Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

B. Veltman, B.J. Lageweg, J.K. Lenstra

Multiprocessor scheduling with communication delays

Department of Operations Research, Statistics, and System Theory Report BS-R9018 June

---

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

Multiprocessor Scheduling with Communication Delays

B. Veltman, B.J. Lageweg
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

J.K. Lenstra
Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

This paper addresses certain types of scheduling problems that arise when a parallel computation is to be executed on a multiprocessor. We define a model that allows for communication delays between precedence-related tasks, and propose a classification of various submodels. We also review complexity results and optimization and approximation algorithms that have been presented in the literature.

1980 Mathematics Subject Classification (1985 Revision): 90B35.
Key Words & Phrases: scheduling, parallel processors, communication delays, algorithms, complexity.

1. Introduction

Over the past decade, distributed memory architectures passed from the state of theoretical models to that of real machines. To take advantage of the inherent parallelism of these architectures, new allocation and scheduling problems have to be solved. These problems differ from their classical variants mainly in that interprocessor communication delays have to be taken into account.

In this paper, we address such problems in the context of deterministic machine scheduling theory. Scheduling theory in general is concerned with the optimal allocation of scarce resources to activities over time. A *processor*, or machine, is a resource that can perform at most one activity at any time. The activities are commonly referred to as *tasks*, or jobs. The problems we consider are *deterministic* in the sense that all the information that defines an instance is known with certainty in advance.

We will assume that there are *data dependencies* among the tasks. That is, some tasks produce output that is required by other tasks as input. Such dependencies define a *precedence relation* on the task set. Whenever the two tasks of a (predecessor, successor) pair are assigned to different processors, a *communication delay* occurs. For this class of problems, we formulate a

Report BS-R9018
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

model, we propose a classification that extends the scheme of Graham, Lawler, Lenstra and Rinnooy Kan [1979], and we review the available literature. We also briefly mention a practical project that motivated this research.

In the literature, we distinguish two basically different approaches to handle communication delays. The first approach formulates the problem in graph theoretic terms and is called the *mapping* problem [Bokhari, 1981]. The program graph is regarded as an undirected graph, where the vertices correspond to program modules and an (undirected) edge indicates that the adjacent modules interact, that is, communicate with each other. The multiprocessor architecture is also regarded as an undirected graph, with nodes corresponding to processors. Processors are assigned to program modules. A mapping aims at reducing the total interprocessor communication time and balancing the workload of the processors, thus attempting to find an allocation that minimizes the overall completion time.

The second approach considers the allocation problem as a pure *scheduling* problem. It regards the program graph as an acyclic directed graph. Again, the vertices represent the program modules, but a (directed) arc indicates a one-way communication between a (predecessor, successor) pair of modules. A *schedule* is an allocation of each task (module) to a time interval on one or more processors, such that, among others, precedence constraints and communication delays are taken into account. It aims at minimizing the completion time.

In this paper we take the second approach. Eventually, it may be desirable to combine the two approaches in allocating a parallel program to a multiprocessor. In that case, a first step would schedule the program modules on a virtual architecture graph, and a second step would find a mapping of the virtual architecture graph onto the physical architecture of the multiprocessor [Kim, 1988].

2. The processor model

The multiprocessor chosen consists of a collection of *m processors*, each provided with a *local memory* and mutually connected by an *intercommunication network*. The multiprocessor architecture can be represented by an undirected graph. The nodes of this graph correspond to the processors of the architecture. Transmitting data from one processor to another is considered as an independent event, which does not influence the availability of the processors on the transmittal path. In case of a shared memory, the assumption of having local memory only overestimates the communication delays.

3. The program model

A parallel program is represented by means of an acyclic directed graph. The nodes of this program graph correspond to the modules in which the program is decomposed; they are called *tasks*. Each task produces *information*, which is in whole or in part required by one or more other tasks. These data dependencies impose a *precedence relation* on the task set; that is, whenever a task requires information, it has to succeed the tasks that deliver this information. The arcs of the graph represent these precedence constraints. The transmittal of information may induce several sorts of *communication delays*, which will be discussed in the next section. *Task duplication*, that is, the creation of copies of a task, might reduce such communication delays.

In general, a task can be processed on various subgraphs of the multiprocessor graph. We

assume that, for each task, a collection of subgraphs on which it can be processed is specified and that, for each task and each of its feasible subgraphs, a corresponding processing time is given. If the processors of the architecture are identical, then for each task the processing times related to isomorphic subgraphs are equal. For instance, one may think of a collection of subhypercubes of a hypercube system of processors. Another possibility occurs when each task can be processed on any subgraph of a given task-dependent size.

If *preemption* is allowed, then the processing of any operation may be interrupted and resumed at a later time. Although task splitting may induce communication delays, it may also decrease the cost of a schedule with respect to one or more criteria. We will not explore the aspect of communication delays that are induced by preemption in detail, but concentrate on communication delays in between precedence-related tasks.

4. Communication

The information a task needs (or produces) has to be (or becomes) available on all the processors handling this task. The size of this data determines the communication times.

If two tasks J_k and J_l both succeed a task J_j, then they might partly use the same information from task J_j. Under the condition that the memory capacity of a processor is adequate, only one transmission of this common information is needed if J_k and J_l are scheduled on the same subgraph of the multiprocessor graph. It is therefore important to determine the *data set* a task needs from each of its predecessors. The transfer of data between J_j and J_k can be represented by associating a data set with the arc (J_j,J_k) of the transitive closure of the program graph. This would generally lead to the specification of $\Theta(n^2)$ sets, if there are n tasks. Another possibility is to associate two sets $IN(j)$ and $OUT(j)$ with each task J_j, representing the data that this task requires and delivers, respectively. This requires $\Theta(n)$ sets. The intersection $OUT(j) \cap IN(k)$ gives the data dependency of tasks J_j and J_k.

Each information set has a *weight*, which is specified by a function $c: 2^D \rightarrow \mathbb{N}$, where D is the set containing all information. This function gives the time needed to transmit data from one processor to another, regarded as independent of the processors involved. Let $U \in 2^D$ be a data set and let $\{U_1, U_2, \ldots, U_u\}$ be a partition of U. We assume that U can be transmitted in such a way that $\cup_{i=1}^{t} U_i$ is available when a time period of length at most $c(\cup_{i=1}^{t} U_i)$ has elapsed, for each t with $1 \leqslant t \leqslant u$. We also assume that $c(\varnothing)=0$ and that $c(U) \leqslant c(W)$ for all $U \subset W \in 2^D$. These conditions state that a data set U can be transmitted in such a way that a subset of U becomes available no later than when this subset would be transmitted on its own.

Interprocessor *communication* occurs when a task J_k needs information from a predecessor J_j and makes use of at least one processor that is not used by J_j. Let M_i be such a processor. Let $F(j)$ denote the set of successors of J_j and, given a schedule, let $P(k,i)$ denote the set of tasks scheduled on M_i before and including J_k. Prior to the execution of J_k, the data set $U(i,j,k) = \cup_{l \in F(j) \cap P(k,i)} (OUT(j) \cap IN(l))$ has to be transmitted to M_i, since not only J_k but also each successor of J_j that precedes J_k on M_i requires its own data set. The time gap in between the completion of J_j (at time C_j) and the start of J_k (at time S_k) has to allow for the transmission of $U(i,j,k)$. The communication time is given by $c(U(i,j,k))$. For feasibility it is required that $S_k - C_j \geqslant c(U(i,j,k))$. At the risk of laboring the obvious, let it be mentioned that the communication time is schedule-dependent.

Sometimes one wishes to disregard the data sets and simply to associate a communication

delay with each pair of tasks. That is, a (predecessor, successor) pair of tasks (J_j, J_k) assigned to different processors needs a communication time of a given duration c_{jk}. The communication time is of length c_{j*} if it depends on the broadcasting task only, it is of length c_{*k} if it depends on the receiving task only. Finally, it may be of constant length c, independent of the tasks.

5. Classification

In general, m processors M_i $(i = 1, \ldots, m)$ have to process n tasks J_j $(j = 1, \ldots, n)$. A schedule is an allocation of each task to a time interval on one or more processors. A schedule is *feasible* if no two of these time intervals on the same processor overlap and if, in addition, it meets a number of specific requirements concerning the processor environment and the task characteristics (e.g., precedence constraints and communication delays). A schedule is *optimal* if it minimizes a given optimality criterion. The processor environment, the task characteristics and the optimality criterion that together define a problem type, are specified in terms of a three-field classification $\alpha \mid \beta \mid \gamma$, which is specified below. Let $\circ$ denote the empty symbol.

5.1. *Processor environment*

The first field $\alpha = \alpha_1 \alpha_2$ specifies the processor environment. The characterization $\alpha_1 = P$ indicates that the processors are *identical parallel processors*. The characterization $\bar{P}$ indicates that, in addition, the number of processors is at least equal to the number of tasks: $m \geqslant n$.

If α_2 is a positive integer, then m is a constant, equal to α_2; it is specified as part of the problem type. If $\alpha_2 = \circ$, then m is a variable, the value of which is specified as part of the problem instance.

5.2. *Task characteristics*

The second field $\beta \subset \{\beta_1, \ldots, \beta_7\}$ indicates a number of task characteristics, which are defined as follows.

1. $\beta_1 \in \{prec, tree, chain, \circ\}$.

   $\beta_1 = prec$: A *precedence relation* $\rightarrow$ is imposed on the task set due to data dependencies. It is denoted by an acyclic directed graph G with vertex set $\{1, \ldots, n\}$. If G contains a directed path from j to k, then we write $J_j \rightarrow J_k$ and require that J_j has been completed before J_k can start.

   $\beta_1 = tree$: G is a *rooted tree* with either outdegree at most one for each vertex or indegree at most one for each vertex.

   $\beta_1 = chain$: G is a collection of vertex-disjoint chains.

   $\beta_1 = \circ$: No data dependencies occur, so that the precedence relation is empty.

2. $\beta_2 \in \{com, c_{jk}, c_{j*}, c_{*k}, c, c = 1, \circ\}$

   This characteristic concerns the communication delays that occur due to data dependencies. To indicate this, one has to write β_2 directly after β_1.

   $\beta_2 = com$: Communication delays are derived from given data sets and a given weight function, as described in Section 4. In all the other cases, the communication delays are directly specified.

   $\beta_2 = c_{jk}$: Whenever $J_j \rightarrow J_k$ and J_j and J_k are assigned to different processors, a communication delay of a given duration c_{jk} occurs.

   $\beta_2 = c_{j*}$: The communication delays depend on the broadcasting task only.

$\beta_2 = c_{*k}$: The communication delays depend on the receiving task only.
$\beta_2 = c$: The communication delays are equal.
$\beta_2 = c = 1$: Each communication delay takes unit time.
$\beta_2 = \circ$: No communication delays occur (which does not imply that no data dependencies occur).

3. $\beta_3 \in \{dup, \circ\}$.
$\beta_3 = dup$: Task duplication is allowed.
$\beta_3 = \circ$: Task duplication is not allowed.

4. $\beta_4 \in \{any, set_j, size_j, cube_j, fix_j, \circ\}$.
$\beta_4 = any$: Each task can be processed on any subgraph of the multiprocessor graph.
$\beta_4 = set_j$: Each task has its own collection of subgraphs of the multiprocessor graph on which it can be processed.
$\beta_4 = size_j$: Each task can be processed on any subgraph of a given task-dependent size.
$\beta_4 = cube_j$: Each task can be processed on a subhypercube of given task-dependent dimension.
$\beta_4 = fix_j$: Each task can be processed on exactly one subgraph.
$\beta_4 = \circ$: Each task can be processed on any single processor.

5. $\beta_5 \in \{\circ, p_j = 1\}$.
$\beta_5 = \circ$: For each task and each subgraph on which it can be processed, a processing time is specified.
$\beta_5 = p_j = 1$: Each task has a unit processing requirement.

6. $\beta_6 \in \{pmtn, \circ\}$.
$\beta_6 = pmtn$: Preemption of tasks is allowed.
$\beta_6 = \circ$: Preemption is not allowed.

7. $\beta_7 \in \{c, c = 1, \circ\}$.
This characteristic concerns the communication delays that occur due to preemption. To indicate this, one has to write β_7 directly after β_6.
$\beta_7 = c$: When a task is preempted and resumed on a different processor, a communication delay of constant length occurs.
$\beta_7 = c = 1$: Each communication delay takes unit time.
$\beta_7 = \circ$: No communication delays occur.

5.3. *Optimality criterion*

The third field γ refers to the optimality criterion. In any schedule, each task J_j has a *completion time* C_j. A traditional optimality criterion involves the minimization of the *maximum completion time* or *makespan* $C_{\max} = \max_{1 \leqslant j \leqslant n} C_j$. Another popular criterion is the *total completion time* $\Sigma C_j = \Sigma_{j=1}^{n} C_j$.

The optimal value of γ will be denoted by γ^*, and the value produced by an (approximation) algorithm A by $\gamma(A)$. If $\gamma(A) \leqslant \rho\gamma^*$ for all instances of a problem, then we say that A is a ρ-approximation algorithm for the problem.

6. Literature review

Practical experience makes it clear that some computational problems are easier to solve than others. Complexity theory provides a mathematical framework in which computational

problems can be classified as being *solvable in polynomial time* or $\mathcal{NP}$*-hard.* The reader is referred to the book by Garey and Johnson [1979] for a detailed treatment of the subject. In reviewing the literature, we will assume that the reader is familiar with the basic concepts of complexity theory. As a general reference on sequencing and scheduling, we mention the survey of deterministic machine scheduling theory by Lawler, Lenstra, Rinnooy Kan and Shmoys [1989], which updates the previous survey by Graham, Lawler, Lenstra and Rinnooy Kan [1979].

6.1. *Single-processor tasks and communication delays*

Colin and Chrétienne [1990] address a problem that arises in the case of scheduling tasks on an idealized distributed multiprocessor system. They investigate $\overline{P}\,|prec, c_{jk}, dup\,|\,C_{\max}$ where the communication delays are small in the sense that for all J_k we have $\min\{p_j\,|\,J_k \in F(j)\} \geqslant \max\{c_{jk}\,|\,J_k \in F(j)\}$. A *critical path*-like algorithm is presented, which is shown to construct an optimal schedule in polynomial time. First, a lower bound b_j on the starting time is computed for each task J_j. Tasks without predecessors get a zero lower bound. For each task J_k that has no lower bound yet but whose predecessors do have lower bounds, let i be such that $b_i + p_i + c_{ik} = \max\{b_j + p_j + c_{jk}\,|\,J_j \rightarrow J_k\}$ and define $b_k = \max\{b_i + p_i, \max\{b_j + p_j + c_{jk}\,|\,J_j \rightarrow J_k, J_j \neq J_i\}\}$. In the second step of the algorithm, a schedule is built such that each task and its duplication are scheduled to start at their lower bound. An arc $J_j \rightarrow J_k$ is *critical* if $b_j + p_j + c_{jk} > b_k$. The subgraph of the precedence graph consisting of all critical arcs is shown to be a spanning forest. Therefore, the assignment of each path of this subgraph to a distinct processor leads to an optimal schedule. The algorithm runs in $O(n^2)$ time.

This work extends the paper by Chrétienne [1989], who studies $\overline{P}\,|tree, c_{jk}\,|\,C_{\max}$. He shows that, if the maximum communication delay is at most equal to the minimum processing time of any task, the scheduling problem is solvable in polynomial time. He gives an $O(n)$ algorithm for problem instances with a rooted tree as a precedence relation, in which each vertex has outdegree at most one. Due to the restriction on communication times, there exist an optimal schedule such that for any task J_j the tasks immediately preceding J_j are assigned to distinct processors, and one immediate predecessor is assigned to the same processor to which J_j itself is assigned. The algorithm has a recursive structure. Leaves are assigned to distinct processors, and each root J_j of a subtree is assigned to a processor that executes one of its immediate predecessors, such that the partial schedule itself is optimal.

Papadimitriou and Yannakakis [1988] show by a reduction from the CLIQUE problem that $\overline{P}\,|prec, c, p_j = 1\,|\,C_{\max}$ is $\mathcal{NP}$-hard. They also present a polynomial-time 2-approximation algorithm for $P\,|prec, c_{j^*}, dup\,|\,C_{\max}$. This rather complicated algorithm starts by assigning a lower bound b_j on the starting time to each task J_j, as follows. Zero lower bounds are assigned to source tasks. For any task J_k other than a source, consider its predecessors, and for each predecessor J_j compute $f_j = b_j + p_j + c_{j^*}$. Sort the predecessors in nonincreasing order of f_j. Next, determine the smallest integer λ satisfying $f_{j_i} \geqslant \lambda \geqslant f_{j_{i+1}}$ with $f_{j_i} > f_{j_{i+1}}$ and such that a subset of $\{J_{j_1}, \ldots, J_{j_i}\}$ of *critical* tasks can be scheduled within a makespan of λ on one processor with their starting times at least equal to their lower bounds. These tasks are critical in the sense that copies of them have to precede any duplicate of J_k on the processor on which this duplicate is executed. The lower bound b_k will be equal to λ. Once the information of all the critical tasks that precede J_k is avaible, this task itself can be executed after a time b_k has elapsed. It is

observed that the information for the critical tasks becomes available no later than time b_k, so that J_k can start no later than $2b_k$.

Rayward-Smith [1987A] allows preemption and studies $\overline{P}\,|\,pmtn,c\,|\,C_{\max}$. He observes that the communication delays increase $C^*_{\max}$ by at most $c-1$. Thus, $\overline{P}\,|\,pmtn,c=1\,|\,C_{\max}$ is solvable in polynomial time by McNaughton's *wrap-around rule* [McNaughton, 1959]. Surprisingly, for any fixed $c \geqslant 2$, the problem is $\mathcal{NP}$-hard, which is proved by a reduction from 3-PARTITION. For the special case that all processing times are at most $C^*_{\max}-c$, the wrap-around algorithm will also yield a valid c-delay schedule.

Rayward-Smith [1987B] shows by a reduction from $P\,|\,prec,p_j=1\,|\,C_{\max}$ that $P\,|\,prec,c=1,p_j=1\,|\,C_{\max}$ is $\mathcal{NP}$-hard. The quality of *greedy* schedules (G) is analyzed. A schedule is said to be greedy if no processor remains idle if there is a task available; list scheduling, for example, produces greedy schedules. It is proved that $C_{\max}(G)/C^*_{\max} \leqslant 3-2/m$. To this end, various concepts are introduced. The *depth* of a node is defined as the number of nodes on a longest path from any source to that node. A *layer* of a digraph comprises all nodes of equal depth. A digraph is *layered* if every node that is not a source has all of its parents in the same layer. A layered digraph is *(n,m)-layered* if it has n layers, all terminal nodes are in the nth layer, and m layers are such that all of their nodes have more than one parent. A precedence relation is *(n,m)-layered* if the corresponding directed graph is (n,m)-layered. It takes at least time $n+m$ to schedule tasks with (n,m)-layered precedence constraints. Given a greedy schedule, let t be a point in time when one or more processors are idle. The tasks processed after t have at least one predecessor processed at $t-1$ or t. Moreover, if all processors are idle at t, then every task processed after t must have at least two predecessors processed at $t-1$. Therefore, from a greedy schedule, a layered digraph can be extracted. Some computations then yield the above result.

Lee, Hwang, Chow and Anger [1988] consider a variant of $P\,|\,prec,c_{jk}\,|\,C_{\max}$. A *distance* is given for each pair of processors. Each communication delay is the product of the distance in between the processors to which two precedence-related tasks J_j and J_k are assigned, and the number c_{jk}. A simple worst-case bound is obtained for their *earliest ready task* heuristic (*ERT*): $C_{\max}(ERT) \leqslant (2-1/m)C_{\max}+C_{\text{com}}$, where $C_{\max}$ is the optimal makespan without considering communication delays and C_{com} is the maximum communication delay in any chain of tasks. The *ERT* algorithm recursively chooses among the available tasks one that can be processed earliest. Hwang, Chow, Anger and Lee [1989] is a rewritten version of this paper.

Kim [1988] also studies $P\,|\,prec,c_{jk}\,|\,C_{\max}$. His approach starts by reducing the program graph, by merging nodes with high internode communication cost through the iterative use of a *critical path* algorithm. This (undirected) graph is then mapped to a multiprocessor graph by mapping algorithms. Numerical results are given.

Sarkar [1989] defines a graphical representation for parallel programs and a cost model for multiprocessors. Together with frequency information obtained from execution profiles, these models give rise to a scheme for compile-time cost assignment of execution times and communication sizes in a program. Most attention is payed to the partitioning of a parallel program, which is outside the scope of this paper. As to scheduling, Sarkar shows for a *runtime scheduling* algorithm (*RS*) with restriction to $P\,|\,prec\,|\,C_{\max}$ that $C_{\max}(RS)/C^*_{\max} \leqslant 2-1/m$. He also proves that $P\,|\,prec,c_{jk}\,|\,C_{\max}$ is $\mathcal{NP}$-hard in the strong sense. Not surprisingly, since this result is dominated by Rayward-Smith [1987B].

6.2. *Multiprocessor tasks*

The problems and algorithms mentioned above deal with tasks that are processed on a single processor and focus on communication delays. The following papers disregard the notion of communication and concentrate on tasks that require a subgraph of the multiprocessor graph.

Chen and Lai [1988A] present a worst-case analysis of *largest dimension, largest processing time list scheduling* (*LDLPT*) for $P \mid cube_j \mid C_{\max}$. They show that $C_{\max}(LDLPT)/C^*_{\max} \leqslant 2-1/m$. *LDLPT* scheduling is an extension of Graham's *largest processing time scheduling* algorithm (*LPT*) [Graham, 1966]. It considers the given tasks one at a time in lexicographical order of nonincreasing dimension of the subcubes and processing times, with each task assigned to a subcube that is earliest available.

For the preemptive problem $P \mid cube_j, pmtn \mid C_{\max}$, Chen and Lai [1988B] give an $O(n^2)$ algorithm that produces a schedule in which each task meets a given deadline, if such a schedule exists. The algorithm considers the tasks one at a time in order of nonincreasing dimension. It builds up a *stairlike* schedule. A schedule is stairlike if a nonincreasing function $f:\{1,\ldots,m\}\rightarrow\mathbb{N}$ exists such that each processor M_i is busy up to time $f(i)$ and idle afterwards. The number of preemptions is at most $n(n-1)/2$. By binary search over the deadline values, an optimal schedule is obtained in $O(n^2(\log n+\log\max_j p_j))$ time.

Van Hoesel [1990] also studies $P \mid cube_j, pmtn \mid C_{\max}$ and presents an $O(n\log n)$ algorithm to decide whether the tasks can be scheduled within a given deadline T. Instead of building up stairlike schedules, this algorithm produces *pseudo-stairlike* schedules. Given a schedule, let t_i be such that processor M_i is busy for $[0,t_i]$ and free for $[t_i,T]$. A schedule is pseudo-stairlike if $t_i<t_h<T$ implies $h<i$, for any two processors M_h and M_i. Again, the tasks are ordered according to nonincreasing dimension. Dealing with J_j, the algorithm recursively searches for the highest i such that $p_j>T-t_i$. It schedules J_j on processors $M_{i-(2^{d_j}-1)},\ldots,M_i$ in the time slot $[t_i,T]$, and on $M_{i+1},\ldots,M_{i+2^{d_j}-1}$ in the time slot $[t_{i+1},p_j-(T-t_i)]$. By a combination of this algorithm and *binary search*, $C^*_{\max}$ can be determined in $O(n\log n\log(n+\max_j p_j))$ time. Furthermore, since each task except the first is preempted at most once, the algorithm creates no more than $n-1$ preemptions, and this bound is tight.

Blazewicz, Weglarz and Drabowski [1984] propose an $O(n\log n)$ algorithm for solving $P \mid size_j, pmtn \mid C_{\max}$, where the tasks require either one or two processors for processing. An initial step computes $C^*_{\max}$ without giving an optimal schedule. Subsequently, the 2-processor tasks are scheduled using McNaughton's *wrap-around rule* [McNaughton, 1959]. A modification of this rule schedules the single-processor tasks one at a time in order of nonincreasing processing times. The following paper extends this result.

Blazewicz, Drabowski and Weglarz [1986] present an $O(n)$ algorithm for solving $P \mid size_j, p_j=1 \mid C_{\max}$, where the tasks require either one or k processors. After calculating the optimal makespan, it schedules the k-processor tasks and next the single-processor tasks. For the problem with sizes belonging to $\{1,2,\ldots,k\}$, an *integer programming* formulation leads to the observation that for fixed k the problem is solvable in polynomial time. However, if k is specified as part of the problem instance, then the problem is strongly $\mathcal{NP}$-complete. For the preemptive case $P \mid size_j, pmtn \mid C_{\max}$, where the tasks require either one or k processors, a modification of McNaughton's *wrap-around rule* [McNaughton, 1959] leads to an $O(n\log n)$ algorithm. Similar to Blazewicz, Weglarz and Drabowski [1984], an initial step computes $C^*_{\max}$. A *linear programming* formulation shows that for any fixed number of processors the problem

$Pm \mid size_j,pmtn \mid C_{max}$ with sizes belonging to $\{1,2,\ldots,k\}$ is solvable in polynomial time.

Bozoki and Richard [1970] study $P \mid fix_j \mid C_{max}$. They concentrate on *incompatibility*, where two tasks are said to be incompatible if they have at least one processor in common. A *branch and bound* algorithm is presented. Lower bounds for the optimal makespan are the maximum amount of processing time that is required by a single processor, and the maximum amount of processing time required by tasks that are mutually incompatible. Upper bounds are obtained by list scheduling according to priority rules such as *shortest processing time* (*SPT*) and *maximum degree of competition* (*MDC*). The degree of competition of a task represents the number of tasks incompatible with it. *MDC* gives tasks with large degree of competition priority over tasks with low degree, breaking ties by use of *SPT*. In branching, an *acceptable* subset of tasks that yield smallest lower bounds is selected at each decision moment t. A set of tasks is acceptable if the tasks are mutually compatible, each task of the set is compatible with each task that is in process at time t, and each task is incompatible with at least one task terminating at t.

Du and Leung [1989] show that $P2 \mid chain,size_j \mid C_{max}$ and $P5 \mid size_j \mid C_{max}$ with sizes belonging to $\{1,2,3\}$ are strongly $\mathcal{NP}$-complete. A *dynamic programming* approach leads to the observation that $P2 \mid any \mid C_{max}$ and $P3 \mid any \mid C_{max}$ are solvable in pseudopolynomial time. Arbitrary schedules for instances of these problems can be transformed into so called *canonical schedules*. A canonical schedule for the machine environment with two processors is one that first processes the tasks using both processors. Such a canonical schedule for two processors is completely determined by three numbers: the total execution times of the single-processor tasks on processor M_1 and M_2 respectively, and the total execution time of the 2-processor tasks. In case of three processors, similar observations are made. These characterizations are the basis for the development of the pseudopolynomial algorithms. The problem $P4 \mid any \mid C_{max}$ remains open; no pseudopolynomial algorithm is given. For the preemptive case, they prove that $P \mid any,pmtn \mid C_{max}$ is strongly $\mathcal{NP}$-complete by a reduction from 3-PARTITION. With restriction to two processors, $P2 \mid any,pmtn \mid C_{max}$ is still $\mathcal{NP}$-complete, as is shown by a reduction from PARTITION. Using a result of Blazewicz, Drabowski and Weglarz [1986], Du and Leung show that for any fixed number of processors $Pm \mid any,pmtn \mid C_{max}$ is also solvable in pseudopolynomial time. The basic idea of the algorithm is as follows. To each schedule S of $Pm \mid any,pmtn \mid C_{max}$, there is a corresponding instance of $P \mid size_j,pmtn \mid C_{max}$ with sizes belonging to $\{1,\ldots,k\}$, in which a task J_j is an l-processor task if it uses l processors with respect to S. An optimal schedule for the latter problem can be found in polynomial time. All that is needed is to generate optimal schedules for all instances of $P \mid size_j,pmtn \mid C_{max}$ that correspond to schedules of $Pm \mid any,pmtn \mid C_{max}$, and choose the shortest among all. It is shown by a dynamic programming approach that the number of schedules generated can be bounded from above by a pseudopolynomial function of the size of $Pm \mid any,pmtn \mid C_{max}$.

7. SCHEDULINK

The work presented here has been carried out at the CWI in Amsterdam as part of the *ScheduLink* subproject within the *ParTool* project. The latter project aims at the creation of a parallel development environment, which is a set of integrated methods and tools that enable the development of programs for parallel processors, with a strong emphasis on programs for scientific and technical computations. The subproject ScheduLink will provide in one of these

tools. It concerns the design and analysis of methods for the scheduling of a given computation graph on a given processor model, allowing for communication delays. Below we will specify the scheduling problems we are studying.

The multiprocessor chosen consists of a collection of m identical parallel processors, each provided with a local memory and mutually connected by an intercommunication network. The multiprocessor architecture is represented by an undirected graph.

Transmitting data does not influence the availability of the processors but is an independent event. One can therefore regard the multiprocessor graph as a complete graph. Yet, by assuming the processor graph to be complete, one disregards routing problems that may occur during the transmission of data. If, for instance, three processors M_1, M_2, M_3 are linearly connected (e.g., $\{M_1,M_2\}$ and $\{M_2,M_3\}$ are edges), then the simultaneous transmissions of data from M_1 and M_2 to M_3 will influence each other. In general, it will take more than the time represented by the model. However, in the type of application under consideration several good routes are usually available, so that the communication time between two processors is practically independent of the chosen route.

The memory size of a processor is large enough to handle each of its tasks. The decomposition of a parallel program has to satisfy this property. It is also desirable that information which is available on a certain processor remains available throughout the entire process. Otherwise the unnecessary transmission of data might cause a delay. If, for instance, tasks J_2 and J_3 need the same information from task J_1 and are scheduled on the same processor, then one transmission of data would be sufficient if the local memory size of the processor is large enough. It remains to be seen whether this assumption is realistic. In case of a shared memory, our assumption of having local memory only overestimates the communication delays.

We assume that the multiprocessor is represented by a complete graph of identical processors. Hence, it suffices to concentrate on the sizes of the subgraphs of the multiprocessor graph and to make no distinction concerning the identities of the processors belonging to a subgraph. In the most general case we consider, each task can be processed on all subgraphs of a predetermined size. These sizes tend to be small with respect to the total number of processors available. Execution times on the subgraphs do not differ, because all the processors are identical, so that each task J_j requires a given processing time p_j.

Preemption of a task produces high communication costs and will therefore not be allowed. A task, once started, has to be processed without interruption until it is finished.

With the notation defined and used in the previous sections, the problem we consider in its most general form is denoted by $P\,|prec,com,dup,size_j\,|\,C_{\max}$. This model is different and more general than the models that have been considered in the literature. In the first place, by the combination of communication delays and multiprocessor tasks and, secondly, by the specification of communication delays by means of data sets.

ACKNOWLEDGEMENT

The Partool project is partially supported by SPIN, a Dutch computer science stimulation program.

REFERENCES

J. BLAZEWICZ, M. DRABOWSKI, J. WEGLARZ (1986). Scheduling multiprocessor tasks to

minimize schedule length. *IEEE Trans. Comput. C-35*, 389-393.

J. BLAZEWICZ, J. WEGLARZ, M. DRABOWSKI (1984). Scheduling independent 2-processor tasks to minimize schedule length. *Inform. Process. Lett. 18*, 267-273.

S.H. BOKHARI (1981). On the mapping problem. *IEEE Trans. Comput. C-30*, 207-214.

G. BOZOKI, J.P. RICHARD (1970). A branch-and-bound algorithm for the continuous-process task shop scheduling problem. *AIIE Trans. 2*, 246-252.

G.I. CHEN, T.H. LAI (1988A). Scheduling independent jobs on hypercubes. *Proc. Conf. Theoretical Aspects of Computer Science*, 273-280.

G.I. CHEN, T.H. LAI (1988B). Preemptive scheduling of independent jobs on a hypercube. *Inform. Process. Lett. 28*, 201-206.

P. CHRÉTIENNE (1989). A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints. *European J. Oper. Res. 43*, 225-230.

J.Y. COLIN, P. CHRÉTIENNE (1990). C.P.M. scheduling with small communication delays and task duplication. *Oper. Res.*, to appear.

J. DU, J. Y-T. LEUNG (1989). Complexity of scheduling parallel task systems. *SIAM J. Discrete Math. 2*, 473-487.

M.R. GAREY, D.S. JOHNSON (1979). *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

R.L. GRAHAM (1966). Bounds for certain multiprocessing anomalies. *Bell System Tech. J. 45*, 1563-1581.

R.L. GRAHAM, E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math. 5*, 287-326.

C.P.M. VAN HOESEL (1990). Preemptive scheduling on a hypercube. Unpublished manuscript.

J.J. HWANG, Y.C. CHOW, F.D. ANGER, C.Y. LEE (1989). Scheduling precedence graphs in systems with interprocessor communication times. *SIAM J. Comput. 18*, 244-257.

S.J. KIM (1988). *A General Approach to Multiprocessor Scheduling*, Dissertation TR-88-04, University of Texas at Austin.

E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, D.B. SHMOYS (1989). Sequencing and scheduling: algorithms and complexity. To appear in the *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory,* edited by S.C. Graves, A.H.G. Rinnooy Kan and P. Zipkin, North-Holland, Amsterdam.

C.Y. LEE, J.J. HWANG, Y.C. CHOW, F.D. ANGER (1988). Multiprocessor scheduling with interprocessor communication delays. *Discrete Appl. Math. 20,* 141-147.

R. MCNAUGHTON (1959). Scheduling with deadlines and loss functions. *Management Sci. 6,* 1-12.

C.H. PAPADIMITRIOU, M. YANNAKAKIS (1988). Towards an architecture-independent analysis of parallel algorithms. *Proc. 20th Annual ACM Symp. Theory of Computing*, 510-513.

V.J. RAYWARD-SMITH (1987A). The complexity of preemptive scheduling given interprocessor communication delays. *Inform. Process. Lett. 25*, 123-125.

V.J. RAYWARD-SMITH (1987B). UET scheduling with unit interprocessor communication delays. *Discrete Appl. Math. 18*, 55-71.

V. SARKAR (1989). *Partitioning and Scheduling Parallel Programs for Multiprocessors*, Pitman, London.