J Sched (2008) 11: 347–356
DOI 10.1007/s10951-007-0051-7

# Scheduling multiple divisible loads in homogeneous star systems

**M. Drozdowski · M. Lawenda**

Received: 19 October 2006 / Accepted: 16 November 2007 / Published online: 9 January 2008


**Abstract** In this paper we analyze scheduling multiple divisible loads on a star-connected system of identical processors. It is shown that this problem is computationally hard. Some special cases appear to be particularly difficult, so it is not even known if they belong to the class **NP**. Exponential algorithms and special cases solvable in polynomial time are presented.

**Keywords** Parallel processing · Scheduling · Divisible loads · Computational complexity

## 1 Introduction

Classic scheduling models assume that a parallel application is composed of a set of sequential communicating processes. Thus, it can be represented as a graph of sequential tasks connected by arcs representing communications. Yet, other types of parallel applications which consist in processing great volumes of data, generally called *load*, also exist. For this kind of parallel program it is more natural to assume that load can be partitioned and distributed for remote processing. Divisible load theory (DLT) assumes that a parallel application is arbitrarily divisible and executable in parallel. Applications conforming with this model can be found in processing databases, measurements and video. Hence, DLT applies in many practical cases. DLT turned out to be a flexible vehicle in modeling systems with various communication algorithms, interconnection topologies, memory hierarchy, time and cost trade offs. Surveys of DLT can be found, for example, in (Bharadwaj et al. 1996; Drozdowski 1997; Robertazzi 2003).

In this paper we consider a set of $m$ processors $P_1, \ldots, P_m$ connected in a star topology. There is a processor $P_0$ called originator in the center of the star. The purpose of the originator is to schedule the computations and perform communications. $P_0$ does no computing. If $P_0$ were able to process the load, its computing capability could be represented as a processor among $P_1, \ldots, P_m$. The communications link only the originator $P_0$ with the remaining processors. There is no direct communication between processors $P_1, \ldots, P_m$. Only one transmission can be performed at a given time. Star interconnection is a general model for various real environments. This can be a set of CPUs connected by a bus in a multiprocessor system, a cluster of workstations connected by a single Ethernet segment, or a set of grid computers connected over the Internet. In a heterogeneous star, processor $P_i$, $i = 1, \ldots, m$, is defined by processing rate $A_i$ (reciprocal of computing speed), communication rate $C_i$ (reciprocal of bandwidth), and communication startup time $S_i$. The times required to send $x$ units of load to $P_i$, and process it are $S_i + C_i x$ and $A_i x$, respectively. Processors are able to compute and communicate simultaneously. For the sake of simplicity we assume that the time of returning results can be neglected. This assumption does not restrict generality of the considerations because the result collection has been successfully incorporated in the DLT model (see, e.g., Bharadwaj et al. 1996;

M. Drozdowski's research partially supported by Polish Ministry of Science and Higher Education.

M. Drozdowski (✉)
Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland
e-mail: Maciej.Drozdowski@cs.put.poznan.pl

M. Lawenda
Poznań Supercomputing and Networking Center, Noskowskiego 10, 61-704 Poznań, Poland

Drozdowski 1997). Initially the originator holds $n$ divisible loads $T_1, \ldots, T_n$ of sizes $V_1, \ldots, V_n$, respectively. We will be using names 'tasks' and 'loads' interchangeably. It is the responsibility of the originator to schedule tasks so that schedule length, denoted $C_{\max}$, is minimum. In particular, it is necessary to:

- select the number of transmissions $g$
- assign tasks to transmissions
- assign processors to transmissions
- select size $\alpha_{ij}$ of task $T_j$ load sent in transmissions $i = 1, \ldots, g$

It is required that $\sum_{i=1}^{g} \alpha_{ij} = V_j$, for each task $T_j$. $\alpha_{ij} = 0$ means that $T_j$ is not using transmission $i$.

Scheduling multiple divisible loads has been considered in (Bharadwaj et al. 1996; Sohn and Robertazzi 1994). It was assumed that tasks were executed in the first-in first-out sequence on a set of heterogeneous processors, all processors were used by each task, computations of a task finished on all processors simultaneously, and startup time was negligible ($\forall_i\ S_i = 0$). It was observed that transmissions of some task $T_j$ could overlap with computations of the preceding task $T_{j-1}$. This allows to start computations of task $T_j$ on some processors $P_1, \ldots, P_{m'}$ immediately after the end of task $T_{j-1}$. Processors $P_{m'+1}, \ldots, P_m$ remain idle until they receive their share of $T_j$ load. For a given $m'$ the distribution of the load can be found from a set of linear equations in $O(m)$ time. Here $m'$ can be found iteratively in at most $m$ steps. The complexity of the algorithm is $O(m^2 n)$.

In (Veeravalli and Barlas 2002) the same set of assumptions was made. A *multi-installment* load distribution strategy has been proposed. In multi-installment (a.k.a. multiround) distribution the load is sent to the processors in many small chunks instead of one long message per processor. When the messages are long, the processors also have to wait a long time for the load and the start of the computations. Hence, multi-installment distribution of the load is advantageous to minimize schedule length. When the overlap of computations on $T_{j-1}$ with the transmissions of $T_j$ is too short to send the whole load $V_j$ to the processors and to avoid idle time (i.e., if $m' < m$), then the load is divided into multiple smaller installments. Since messages are shorter, all processors may receive some load earlier and may work continuously on $T_j$. However, not in all instances can the idle time be avoided (Veeravalli and Barlas 2002).

The problem of scheduling multiple loads in heterogeneous systems has been generalized in (Drozdowski et al. 2006). It was observed that the performance of the processors may be perceived by tasks in different ways depending on the type of processing environment. Hence, three types of processing environments were distinguished: *Unrelated processors* where the computation and communication rates, as well as the startup time depend both on the processor and the task. For example, for task $T_j$ processed on processor $P_i$ these parameters could be denoted: $A_{ij}$, $C_{ij}$, $S_{ij}$, respectively. *Uniform processors* which differ between themselves, but all tasks perceive these differences identically (this case was defined in the preceding paragraphs). *Identical processors* which are the same for all the tasks. Consequently, $A_i = A$, $S_i = S$, $C_i = C$, for $i = 1, \ldots, m$. Only *permutation schedules* were considered in (Drozdowski et al. 2006). The permutation schedule was defined as follows: a task is sent to the processors only once, a processor executes the task only once, on all processors the sequence of executing the tasks is the same as the sequence of distributing the tasks. Consequently, transmissions and computations of some task cannot be suspended and restarted later. It was shown that scheduling multiple loads is **NP**-hard for $m = 1$ unrelated processor if result returning has to be explicitly scheduled, for $m = 2$ unrelated processors if result returning is not considered, for $n = 2$ tasks and uniform processors. However, if the sequence of the communications is known then the optimum distribution of the load ($\alpha_{ij}$s) can be calculated in polynomial time using linear programming. Also some other polynomially solvable cases and approximability bounds were given in (Drozdowski et al. 2006).

Note that in (Sohn and Robertazzi 1994; Veeravalli and Barlas 2002; Drozdowski et al. 2006) the load distribution algorithm is the key element of the solution. The hard combinatorial nature of divisible load scheduling problems often follows from load scattering over communication network with certain properties. For example, England et al. (2007) analyzes efficient spanning tree topologies for load distribution. Considering the flexibility in designing communication algorithms, there seems to be no general rule on which one could base a proof that a certain load scattering method leads to the optimum schedules. Here we attempt to delineate the border between computationally hard and easy cases. To avoid these difficulties a rudimentary network topology and a transmission time model are assumed.

In this paper we analyze identical processors with startup time dominating in the transmission time. Startup time is the main part of transmission time, especially if messages are short. Consequently, we assume $A_i = A$, $S_i = S$, $C_i = 0$, for $i = 1, \ldots, m$. We hope to show that despite the simplicity of formulation it is still a challenging problem.

The rest of the paper is organized as follows. In the next section we show that scheduling different tasks is **NP**-hard even on $m = 2$ processors. In Sect. 3 we analyze properties of the schedules for a big number of identical tasks, to propose polynomial time solutions in Sect. 4. The case of a small number of tasks is studied in Sect. 5. The results are summed up in the last section.

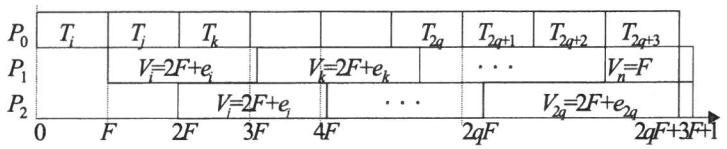


**Fig. 1** Illustration to the proof of Theorem 1

## 2 Different tasks

### 2.1 Complexity of the problem

In this section we prove that scheduling multiple divisible loads is **NP**-hard even for two identical processors if tasks are different. In the proof of **NP**-hardness we will use the **NP**-complete problem PARTITION WITH EQUAL CARDINALITY defined as follows (Garey and Johnson 1979):

PARTITION WITH EQUAL CARDINALITY
INSTANCE: A finite set $E = \{e_1, \ldots, e_{2q}\}$ of positive integers.
QUESTION: Is there a subset $E' \subset E$ such that $\sum_{j \in E'} e_j = \sum_{j \in E - E'} e_j = \frac{1}{2} \sum_{j=1}^{2q} e_j = F$, and $|E'| = |E - E'| = q$?

**Theorem 1** *Scheduling of multiple divisible loads is* **NP***-hard even for two identical processors.*

*Proof* We will show that our scheduling problem is **NP**-hard by presenting a Turing reduction from PARTITION WITH EQUAL CARDINALITY to a decision version of our problem. Without loss of generality we assume that $e_i$ are multiples of 3, for $i = 1, \ldots, 2q$. Were it otherwise, all $e_i$ could be multiplied by 3 to satisfy this requirement without changing the instance answer. The transformation is as follows:

$$m = 2, \qquad S = F, \qquad C = 0,$$

$$A = 1, \qquad n = 2q + 3,$$

$$V_j = 2F + e_j \quad \text{for } j = 1, \ldots, 2q,$$

$$V_{2q+1} = F, V_{2q+2} = 1, V_{2q+3} = 1.$$

We ask if a schedule with length at most $y = (2q + 3)F + 1$ exists. Suppose, that the PARTITION WITH EQUAL CARDINALITY instance has a positive answer. Then a feasible schedule of length $(2q + 3)F + 1$ can be constructed as shown in Fig. 1: Tasks corresponding to $E'$ and $T_{2q+1}, T_{2q+3}$ are sent to $P_1$, and the remaining tasks are sent to $P_2$. Transmissions of $P_2$ interleave the transmissions of $P_1$.

Suppose that there is a schedule of length at most $y$. Without loss of generality, let $P_1$ be the first processor which started computations. Note that only $2q + 3$ transmissions of length $S = F$ can be initiated and completed. Also $n = 2q + 3$. Thus, the load for each task is sent in only one message, and each task can be executed on one processor only. The time required for processing the tasks $\sum_{j=1}^{n} V_j A = 4qF + 3F + 2$ is equal to the length of available processing intervals on processors $P_1$ and $P_2$. Hence, there is no idle time in the computations on the processors. Tasks $T_{2q+1}, T_{2q+2}, T_{2q+3}$ have to be sent from the originator at the end of the schedule, because no other tasks with transmissions initiated at $2qF, (2q + 1)F, (2q + 2)F$ can be finished by $y = (2q + 3)F + 1$. $T_{2q+2}, T_{2q+3}$ must be executed on different processors because processing times of other tasks are multiples of 3 and otherwise there would be an idle time on some processor. This leaves $(2q + 2)F$ free time on $P_1$ and $(2q + 1)F$ time on $P_2$. Let $\mathcal{T}_i$ denote the set of tasks executed by processor $P_i$, for $i = 1, 2$. If $T_{2q+1}$ were processed by $P_2$, then at most $q - 1$ tasks from $\{T_1, \ldots, T_{2q}\}$ would be processed on $P_2$ because $V_j A > 2F$, for $j = 1, \ldots, 2q$. Processing the tasks on $P_2$ would take $2F(q - 1) + \sum_{T_j \in \mathcal{T}_2} e_j + F < (2q + 1)F$, and an idle time would arise on $P_2$. Thus, $T_{2q}$ must be executed on $P_1$, exactly $q$ tasks from $\{T_1, \ldots, T_{2q}\}$ have to be on $P_2$ to avoid the idle time. Consequently, $q$ tasks from $\{T_1, \ldots, T_{2q}\}$ are executed also on $P_1$. Since $T_{2q+1}$ is executed on $P_1$, the processing requirements of the tasks on $P_1$, and on $P_2$ must be equal to the remaining available time, i.e., $\sum_{T_j \in \mathcal{T}_1 - \{T_{2q+1}\}} AV_j = 2qF + \sum_{T_j \in \mathcal{T}_1 - \{T_{2q+1}\}} e_j = \sum_{T_j \in \mathcal{T}_2} AV_j = 2qF + \sum_{T_j \in \mathcal{T}_2} e_j = 2qF + F$. Hence, $\sum_{T_j \in \mathcal{T}_1 - \{T_{2q+1}\}} e_j = \sum_{T_j \in \mathcal{T}_2} e_j = F$, and the answer to PARTITION WITH EQUAL CARDINALITY is affirmative. □

In the proof of Theorem 1 tasks use one transmission. Thus, it is an example of the 1-round load distribution. The restriction to a single communication is not presumed, but it follows from the features of the instance. Therefore, Theorem 1 applies both to 1- and multi-round distributions. For similar reasons it applies to permutation and non-permutation schedules, as well as to the schedules with or without simultaneous completion of the computations. Note that we did not prove that our problem is in the class **NP**. Thus, it is **NP**-hard, but it is not known if it is **NP**-complete.

### 2.2 Minimum schedule length for a given communication pattern

We will show that for a given communication pattern the existence of a schedule of some given length $D$ can be verified in polynomial time in the length of the string encoding the

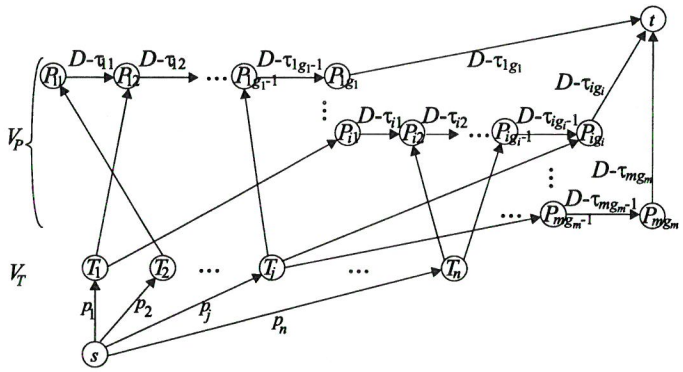


**Fig. 2** Network for a given pattern

instance and the communication sequence, by finding maximum flow in a network.

Since there are no memory limitations, all transmissions may be shifted to the left (i.e., executed as early as possible, so that there is no idle time between them) without changing the schedule length. Thus, for a given communication pattern we know when a particular communication finishes.

The construction of the network is shown in Fig. 2. Beyond source $s$ and sink $t$ there are $n$ nodes in set $V_T$ representing tasks and $g$ nodes in set $V_P$ which represent positions of task transmissions to the processors. Let $g_i$ denote the number of tasks executed on processor $P_i$. For each transmission $l$ to processor $P_i$, $l = 1, \ldots, g_i$, a node denoted $P_{il}$, is created in the set $V_P$ (cf. Fig. 2). The transmissions are counted from the last message sent to $P_i$ (for which $l = 1$) to the first one (for which $l = g_i$). There are arcs $(s, v_j)$ of capacity $p_j$ for each node (task) $v_j \in V_T$. For a node $v_j \in V_T$ and $P_{il} \in V_P$ an arc $(v_j, P_{il})$ is created if task $T_j$ is sent to processor $P_i$ as the $l$th message counted from the last transmission to $P_i$. The capacity of arcs $(v_j, P_{il})$ is not bounded. For a pair of transmissions $l, l+1$ to processor $P_i$, an arc $(P_{il}, P_{i(l+1)})$ is created with the capacity $D - \tau_{il}$, where $\tau_{il}$ is the time moment when transmission $l$ to processor $P_i$ is finished. An arc $(P_{ig_i}, t)$ with capacity $D - \tau_{ig_i}$ is created for each processor $P_i$.

A schedule of length $D$ exists for the given communication pattern, if the maximum flow saturates arcs $(s, v_j)$ for each $v_j \in V_T$. A schedule for processor $P_i$ can be constructed analogously to the construction of a schedule for problem $1|r_j, pmtn|C_{\max}$. Here, flows $\phi(v_j, P_{il})$ are lengths of the pieces of tasks $T_j$; ready times are the communication completion times $\tau_{il}$ dictated by the communication pattern. For feasibility of a schedule on $P_i$ it is necessary to fulfill condition $\tau_{il} + \sum_{h=1}^{l} \phi(v_j, P_{ih}) \leq D$. This inequality is satisfied by the construction of a chain of nodes $P_{i1} \to P_{i2} \to \cdots \to P_{ig_i} \to t$, and the capacities of the subsequent arcs (see Fig. 3). Note that a flow $\sum_{h=1}^{l} \phi(v_j, P_{ih})$ is passed over arc $(P_{il}, P_{il+1})$ with capacity $D - \tau_{il}$ for $l = 1, \ldots, g_i - 1$.

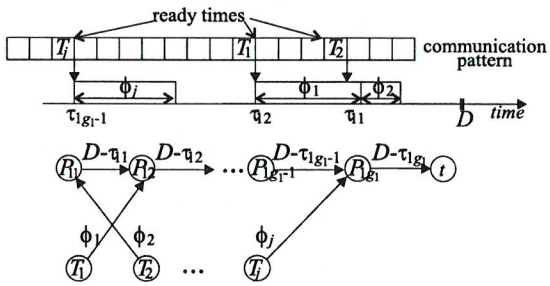


**Fig. 3** Processor load bounds imposed by the network

**Theorem 2** *There is an algorithm which constructs an optimum schedule in $O(m^3 n^3 \log m)$ time for a given communication pattern.*

*Proof* The number of nodes in the network is $2 + n + \sum_{i=1}^{m} g_i$, which is $O(mn)$, because no task has to be delivered to a processor twice and hence $g_i \leq n$. Thus, for a given communication pattern and value of $D$ the maximum network flow can be constructed in $O(m^3 n^3)$.

The above algorithm can be used to find the optimum schedule length for a given communication pattern. The optimum schedule length is the minimum $D \geq g$ for which all arcs $(s, v_j)$ are saturated, and consequently the value of the flow is $\phi = \sum_{j=1}^{n} p_j$. If a schedule with length $D = g$ does not exist because $\phi < \sum_{j=1}^{n} p_j$, then its length must be increased by some value $\Delta$. The value of the flow $\phi$ is determined by the capacity of the minimum cut. When $D$ grows by $\Delta$, then the capacity of the cut may grow by $\mu\Delta$ where $\mu$ is a multiplier from set $\{1, \ldots, m\}$. The multiplier $\mu$ is determined by the number of arcs on the minimum cut whose capacities grow as $D$ is growing. It can be observed (cf. Fig. 2) that the minimum number of edges on the minimum cut with capacity growing with $D$ is one, and the maximum is $m$. Suppose $\phi \leq \sum_{j=1}^{n} p_j$ is the value of the flow obtained for $D = g$. The length of the schedule must be increased by $\Delta = (\sum_{j=1}^{n} p_j - \phi)/\mu$. Though the actual multiplier is not known initially, it can be determined by a binary search over the interval of (discrete) values $\{1, \ldots, m\}$. Thus, the mini-

mum schedule length for a given communication pattern can be found in $O(m^3n^3 \log m)$ time. □

Note that this algorithm is pseudopolynomial in $m$. However, it is polynomial in the length of the communication pattern because $m$ is the number of used processors.

**Observation 3** *There is an algorithm which constructs an optimum schedule in* $O(m^{nm}m^3n^3 \log m)$ *time.*

*Proof* The length of the communication pattern is at most $nm$. Hence, the number of patterns is at most $m^{nm}$. For each of them the optimum schedule length may be calculated in $O(m^3n^3 \log m)$ time. □

### 2.3 A remark on approximability

Since this problem is **NP**-hard, it is reasonable to propose approximation algorithms. Let $C^*_{\max}$ denote the length of the optimum schedule, and $C^H_{\max}$ the length of a schedule built by heuristic $H$. The worst case of the ratio $S_H = \frac{C^H_{\max}}{C_{\max *}}$ is typically used as a measure of an approximation algorithm performance. Following (Drozdowski et al. 2006) it can be concluded that any greedy algorithm $H$ for our problem has the worst case performance ratio $S_H \leq m + 1$. If, furthermore, $\forall j$, $Sm \leq \frac{V_j A}{m}$, which means that the computation time dominates transmission time, then a $CC$ heuristic exists which has the worst case performance ratio $S_{CC} \leq 2$. $CC$ divides the load of each task into $m$ equal parts, then sends the load to the processors in arbitrary order, and finally executes the load chunks in parallel on all $m$ processors. Furthermore, if $\forall j$, $\frac{V_j A}{m^2 S} \to \infty$ then $S_{CC} \to 1$. In other words, if the tasks tend to be computationally dominated, $CC$ becomes asymptotically optimum, because transmission time is negligible in the schedule length.

## 3 Properties of schedules for identical tasks

It follows from Theorem 1 and the results in (Drozdowski et al. 2006) that the only problem with non-zero startup time which may admit polynomial solvability is scheduling of identical tasks on identical processors. We assume that all tasks have load $V$, for $j = 1, \ldots, n$. We will use startup time $S$ as time unit and will denote by $k$ the computation time of a task on a single processor, i.e., $k = \frac{VA}{S}$. Observe that the parameters of an instance in this special case are $k, m, n$, and a polynomial time algorithm must have a running time polynomial in $\log k, \log m, \log n$. On the other hand, sheer assignment of the tasks to the processors requires pseudopolynomial time $\Omega(n)$. Thus, a polynomial time algorithm may not consist in assigning the tasks to the processors on the task-by-task basis. Intuitively, it may be expected that a polynomial algorithm can only identify patterns of the optimum schedule. Unfortunately, the results of Sect. 5.1 show no apparent regularity of optimum communication sequences, and such pattern solutions may be hard to find in general.

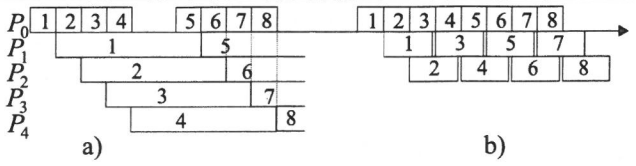


**Fig. 4** Bounding (**a**), and not bounding (**b**) schedules

Let us start with some observations on possible patterns of the optimum schedules.

**Observation 4** *The earliest time moments when the transmissions may be finished, and the computations started are integer time instants* $(1, 2, 3, \ldots)$.

A pattern of events taking place on all processors consecutively in unit time distances will be called a *staircase* pattern. Thus, Observation 4 mentions the staircase pattern of computation start times. Note that within $k$ time units of executing some task $T_j$ on a single processor, the load of $\lceil k \rceil$ other tasks can be sent to the processors. Out of this load $\lceil k \rceil - 1$ tasks can be sent to other processors than $T_j$, but the $\lceil k \rceil$th task after $T_j$ may be computed on the same processor as $T_j$, because at the time of finishing the load transfer for the $\lceil k \rceil$th task after $T_j$ the processor computing task $T_j$ will be free (cf. Fig. 4). We will say that the number of processors $m$ is *not bounding* if the number of processors which can be effectively used follows from the number of transmissions which may be performed in the interval of a single task computation, i.e., when $m \geq \lceil k \rceil$. In the opposite case we will say that the number of processors is *bounding*.

**Observation 5** *When the number of processors is bounding, and processors are computing all the time since the earliest possible activation time to the simultaneous completion of the computations, then schedule is optimum.*

*Proof* Such a schedule cannot be made shorter because there is no idle time on the processors which could be eliminated. □

**Observation 6** *When the processor number is not bounding, and the originator is sending the load all the time with the exception of an interval not longer than* 1 *at the end of the schedule, and the processors computing in this last interval finish the computations simultaneously, then the schedule is optimum.*

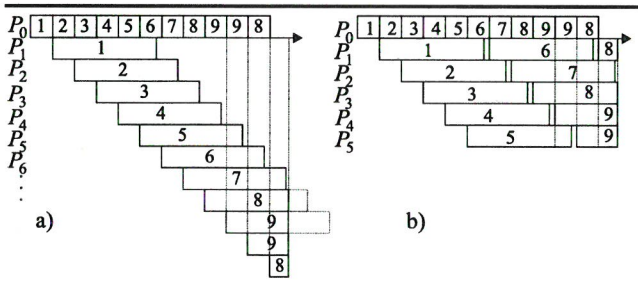


**Fig. 5** Observation 6. Folding a schedule from **a** $m = \infty$, to **b** $m = \lceil k \rceil$

*Proof* Let us assume, for the time being, that the number of processors is infinite ($m \geq n + \lceil \frac{k}{2} \rceil$ is sufficient). If the originator sends the load to a new processor in each time unit (see Fig. 5a), then the tasks (except for the last $\lfloor \frac{k}{2} \rfloor$ trailing tasks) cannot be completed earlier because the originator is communicating all the time, and no additional processor can be activated for a given task without delaying another task. If at the end of the schedule the originator is idle at most 1 time unit, then no more processors can be activated and effectively exploited. If the computation on the trailing tasks activates processors at the earliest possible moments, processors work without idle time, and finish computations simultaneously, then the trailing tasks cannot be completed earlier. Observe that this reasoning does not require an infinite number of processors. The above schedule can be folded, without changing the length, to $\lceil k \rceil$ processors (see Fig. 5b). □

## 4 Big number of tasks: $n > \min\{\lceil k \rceil, m\}$

In this section we propose a method of identifying optimum schedules in some cases if the number of tasks is sufficiently big.

### 4.1 Processor number is not bounding

**Theorem 7** *If $n > \min\{\lceil k \rceil, m\}$, $m \geq \lceil k \rceil$, then the optimum schedule for identical tasks on a star of identical processors with $C = 0$ can be determined in polynomial time.*

*Proof* Since $m \geq \lceil k \rceil$, the processor number is not bounding, and the number of usable processors $\lceil k \rceil$ is determined by the number of transmissions which can be done during the computation. The optimum schedules have two parts: a leading part which we will call a *main sequence*, and a trailing part which will be called a *tail*. In the main sequence the originator sends tasks to consecutive processors in a round-robin fashion. It means that after sending load of task $T_j$ to processor $P_i$ in the interval $[l, l+1]$, task $T_{j+1}$ is sent to processor $P_{(i \bmod \lceil k \rceil)+1}$ in the interval $[l+1, l+2]$, where $l$ is integer. Task $T_j$ is processed in the interval $[l+1, l+k+1]$, and task $T_{j+1}$ in the interval $[l+2, l+k+2]$. In the tail transmissions and computations are arranged so that all processors stop computing simultaneously. The schedules proposed in this case are optimum by Observation 6. W.l.o.g. let $P_{\lceil k \rceil}$ be the processor executing the last task from the main sequence. If it is not the case, then the processors may be renumbered (precisely, their indexes must be rotated) to meet this requirement. Let $a$ denote the number of tasks in the tail, and $t$ the time moment when the communication of the last task in the main sequence is finished. Below we analyze possible cases.

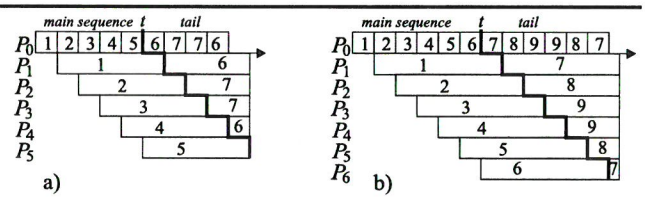


**Fig. 6** Proof of Theorem 7. Processor number is not bounding

1. *$k$ is integer.*

1.1. *$k$ is odd* (Fig. 6a). In this case the tail comprises $a = (k-1)/2$ trailing tasks, $C_{\max} = t + k$. Task $n - a + i$, for $i = 1, \ldots, a$, is sent to processors $P_i$, $P_{k-i}$, in intervals $[t - 1 + i, t + i]$, $[t - 1 + k - i, t + k - i]$, respectively. It is processed on $P_i$, $P_{k-i}$ in intervals $[t + i, t + k]$, $[t + k - i, t + k]$, respectively.

1.2. *$k$ is even* (Fig. 6b). In this case $a = k/2$, $C_{\max} = t + k + \frac{1}{2}$. Task $n - a + i$, for $i = 1, \ldots, a$, is sent to processors $P_i$, $P_{k-i+1}$ in intervals $[t - 1 + i, t + i]$, $[t + k - i, t + k - i + 1]$, respectively. It is processed on $P_i$, $P_{k-i}$ in intervals $[t + i, t + k + \frac{1}{2}]$, $[t + k - i + 1, t + k + \frac{1}{2}]$, respectively.

2. *$k$ is fractional.* A schedule can be constructed as in Case 1 for tasks with computational demand $\lceil k \rceil$. Since $m \geq \lceil k \rceil$, the schedule length is determined by the processing power which can be engaged during the schedule. Though there may be $(\lceil k \rceil - k)$-long idle times on the processors, the maximum processing power is used because the originator starts computations on an inactive processor at the end of every time unit. A schedule constructed as in Case 1 for $\lceil k \rceil$ is optimal by Observation 6.

Identifying a proper case for the construction of a tail requires checking fractionality, parity of $k$, comparing $k$ with $m$ and $n$. All this can be done in polynomial time $O(\max\{\log n, \log m, \log k\})$. □

### 4.2 Processor number is bounding

In the above Theorem 7 we analyzed the non-bounding processor number. In this case, if $n - a$ were not a multiple of $\lceil k \rceil$, it was possible to rotate processor indices so that both start and finish of the computations in the main sequence form a staircase pattern on all usable processors without creating an idle time on the originator. Unfortunately, this idea

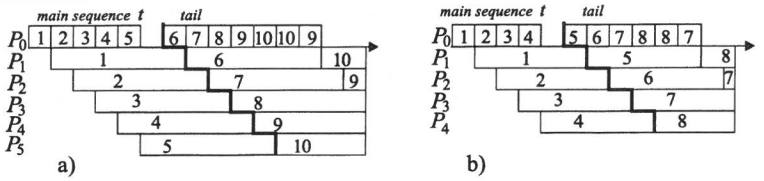


**Fig. 7** Proof of Theorem 8. **a** $m$ is odd, **b** $m$ is even

cannot be applied when the processor number is bounding. If $(n-a) \bmod m \neq 0$, then a staircase pattern at the end of the main sequence computations results in the lack of a staircase pattern at the beginning of the computations and in processor idle times. And vice versa, a staircase pattern of computation start times results in the lack of a staircase pattern at the end of the main sequence. Therefore, if the processor number is bounding, the technique used in Theorem 7 can be applied only for task numbers for which staircase patterns appear without processor rotation. Below we present special cases of the bounding processor number ($m < k$) for which the optimum schedules can be identified in polynomial time.

**Theorem 8** *If $n > \min\{\lceil k \rceil, m\}, m < \lceil k \rceil, n \bmod m = 0$, then the optimum schedule for identical tasks on a star of identical processors with $C = 0$ can be determined in polynomial time.*

*Proof* The schedules consist of the main sequence which starts and ends in a staircase pattern. The number of tasks in the main sequence is a multiple of $m$. The tails are constructed so that computations are finished simultaneously on all processors. The schedules proposed are optimum by Observation 5. Since $m < \lceil k \rceil$, idle time may arise on the originator. Therefore, in the following discussion we assume that the communications of the main sequence are shifted to the right, i.e., are performed as late as possible. Below we analyze possible cases.

1. *$k$ is integer.* In this case $a = m$, $C_{\max} = t + 2k - \frac{m-1}{2}$. The number of tail transmissions is $k + \frac{m}{2}$, which can be done feasibly before $C_{\max}$.

1.1. *$m$ is odd* (Fig. 7a). The first $\frac{m+1}{2}$ tasks of the tail continue the schedule of the main sequence finishing at $C_{\max}$ on processor $P_{(m+1)/2}$. The remaining $\frac{m-1}{2}$ tasks are split into two parts and executed on processors placed symmetrically to $P_{(m+1)/2}$. Precisely, task $n - i + 1$ for $i = 1, \ldots, (m-1)/2$, is executed in intervals $[t + 2k - m + i, C_{\max}]$, and $[t + k - i + 1, C_{\max}]$ on processors $P_i$, $P_{m-i+1}$, respectively.

1.2. *$m$ is even* (Fig. 7b). The tail construction is similar. The first $\frac{m}{2}$ tasks continue the main sequence finishing at $C_{\max} - \frac{1}{2}$ on processor $P_{m/2}$. The remaining $\frac{m}{2}$ tasks are split into two parts. Task $n - i + 1$ for $i = 1, \ldots, \frac{m}{2}$, is executed in intervals $[t + 2k - m + i, C_{\max}]$, and $[t + k - i + 1, C_{\max}]$ on processors $P_i$, $P_{m-i+1}$, respectively.

2. *$k$ is fractional.* The main sequence starts computations on the processors at the earliest time moments, and processors have no idle times because $k > m$. Since the computations of the main sequence finish in unit time distances, it is possible to apply the tail constructions from Case 1, so that processors stop computing simultaneously. □

Let us observe that the tail construction method from Theorem 7 Case 1.2 can be applied when the processor number is bounding, $m$ is even, and $n \bmod m = \frac{m}{2}$. In such a case $a = m/2$, $C_{\max} = t + 2k - \frac{m-1}{2}$. In general, constructing an optimum tail to the main sequence is an open problem which seems equivalent to scheduling a small number of tasks with a bounding processor number. We analyze it in Sect. 5.2.

## 5 Small number of tasks: $n \leq \min\{\lceil k \rceil, m\}$

The case with too few tasks to have both the main sequence and a tail is more involved. We start the analysis with the case of a non-bounding processor number.

### 5.1 Non-bounding processor number

Since the processor number is not bounding, there is no advantage in sending load to a processor more than once. Hence, with each time unit of the schedule, computation on one new processor can be activated. In time $y > 1$, $\lfloor y \rfloor$ processors are able to compute for time $y - 1, \ldots, y - \lfloor y \rfloor$. The total processing capacity in $y > 1$ units of time is $\sum_{i=1}^{\lfloor y \rfloor} (y - i)$. For example, in a schedule of length 7, different processors will work for 1, 2, 3, 4, 5, 6 time units, performing 21 units of work. Our problem of scheduling $n$ tasks with computational demand $k$ can be represented as finding decomposition of numbers $y - 1, \ldots, y - \lfloor y \rfloor$ into $n$ sums equal at least $k$. From this formulation we can draw a conclusion that an ideal solution in which all tasks finish computations simultaneously does not always exist. There is an analogy to a scheduling problem $P \| C_{\max}$ which consists in decomposing of a set of task processing times into subsets assigned to the processors so that none of the subsets requires more than some given schedule length. Problem $P \| C_{\max}$ can be solved by a pseudopolynomial algorithm for a fixed number of processors (Rothkopf 1966). Let us treat the $n$ tasks of the original problem as processors, and processing capacities $y - 1, \ldots, y - \lfloor y \rfloor$ of the original processors as tasks. This transforms our original scheduling problem into problem $P \| C_{\max}$ with $n$ processors, tasks of

**Table 1**

| Solution A | | | Solution B | | |
|---|---|---|---|---|---|
| Task | Intervals | Sum | Task | Intervals | Sum |
| 1: | $7+x, 2+x, 1+x$ | $10+3x$ | 1: | $7+x, 3+x$ | $10+2x$ |
| 2: | $6+x, 3+x, x$ | $9+3x$ | 2: | $6+x, 4+x$ | $10+2x$ |
| 3: | $5+x, 4+x,$ | $9+2x$ | 3: | $5+x, 2+x, 1+x, x$ | $8+4x$ |

length $y-1,\ldots,y-\lfloor y\rfloor$, and a demand that each processor works for at least $k$ units of time. Yet, such a transformation to a version of problem $P\|C_{\max}$ is not sufficient, because the completion time of a task (in the original problem) depends on the number of processors executing it. We demonstrate it in the following example.

*Example* $n=3, k=11$. Since $3V=33$, the schedule length must be at least 8. Assume that $C_{\max}=8+x$ and $0<x<1$. With a schedule of this length, processors will work for $x, 1+x, 2+x, \ldots, 7+x$ units of time. Two possible assignments of these computing intervals to tasks are shown in Table 1 (intervals are identified by their lengths).

Solution A results in $C_{\max}=9$, and solution B has $C_{\max}=8.75$, because each task requires $k=11$ units of computation time. Thus, an algorithm solving our problem should take into consideration not only the sum of the lengths of intervals received by a task, but also their number.

Our problem can be solved by a dynamic programming algorithm. Suppose that the optimum schedule length is $D+x$, where $D$ is an integer lower bound which can be calculated, e.g., from requirement $\sum_{i=1}^{D}(i-1)\le nk\le\sum_{i=1}^{D+1}(i-1)$, and $0\le x<1$ is a necessary schedule extension beyond $D$ (cf. Fig. 8). Our method finds the smallest $x$ for which a feasible schedule exists, or determines that no schedule with $x<1$ exists. Let the intervals on the processors be ordered according to their decreasing length, i.e., $D+x-1, D+x-2, \ldots, 1+x, x$. The method is based on the calculation of the function

$$f(a_1,b_1,\ldots,a_j,b_j,\ldots,a_n,b_n,i)$$

$$=\begin{cases}1, & \text{if a schedule exists which is using the first } i \text{ intervals, task } T_j \text{ receives } a_j \text{ units of processing in the interval } [1,D] \text{ and uses } b_j \text{ processors in the interval } [D,D+x],\\ 0, & \text{otherwise,}\end{cases}$$

for $a_j=0,\ldots,k$, $b_j=0,\ldots,z$, $j=1,\ldots,n$, $i=0,\ldots,D$, where $z$ is the maximum number of processors needed to execute a task with computational demand $k$. Here $z$ can be calculated from the condition $k\le\sum_{i=1}^{z}(i-1)$. Function $f$ may be calculated using the following recursive equations:

$$f(a_1,b_1,\ldots,a_n,b_n,0)$$

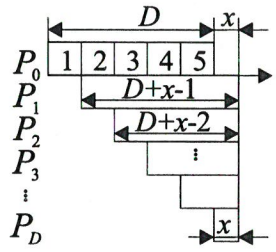


**Fig. 8** Structure of a schedule for a small number of tasks and a lot of processors

$$=\begin{cases}1, & \text{if } a_1=b_1=\cdots=a_n=b_n=0,\\ 0, & \text{otherwise,}\end{cases}$$

$$f(a_1,b_1,\ldots,a_j,b_j,\ldots,i+1)$$

$$=f(a_1,b_1,\ldots,a_j-D-i-1,b_j-1,\ldots,i)$$

for $a_j=0,\ldots,k$, $b_j=0,\ldots,z$, $j=1,\ldots,n$, $i=0,\ldots,D-1$. Given values of vectors $(a_1,b_1,\ldots,a_n,b_n,D)$ for which $f(a_1,b_1,\ldots,a_n,b_n,D)=1$, it is possible to calculate schedule length $C_{\max}(a_1,b_1,\ldots,a_n,b_n,D)=D+\max_{j=1}^{n}\{\frac{k-a_j}{b_j}\}$, where $x=\max_{j=1}^{n}\{\frac{k-a_j}{b_j}\}$ is the extension of the schedule beyond $D$. The optimum schedule extension $x$ can be found for such vector $(a_1',b_1',\ldots,a_n',b_n',D)$ for which $\max_{j=1}^{n}\{\frac{k-a_j}{b_j}\}$ is minimum, i.e.,

$$(a_1',b_1',\ldots,D)$$

$$=\operatorname*{argmin}_{(a_1,b_1,\ldots,D)}\left\{\max_{j=1}^{n}\left\{\frac{k-a_j}{b_j}\right\}\;\middle|\; f(a_1,b_1,\ldots,D)=1\right\}.$$

If $x\ge 1$ then the initial value of $D$ was assumed too small. In the opposite case the optimum assignment of the intervals can be deduced from the values of function $f$ by backtracking from $f(a_1',b_1',\ldots,a_n',b_n',D)$ to $f(0,\ldots,0)$.

**Observation 9** *There is an algorithm with complexity* $O(k^{\frac{3}{2}k+\frac{5}{2}}\log k)$, *for scheduling identical divisible tasks on a star of identical processors with* $C=0$.

*Proof* The number of processors $z$ needed to execute a task of length $k$ calculated from condition $k\le\sum_{i=1}^{z}(i-1)$ is $z\le\lceil\frac{\sqrt{8k+1}+1}{2}\rceil$. Hence, calculation of the function $f$ as described above requires determining $O(k^n\sqrt{k^n}D)$ values of $f$. $C_{\max}\le 2k+1$ because $n\le\min\{m,\lceil k\rceil\}$, processor number is not bounding ($\lceil k\rceil<m$), and at most $k$ tasks can be executed in at most $2k+1$ units of time. Since $n\le\lceil k\rceil$, and

$D \leq 2k+1$, the total number of calculations for a certain $D$ is $O(k^{\frac{3}{2}k+\frac{5}{2}})$. The optimum schedule length may be found in $\log k$ steps of binary search over values $1, \ldots, 2k$ of $D$. □

Unfortunately, the dynamic programming method described above is not polynomial in $k$. However, we infer from Observation 9 that this case can be solved in constant time if $k$ is fixed, independently of $n$. Example optimum communication patterns for $n = 2$ and several small values of $k$ are shown in Table 2. Here • denotes that a transmission must be performed, but the schedule length is the same no matter if we send load of task $T_1$, or of $T_2$. No apparent regularity can be observed in these communication patterns.

Let us note that the above dynamic programming algorithm may be used to solve a version of our problem with different tasks. The algorithm requires a modification to guarantee that each task $T_j$ received $p_j$ units of processing. We finish this section by presenting several special cases of the problem.

**Observation 10** *If $n = 1$, then the optimum schedule length can be found in $O(\log k)$ time as minimum $D$ satisfying $\sum_{i=1}^{\lfloor D \rfloor}(D-i) \geq k$.*

**Observation 11** *If $n > 1$, $i \leq n \leq k, k \in \{2i, 2i+1\}$ and $i$ is an integer, then the optimum task distribution sequence is $1, 2, 3, \ldots, n, n, n-1, \ldots, n-i+1$.*

**Table 2**

| $k$ | Communication pattern |
|---|---|
| 1 | (1, 2) |
| 2 | (1, 2, 2) |
| 3 | (1, 2, 2) |
| 4 | (1, 2, 2, 1) |
| 5 | (1, 2, 2, 1) |
| 6 | (1, 2, 2, 1, •) |
| 7 | (1, 2, 2, 1, •) |
| 8 | (1, 2, 1, 2, 2, 2) |
| 9 | (1, 2, 2, 1, 1, 2) |
| 10 | (1, 2, 1, 2, 2, •) |
| 11 | (1, 1, 2, 2, 2, 2, 2) |
| 12 | (1, 2, 2, 1, 2, 1, 1) |

*Proof* A pattern of a processor availability intervals assignment is shown in Fig. 9a. It is required in this pattern that $n > 1$, and task $n$ is assigned intervals of length $i+1, i$. Suppose $k$ is odd ($k = 2i+1$). Since the originator is activating a new processor in each time unit, this pattern is optimal by Observation 6. The relations between $n, k$ are the following: $i \leq n \leq k = 2i+1$ where $i$ is a positive integer, with the exception of $i = 1, n = 1$. This pattern can also be applied for even $k$. If $k$ is even, one should apply a pattern of interval distributions as for $k+1$, but the last $2i$ processors should stop computing $\frac{1}{2}$ before the end of the schedule dictated by the solution for odd $k+1$ (see Fig. 9b for $k = 7$, Fig. 9c for $k = 6$, and $n = 4$). □

**Observation 12** *For $n = 2$ a schedule of length $D$ exists, where $D$ is minimum integer satisfying $2k \leq \sum_{i=1}^{D}(D-i)$.*

*Proof* This observation can be shown by demonstrating that it is always possible to partition the set $\{D-1, \ldots, 1\}$ into two subsets $A, B$ so that the sum of elements in (w.l.o.g.) $A$ is equal to $k$, and is at least equal $k$ for the set $B$. It can be shown by induction: For $k = 1$, $D = 3$, decomposition of the set $\{2, 1\}$ is $A = \{1\}$ and $B = \{2\}$. Suppose the observation is satisfied for some $k \geq 1$, and set $\{D-1, \ldots, 1\}$. Then $A = \{\alpha, \ldots, \omega\}$ and $k = \alpha + \cdots + \omega$. Note that $A \neq \{D-1, \ldots, 1\}$ because $k = \alpha + \cdots + \omega$, and $\sum_{i=1}^{D}(D-i) \geq 2k$. It follows that also the set $\{D'-1, \ldots, 1\}$, where $2(k+1) \leq \sum_{i=1}^{D'}(D'-i)$, can be partitioned so that the sum of elements in $A'$ is $k+1$, and the sum of elements in $B'$ is at least $k+1$. Set $A'$ can be constructed by exchanging one of the numbers in $A$ for one in $\{D-1, \ldots, 1\} - A$ bigger by 1, or adding $\{1\}$ to $A$. This is possible because $A \neq \{D-1, \ldots, 1\}$. Consequently $B'$ can be constructed by using the remaining elements of $\{D'-1, \ldots, 1\}$ because $\sum_{i=1}^{D'}(D'-i) \geq 2(k+1)$. □

## 5.2 Processor number is bounding

Following the results of the previous section, let us observe that there seems to be no simple way of determining the optimum schedule length and, consequently, the necessary number of processors when the processor number is not bounding. And vice versa, in general it is not easy to say whether

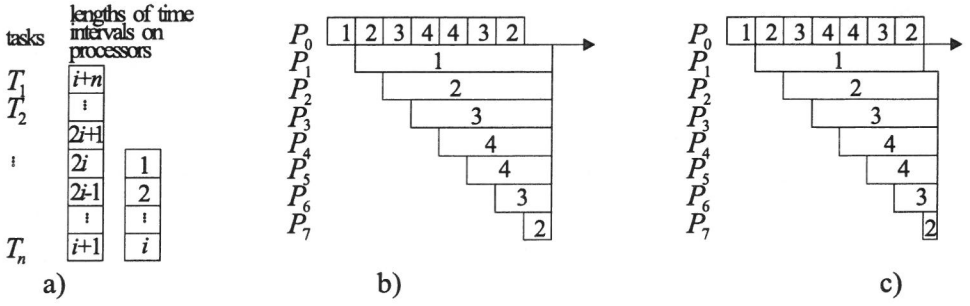


**Fig. 9** Special pattern of load distribution for $n \leq \min\{m, \lceil k \rceil\}$

the processor number is small enough to influence the construction of the schedule. A simple sufficient condition (for the processor number restricting the schedule) demands that the amount of required work exceeds the amount of load which can be processed on $m$ processors in at most $m+1$ units of time, i.e., $nk \geq \sum_{i=1}^{m}(m+1-i)$. Till the end of this section we assume that all processors are used in the optimum schedule.

The network flow algorithm from Sect. 2.2 can be used to determine optimum schedule length for a given communication pattern.

**Observation 13** *For a given communication pattern, the optimum schedule length can be found in* $O(m^6 \log m)$ *time.*

*Proof* Follows from Theorem 2 because $n < \min\{m, \lceil k \rceil\} \leq m$. □

**Observation 14** *An optimum solution can be found in* $O(m^{m^2+6} \log m)$ *time.*

*Proof* Follows from Observations 3 and 13 because $n < m$. □

## 6 Conclusions

In this paper we analyzed the problem of scheduling divisible loads on identical processors. It has been shown that this problem is **NP**-hard even for two processors if tasks are different. But we did not succeed in showing that our problem is in **NP**. If the communication pattern is given, then this problem can be solved by an algorithm which is pseudopolynomial in $m$, but polynomial in the length of the communication pattern. The complexity status of scheduling identical tasks on identical processors is even more perplexing. The key issue is if this problem is in **NP**. To show that this problem is in **NP** it is necessary to prove that communication pattern can at least be recorded in polynomial time. In this case an instance description is very compact, and consists of three numbers $m, n, k$. Thus, the communication pattern length should be polynomial in $\log m, \log n, \log k$. A feasible schedule must provide appropriate processing capacity to execute the tasks. Hence, we have $nk \leq \sum_{i=1}^{\lfloor C_{\max} \rfloor}(i-1)$ from which we conclude that $C_{\max}$ is $\Omega(\sqrt{nk})$. Thus, the communication pattern length may not be bounded by a polynomial in $\log m, \log n, \log k$. In such a situation the polynomial time description of the communication pattern may still be possible if the pattern is fixed, and it can be recognized in polynomial time. Indeed, there are cases described in Sect. 4 for which communication patterns of optimum schedules are fixed and can be identified in polynomial time. On the other hand, the results from Sect. 5.1 show that there are also cases for which no fixed regularity in the schedule construction could be observed. In conclusion, this paper leaves at least one open question of the problem **NP** membership.

## References


Bharadwaj, V., Ghose, D., Mani, V., & Robertazzi, T. (1996). *Scheduling divisible loads in parallel and distributed systems*. Los Alamitos: IEEE Computer Society Press.

Drozdowski, M. (1997). *Monographs: Vol. 321. Selected problems of scheduling tasks in multiprocessor computer systems*. Poznan University of Technology Press: Poznan. Downloadable from http://www.cs.put.poznan.pl/mdrozdowski/txt/h.ps.

Drozdowski, M., Lawenda, M., & Guinand, F. (2006). Scheduling multiple divisible loads. *International Journal of High Performance Computing*, *20*(1), 19–30.

England, D., Veeravalli, B., & Weissman, J. B. (2007). A robust spanning tree topology for data collection and dissemination in distributed environments. *IEEE Transactions on Parallel and Distributed Systems*, *18*(5), 608–620.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: a guide to the theory of NP-completeness*. San Francisco: Freeman.

Robertazzi, T. (2003). Ten reasons to use divisible load theory. *IEEE Computer*, *36*(5), 63–68.

Rothkopf, M. H. (1966). Scheduling independent tasks on parallel processors. *Management Science*, *12*, 347–447.

Sohn, J., & Robertazzi, T. (1994). *A muli-job load sharing strategy for divisible jobs on bus networks* (Technical Report 697). Department of Electrical Engineering, SUNY at Stony Brook, Stony Brook, New York.

Veeravalli, B., & Barlas, G. (2002). Efficient scheduling strategies for processing multiple divisible loads on bus networks. *Journal of Parallel and Distributed Computing*, *62*, 132–151.