrix D of graph $G(V,E)$ is given by:

$$d_{ij} = \begin{cases} 1 \text{ if } (v_i, v_j) \in E \\ 0 \ \ otherwise \end{cases}$$

The Tutte matrix A of graph $G(V,E)$ is given by:

$$a_{ij} = \begin{cases} x_{ij} \text{ if } (v_i, v_j) \in E \text{ and } i > j \\ -x_{ij} \text{ if } (v_i, v_j) \in E \text{ and } i < j \\ 0 \ \ otherwise \end{cases}$$

Theorem 2 (Tutte [Tu]): Let A be the Tutte Matrix of a graph $G(V,E)$. Then, $\det(A) \neq 0$ iff there exists a perfect matching in G.

The following is a natural generalization of Tutte's Theorem to maximum matchings:

Theorem 3 (Lovász [LP]): Let A be the Tutte matrix of a graph $G(V,E)$, and let m be the number of edges in a maximum matching in G. Then, $\text{rank}(A) = 2m$.

Rabin and Vazirani [RV] gave a simple proof of Theorem 3 using a theorem of Fröbenius [Ko, page 144] stated below. We use their approach to give a further generalization below.

Definition: The *node-set* of a maximum matching M is an n-dimensional 0/1 vector whose i^{th} component is 1 iff vertex v_i is matched in M. Let n_G denote the lexicographically first such node-set.

Notation: We represent a basis of an $n \times n$ matrix A as an n-dimensional 0/1 vector, whose i^{th} component is 1 iff the i^{th} row of A is in the basis. Given n-dimensional vectors u, v, A_{uv} is the submatrix of A obtained by choosing rows corresponding to indices having 1's in u, and columns corresponding to indices having 1's in v. Similarly, G_u is the subgraph of G induced on vertices corresponding to indices having 1's in u. We will denote the number of 1's in u by $\#u$.

Theorem 4: (Frobenius [Ko, page 144]): Let B be an n by n skew-symmetric matrix. Let u, v be n-dim vectors such that $\#u = \#v = \text{rank}(B)$. Then

$$\det(B_{uu})\cdot\det(B_{vv}) = (-1)^{\#u}\det(B_{uv})^2.$$

Theorem 5: Let A be the Tutte matrix of a graph $G(V,E)$. A vector $w \in \{0,1\}^n$ is a basis of A iff w is the node-set of a maximum matching in G.

Proof: Let w be the node set of a maximum matching in G. Since G_w has a perfect matching, by Tutte's theorem $\det(A_{ww}) \neq 0$. Thus the rows of A indexed by w are linearly independent. Moreover by Theorem 3 rank (A) = $\#w$. Thus w is a basis of A.

Conversely, let w be a basis of A. Then, there is a vector $v \in \{0,1\}^n$ such that $\#v = \#w$, and $\det(A_{wv}) \neq 0$. Since $\text{rank}(A) = \#w = \#v$, by Frobenius's theorem $\det(A_{ww}) \neq 0$. Now, by Tutte's theorem, G_w has a perfect matching, and by Theorem 3, this matching is a maximum matching in G. Hence w is the node-set of a maximum matching in G. $\square$

Corollary: The lexicographically first basis of A is n_G.

Theorem 6: There is an RNC^2 algorithm for finding n_G. It requires $(M(n)\log^2 n)$ processors.

Proof: Choose a random substitution S: set of variables in $A \rightarrow [1,...2n]$ to obtain the matrix A^S. Certainly, any basis of A^S is a basis of A. Let $w \in \{0,1\}^n$ be the lexicographically first basis of A. Then, $\det(A_{ww}) \neq 0$. This is a non-vanishing polynomial of degree at most n. Hence, by Schwartz's theorem [Sc], the probability that this polynomial vanishes under the random substitution S:

$$Pr[\det(A^S_{ww}) = 0\ ] \leq \frac{\deg\ (det(A_{ww}))}{2n} \leq \frac{1}{2}.$$

Therefore, with probability at least $\frac{1}{2}$, A^S has the same lexicographically first basis as A. We now use the RNC^2 algorithm of Borodin, van zur Gathen and Hopcroft [BGH] for finding the lexicographically first basis of A. For better processor efficiency, the computations can be done over a finite field (see [RV] for details). This requires $O(M(n)\log^2 n)$ processors, where $M(n)$ is the number of arithmetic operations required to multiply two $n \times n$ matrices.

4. Highest-Level-First Schedules

As before, let V be the set of jobs to be scheduled, and let $G(V,E)$ be a directed acyclic graph that represents the precedence constraints among the jobs.

Definition: The *level* of a job is one plus the length of the longest directed path in G starting at the job.

Notice that all constraints go from higher (numbered) levels to lower levels, and there are no constraints among jobs which are at the same level. Assume that the levels are numbered $L, L-1, ..., 1$.

Definition: A schedule is a *level schedule* if for every level i, there is at most one job at level i which is paired with a job at a lower level, say j. Such a pair (if it exists) is called a *jump*, and the jump of level i is said to be j. If all jobs at level i are paired with jobs at level i or larger, then its jump is i, and if a job remained unpaired at level i, then its jump is 0.

Theorem 7 (Gabow [Ga1]): Any optimal schedule on a dag $G(V,E)$ can be transformed into a level schedule, without changing the set of unpaired jobs.

Definition: The *jump-sequence* of a level schedule is an L-tuple $(p_L, p_{L-1}, \cdots p_1)$ where p_i is the jump of level i.

The lexicographic ordering on jump-sequences induces a partial ordering on level schedules.

Definition: A level schedule is *highest-level-first* (*HLF*) if its jump sequence is lexicographically largest among all valid schedules for G.

Intuitively, such a schedule always jumps to the highest level possible. Moreover, within this level, it jumps to a job that allows subsequent jumps to be highest possible. The importance of such a schedule comes from:

Theorem 8 (Gabow [Ga1]): Any highest-level-first schedule on a dag $G(V,E)$ is optimal.

Remark: Let us assign to each edge in the compatibility graph the ordered pair (h, l) where h is the level of the higher endpoint of the edge and l is the level of the lower endpoint. Assign priorities to the edges by comparing the associated ordered pair; note that edges that run between the same levels have the same priority. Now matchings can be compared lexicographically in the obvious manner. It is easy to check that the paired jobs of any HLF schedule form a lexicographically highest matching in this sense.

5. Further Connections between Matchings and HLF Schedules

In this section we prove two new properties of HLF schedules; these properties show that HLF schedules are "well-behaved", and are essential to proving the correctness of the parallel algorithm. The first property (proved in theorem 9) states that HLF schedules leave unpaired jobs at the lowest possible levels. The second (proved in Theorem 10) states that restricting an HLF schedule to jobs at levels i or greater yields an HLF schedule for the resulting dag.

Definition: The coarse-jump-sequence of a level schedule S is an L-dimensional 0/1 vector $(s_L \cdots s_1)$ such that s_i is 0 iff S leaves a job unpaired on level i.

Note that if S is a level schedule with jump sequence $(p_L, \cdots p_1)$, its coarse-jump-sequence is given by $s_i = 0$ iff $p_i = 0$.

Definition: The *level − set* of a maximum matching M in H is an L-dimensional 0/1 vector whose i^{th} component is 0 iff there is an unmatched vertex at level i. Let l_H denote the lexicographically largest level-set. Recall that n_H is the lexicographically first node-set of a maximum matching in H.

Theorem 9: The coarse-jump-sequence of an *HLF* schedule is lexicographically greater than or equal to the coarse-jump-sequence of any level schedule on G.

Proof: By contradiction. Suppose S is an *HLF* schedule on G with coarse-jump-sequence s, and T is a level schedule whose coarse-jump-sequence, t, is lexicographically larger than s. Let i be the highest level at which t and s differ; then S has an unpaired job at level i and T doesn't. Let k be the number of unpaired jobs in T at levels lower than i, and l the number of unpaired jobs in T at levels higher than i.

Obtain a new dag G' from G as follows: add k new jobs at the first level; put in precedence constraints from each job at level i or higher to each new job. Let S' be an HLF schedule on G'. Notice that any level schedule for G when restricted to levels i or higher (including jumps from these levels to lower levels) is valid for G', and vice-versa. Since S and S' are HLF in G and G' respectively, their jump sequences must agree on level $\geq i$. Therefore S' has at least $l+1$ unpaired jobs. On the other hand T can be modified into a schedule for G' with only l unpaired jobs. This contradicts the optimality of S'.

Lemma 1: The *HLF* coarse-jump-sequence in G is the same as l_H.

Proof: Let M be a maximum matching in H whose level-set is l_H. By theorems 1 and 7 there is a level schedule for G which has the same unpaired vertices as M, and therefore its coarse-jump-sequence is also l_H. The lemma now follows from theorem 9.

Computing the HLF unpaired jobs in G has now been reduced to finding l_H. In lemma 2 we will reduce this to finding n_G, for which an RNC^2 algorithm was given in section 3. Notice that if a maximum matching M achieves node-set n_H, then M leaves at most one vertex unmatched at each level. It is therefore straightforward to obtain the level-set of M.

Lemma 2: Pick any ordering $>$ on the vertices of H such that if $level(x)$ is higher than $level(y)$ then $x > y$. Under this ordering, the level-set of any maximum matching M achieving node-set n_H is l_H.

Proof: Let M be a maximum matching achieving node-set n_H for this ordering, and let N be a maximum matching with level-set l_H. Since both matchings are maximum, the symmetric difference of M and N contains even cycles and even length paths. Each even length path must have its end-points at the same level, otherwise we could improve either M or N. Hence M and N have their unmatched nodes at the same levels. $\square$

Definition: Let $G(V,E)$ be a dag, partitioned into L levels. For $1 \le i \le L$, define G_i to be the dag induced on vertices belonging to levels L, $L-1$, ... i.

For convenience, we number the levels of G_i as $L,\ L-1,\ \ldots\ i$.

Theorem 10: Let the *HLF* jump-sequence for G be $(p_L,\ p_{L-1},\ \ldots\ p_1)$. Then, for $1 \le i \le L$, the *HLF* jump-sequence for G_i is $(q_L,\ q_{L-1},\ \ldots\ q_i)$, where

$$q_j = \begin{cases} p_j & \text{if } p_j \ge i \\ 0 & \text{otherwise} \end{cases}$$

We follow the method of Gabow for modifying schedules. First we introduce the relevant definitions and lemmas from [Ga1].

Definition: A $k-$*schedule* is a level schedule for levels $L,\ \ldots,\ k$ and nodes in lower levels jumped from these levels. (A k-schedule is just a prefix of a complete schedule).

Definition: Let S be a k-schedule, with $level(x) > level(y)$. Say that y is $x-$*ready* in S if x does not precede y and all predecessors of y below $level(x)$ are jumped from above $level(x)$.

Let M be an HLF k-schedule, and let O be a k-schedule that is HLF up to level $k+1$.

Lemma 3 (Gabow): If (x,y) is a jump in O with $level(x) > k$, then y is x-ready in M.

Definition: A chain consists of nodes x_i, $1 \le i \le c+1$, and y_i, $0 \le i \le c$ for $c \ge 1$, where

1. (x_i, y_i) is a jump in O, for $1 \le i \le c$.
2. (y_i, x_{i+2}) is a jump in M, for $0 \le i \le c-1$.

3. level(x_i) = level(y_{i-1}), for $1 \le i \le c+1$.

Adapting Theorem 2.1 of [Ga1] to our setting gives:

Lemma 4: (Gabow): Let M and O be as above; consider a chain with level(x_1) as high as possible, and with c maximal. and with *level* $(x_{c+1}) \ge k$. Let x_1 is paired with u in M and y_0 be paired with x_0 in O; clearly level(u) = level(y_0) = level(x_0) = level(x_1).

1. If there is no jump from x_{c+1} in O, then O can be modified into a valid k-schedule by replacing (x_i, y_i) by (y_{i-1}, x_{i+1}) for i = 1 to c, and replacing (x_0, y_0) and (x_{c+1}, y_{c+1}) by (x_0, x_1) and (y_{c+1}, y_c).
2. If there is no jump from y_c in M, then M can be modified into a valid k-schedule by replacing (y_{i-1}, x_{i+1}) by (x_i, y_i) for i = 1 to c, and replacing (u, x_1) and (v, y_c) by (u, y_0) and (v, x_{c+1}).
3. level(x_{c+1}) = level(y_c) = k, and (x_{c+1}, y_{c+1}) is a jump in O and (y_c, x_{c+2}) is a jump in M. Then O can be modified into a valid k-schedule, by replacing (x_i, y_i) by (y_{i-1}, x_{i+1}) for i = 1 to $c+1$, and replacing (x_0, y_0) by (x_0, x_1).

Proof of Theorem 10: Let S' be an HLF schedule for G_i, and let $(r_L, r_{L-1}, \ldots, r_i)$ be its jump-sequence. Assume for contradiction that $(r_L, r_{L-1}, \ldots, r_i)$ is larger than $(q_L, q_{L-1}, \ldots, q_i)$. Let k be the highest level where the two jump-sequences differ.

Choose k-schedules M and O for $G(V,E)$:

1. M has jump sequence $(p_L, p_{L-1}, \ldots, p_k)$.

2. O has jump sequence $(p_L, \ldots, p_{j+1}, r_j, \ldots, r_k)$.

3. j is as small as possible

4. Among pairs of k-schedules satisfying 1, 2 & 3, M and O have the largest number of common jumps.

If $j = k$ then O has a larger jump-sequence than M, thus contradicting the fact that M is HLF. Otherwise pick jumps (x_1, y_1) in O and (y_0, x_2) in M, with $level(x_1) = level(y_0)$ as high as possible, and $(x_1, y_1) \neq (y_0, x_2)$. Let these jumps be the start of a chain as defined above with c maximal. We first prove that $level(x_c) \geq j$. Assume for contradiction that $level(x_c) < j$. Let $level(x_d) > j > level(x_{d+1}) \geq k \geq i$. Let x be the job that jumps from level j in M. By lemma 3 x_{d+1} is y_{d+1}-ready in M, and therefore x-ready in M. Thus M can be modified by jumping x to y_{d-1}, thus contradicting the fact that M is HLF.

Now let M' and O' denote j-schedules corresponding to M and O respectively. Then M' and O' satisfy the conditions of lemma 4. Therefore either M' or O' can be modified (according to which of the three cases applies to the chain defined above) to increase the number of common jumps or to make O' jump to level p_j from level j. The new pair of j-schedules can be extended to k-schedules thus contradicting condition 3 or condition 4 above.

Corollary: max $\{ i$: j^{th} component of the HLF jump-sequence for G_i is non-zero$\}$ $= p_j$.

Say that a jump-sequence is realizable if there is a level schedule for G with this jump-sequence. Given a jump-sequence, we show how to construct a graph $K(V',E')$ which has a perfect matching if and only if the jump-sequence is realizable, and if so each perfect matching reveals the set of jumps of one such schedule.

$V' = V \cup V_1 \cup \cdots V_L$, where L is the number of levels in G, and the V_i's are disjoint and disjoint from V.

Let $l(i)$ denote the number of vertices at level i;

Let $k(i)$ = number of levels that jump to level i;

If $0 < jump(i) < i$ then $|V_i| = l(i) - k(i) - 1$, and otherwise $|V_i| = l(i) - k(i)$.

$(x,y) \in E'$ if either

1. $x,y \in V$, with $level(x) = i$ and $level(y) = j$, $jump(i) = j$, $0 < j < i$, and x is not a predecessor of y. (these will be called the type 1 edges).

2. $x \in V$, with $level(x) = i$, and $y \in V'$. (these will be called the type 2 edges).

Lemma 6: H has a perfect matching iff the given jump sequence is realizable. Moreover the type 1 edges in the perfect matching are a valid realization of the jump sequence (with the exception of jumps to level 0).

Proof: If the jump sequence is realizable, then H has the following perfect matching: match a type 1 edge only if it is a jump. This leaves exactly the correct number of vertices at level i to be matched with vertices in V_i.

We prove that the type 1 edges in the perfect matching realize the jump sequence by induction on L. This is true for the L^{th} level by the choice of $|V_L|$. Removing V_L and the L^{th} level of V, we obtain a graph with $L-1$ partitions, and are done by induction.

6. The Parallel Algorithm

Procedure COARSE-JUMP-SEQUENCE below computes the coarse-jump-sequence of an HLF schedule for the given directed graph $G(V,E)$. Procedure JUMP-SEQUENCE computes the HLF jump-sequence for $G(V,E)$. Procedure JUMPS computes the jumps for some HLF schedule for $G(V,E)$. Once the jumps are known, it is an easy matter to pair up the remaining jobs to get an HLF schedule.

Procedure COARSE-JUMP-SEQUENCE

1. In parallel topologically sort G to assign levels
2. Assign distinct numbers to the vertices of G, with $\#v > \#w$ if $level(v) > level(w)$.
3. Obtain the compatibility graph H.
4. In parallel compute n_H (under the above numbering).
5. Compute l_H from n_H and output it.

end.

procedure JUMP-SEQUENCE;

For $j = 1$ to L do in parallel:

Find the *HLF* coarse-jump sequence for G_j.

end;

For $j = 1$ to L do in parallel:

$jump(j) \;=\; \max \{ i : i \leq j \;\wedge\; j^{th}$ component of the *HLF* coarse-jump-sequence for G_i is 1 $\}$.

output jump(j);

end;

end procedure;

procedure JUMPS [G];

Compute the *HLF* jump-sequence for G;

Construct an undirected graph K for this jump sequence as in Lemma 6.

Find a perfect matching in H;

Output the type 1 edges in the perfect matching.

end procedure;

Theorem 11: There is an RNC^2 algorithm which finds an optimal two-processor schedule (in particular, an *HLF* schedule) in a dag $G(V,E)$.

Proof: The correctness of procedure COARSE-JUMP-SEQUENCE follows from Theorem 9 and Lemmas 1 and 2; the correctness of procedure JUMP-

SEQUENCE follows from the corollary to Theorem 10. The correctness of procedure JUMPS follows from lemma 5. Procedure JUMPS determines the set of jumps of an *HLF* schedule (except the jumps to level 0). Now, if a level has an even number of jobs remaining, they can be paired off, and if it has an odd number of jobs remaining, one job will remain unpaired. The jobs can also be assigned appropriate time slots.

Each of the procedures can be implemented in RNC^2, yielding an RNC^2 algorithm overall. Two computations require the most number of processors: computing the ranks of n matrices in parallel, and finding a perfect matching. The second task dominates the first, since in the first task, the computations can be done modulo a prime, as elaborated in [RV] (see Theorem 6). The graph H in which a perfect matching is found has $O(n)$ vertices and $O(n^{2)}$ edges. This requires $O(n^2 M(N))$ processors using the matching algorithm of [MVV], which uses the matrix inversion algorithm of [Pa]. Here $M(n))$ denotes the number of arithmetic operations required to multiply two $n \times n$ matrices. The current best bound for $M(n)$ is $O(n^{2.376})$ [CW]. □

Using [FKN] and the RNC^2 algorithm of Karloff [Ka] for upper-bounding the size of maximum matching in a graph, we get an RNC^2 algorithm for upper-bounding the length of the optimal schedule. By running this algorithm and the scheduling algorithm in parallel, until they agree on the size of the optimal sequence, we get the following:

Corollary: There is a Las Vegas parallel algorithm, which in expected time $O(\log^2 n)$ finds an optimal schedule in a dag $G(V,E)$. It runs on $O(n^2 M(n))$

processors.

6. Discussion

An important issue left open in this paper is whether two processor scheduling is in (deterministic) *NC*. Helmboldt and Mayr [HM] have resolved this question affirmatively. This result, together with the *NC* algorithm for transitively directing a comparability graph yields a *NC* algorithm for constructing a maximum matching in an incomparability graph [KVV]. Both the parallel algorithm presented here as well as that due to Helmboldt and Mayr suffer from a large processor complexity - the parallel algorithm of this paper uses $O(n^{4.5})$ processors, and Helmboldt and Mayr's algorithm uses $O(n^7 L^2)$ processors. On the other hand, the problem can be solved in linear time sequentially [Ga2], [GT]. Improving the processor efficiency of the parallel algorithm remains an open problem.

A long-standing open problem in scheduling theory is to resolve the complexity of the k-processor scheduling problem, for fixed k, $k \geq 3$. We conjecture that in contrast to the $k = 2$ case, for $k \geq 3$, this problem is log-space hard for P.

Finally, the problem of finding an *HLF* schedule may be stated as the problem of finding the lexicographically largest maximum matching in an incomparability graph, with a partial ordering on the edges given by the level numbers of their end-points (see remark in Section 4). The general problem of finding the lexicographically first maximum matching is open. Can some of the techniques of this paper be generalized to handle this larger problem?

Acknowledgements

The algorithm presented here benefited greatly from work done with Dexter Kozen on computing the interval representation of interval graphs in parallel. We wish to thank Dexter for his generous help throughout the evolution of this paper. We are also thankful to Ashok Chandra and Michael Rabin for several inspiring discussions.

References

[BGH] Borodin, A., von zur Gathen, J., and Hopcroft, J.E. "Fast Parallel Matrix and GCD Computations,"*Information and Control 52:3*, (March 1982) 241-256 .

[CW] Coppersmith, D., and Winograd, S., "Matrix Multiplication via Arithmetic Progressions," *Proc. 19th STOC Conf.* (1987) 1-6.

[FKN] Fujii, M., Kasami, T., and Ninomiya K. "Optimal Sequencing of Two Equivalent Processors," *SIAM J. of Applied Math. 17* (1969), 784-789.

[Ga1] Gabow, H.N. "An Almost-linear Algorithm for Two-Processor Scheduling," University o Colorado Tech. Report No. CU-CS-169-80, January, (1980).

[Ga2] Gabow, H.N. "An Almost-linear Algorithm for Two-Processor Scheduling," *JACM 29*, 3 (1982), 766-780.

[GT] Gabow, H.N., and Tarjan, R.E. "A Linear-time Algorithm for a Special Case of Disjoint Set Union," *JCSS 30* (1985) 209-221.

[HM] Helmboldt D., and Mayr E. "Two Processor Scheduling is in NC," *Siam Journal of Comp. 16* (1987), 747-759.

[Ka] Karloff, H. "A Randomized Parallel Algorithm for the Odd Set Cover Problem," to appear in *Combinatorica.*

[Ko] Kowalewski, G. "Einfuhrung in die Determinanten Theorie," *Leipzig Verlag von Veit & Comp.* (1909), 144.

[KUW] Karp, R.M., Upfal, E., and Wigderson, A. "Constructing a Maximum Matching is in Random *NC*," *Combinatorica 6:1*, (1986), 35-48.

[KVV] Kozen, D., Vazirani, U.V., and Vazirani, V.V. "NC Algorithms for Comparability, Interval Graphs, and Testing for Unique Perfect Matchings." Fifth FST&TCS Conference, 1985.

[LP] Lovasz, L. and Plummer M.D. *Matching Theory*, Academic Press, Budapest, Hungary.

[LR] Lenstra, J.K., and Rinnoy Kan, A.H.G., "Complexity of Scheduling under Precedence Constraints," *Operations Res. 26*, (1978) 22-35.

[MVV] Mulmuley, K., Vazirani, U.V., and Vazirani, V.V. "Matching is as Easy as Matrix Multiplication," *Combinatorica 7:1* (1987), 105-113.

[Pa] Pan, V., "Fast and Efficient Algorithms for the Exact Inversion of Integer Matrices," *Fifth Annual Found. of Software Technology and Theoretical Computer Science Conf.* (1985).

[RV] Rabin, M.O., and Vazirani, V.V. "Maximum Matchings in General Graphs through Randomization," to appear in *J. Algorithms.*

[Sc] Schwarta, J. T., "Fast Polynomial Algorithms for Verification of Polynomial Identities," *JACM* 27, 4 (1980) pp 701-717.

[Tu] Tutte, W.T. "The Factorization of Linear Graphs," *J. London Math. Soc. 22* (1974), 107-111.

[VV] Vazirani, U.V., and Vazirani, V.V. "The Two-Processor Scheduling Problem is in R-NC," *Proc. Seventeenth Ann. Symp. on Theory of Computing* (1985).