Ann Oper Res (2014) 213:235–252
DOI 10.1007/s10479-011-0957-5

Power-aware scheduling of preemptable jobs on identical parallel processors to minimize makespan

R. Różycki · J. Węglarz

Published online: 8 September 2011

Abstract This paper deals with power-aware scheduling of preemptable jobs on identical parallel processors to minimize schedule length when jobs are described by continuous, strictly concave functions relating their processing speed at time t to the amount of power allotted at the moment. Power is a continuous, doubly constrained resource, i.e. both: its availability at time t and consumption over scheduling horizon are constrained. Precedence constraints among jobs are represented by a task-on-arc graph. A methodology based on properties of optimal schedules is presented for solving the problem optimally for a given ordering of nodes in the graph. Heuristics for finding an ordering which leads to possibly short schedules are proposed and examined experimentally.

Keywords Power-aware computing · Job model · Preemptable jobs · Makespan minimization

1 Introduction

Green computing, as a general philosophy aimed at efficient and environmentally friendly usage of computer resources, gained recently an increasing interest among OR and IT specialists. One of the most promising directions of green computing is ecologically aware management of power/energy on different levels of computer systems. We consider the problem on the level of a single computer system, where a particular attention is paid on processors (CPU) as one of the most energy consuming elements of such systems. One of the research directions towards an advanced power management on the level of a single computer system is the use of variable speed processors (VSP) (Boyer et al. 2006). Modern operating systems utilizing e.g. Intel's Speedstep or Foxton as well as AMD's PowerNow! technologies are able to control the speed of processors to prolong the battery life or to improve computing efficiency. Therefore, an operating system of a computer can decide not only which job to

R. Różycki · J. Węglarz (✉)
Institute of Computing Science, Poznan University of Technology, 2 Piotrowo St., 60-965 Poznań, Poland
e-mail: jan.weglarz@cs.put.poznan.pl

perform at the moment but also at what processing rate. We assume for simplicity that considered variable speed processors are able to adjust their clock period at each cycle at no cost.

Since the processing rate of a job is related to the power/energy used during its execution, several job processing models have been proposed in literature to represent this relation (Bansal et al. 2005; Irani and Pruhs 2005; Pruhs et al. 2005). The power vs. speed model is the most popular, where a single continuous, strictly convex function represents the relation.

In this paper we consider a speed vs. power model, in which jobs are characterized by (different) continuous, strictly concave functions relating their processing speeds to the amount of power allotted at a time (Węglarz 1981).

Of course, the assumption that speed function is continuous remains only an approximation of discrete speed increments in real processors.

Schedule length of a given set of jobs processed on processors of a single computer system is to be minimized for a given level of energy available. Such a problem belongs to the class of *laptop problems* (Bunde 2006), which are typical for portable electronic devices driven by energy accumulated in batteries with limited capacity. However, in contrary to the power vs. speed model in which power is not limited, we consider it as a doubly constrained resource for which both: consumption (i.e. energy), and temporary availability are constrained. The similar assumption, defined as bounded speed model, where a single processor can vary its speed between 0 and a fixed maximum value, was proposed in Bansal et al. (2008). Nevertheless, our model is more natural in the multiprocessor setting, since it enables to share power among processors as a common renewable resource.

We consider a set of precedence-related and preemptable jobs as well as parallel identical processors, since it is a natural way to model a set of computer applications that have to be scheduled on contemporary computer system equipped with an advanced multicore processor.

We represent the precedence relation among jobs by a task-on-arc directed graph. This representation, commonly used in project scheduling, has been also used for solving machine scheduling problems by transforming them to LP problems (see Błażewicz et al. 2007 as survey) for a given ordering of nodes (i.e. events) in such a digraph. If in a digraph each two nodes are connected by a path then the ordering of nodes is unique and corresponding scheduling problem is polynomially solvable. In general, the optimal solution depends on the ordering of nodes, and thus finding an ordering that leads to the best schedule is of crucial importance. Unfortunately, no polynomial algorithm for finding such an ordering is known even for the simplest case of parallel, identical machines. Thus, it is advisable to consider various heuristic or metaheuristic approaches.

Although we consider a single computer system, presented approach may be applied to the case of a cluster of identical computational nodes driven by a common power source and managed on the level of a single metascheduler.

In Sect. 2 we describe and compare two models of job processing. Scheduling problem is formulated in Sect. 3. In Sect. 4 we present a methodology for finding optimal schedules for independent jobs. This methodology is generalized in Sect. 5 for the case of dependent jobs. In this section we also propose two heuristics for finding an ordering of nodes leading to possibly short schedules. A computational experiment and its results are described in Sect. 6. Section 7 contains final remarks and future research directions.

2 Job processing models

Let us start with the model commonly used in the literature (e.g. Bansal et al. 2005; Irani and Pruhs 2005) concerning power management in microprocessor systems. The following form of the function (*power usage function*) is utilized to express the relation between processing speed s of a processor, and power p consumed during job processing:

$$p(s) = s^{\alpha}, \quad \alpha > 1 \tag{1}$$

It is easy to notice that function $p(s)$ is strictly convex. In particular, for microprocessors based on CMOS technology α is assumed to be equal to 3 (Bunde 2006).

Moreover, it is assumed that job i is characterized by the parameter $w_i > 0$ being the number of CPU cycles needed by job i to be processed. Of course, processing time of job i depends on its size w_i and on the processing speed of a processor executing this job. A job i started at time a_i is accomplished at completion time C_i if the following equation is fulfilled:

$$\int_{a_i}^{C_i} s(t)dt = w_i \tag{2}$$

During the execution of job i the energy amount E_i given by the formula

$$E_i = \int_{a_i}^{C_i} p(s(t))dt \tag{3}$$

is consumed.

It is worth to notice that in model (1) the processing speed of a processor is treated as a decision variable and determines power p. Moreover, s is not limited, i.e. $s \in [0, \infty)$. As a consequence, the resulting power is theoretically unlimited too. However, power directly determines the temperature of a microprocessor (nearly all energy consumed by a processor is released as heat). Thus, no limits for power may lead to a processor overheating and, in consequence, to a serious damage of a computer system.

Let us pass to the model which determines a temporal rate of the job execution. As a consequence, instead of power usage functions we have so called *processing speed functions* (*speed functions* in short). Formally the model is expressed as follows:

$$\dot{x}_i(t) = \frac{dx_i(t)}{dt} = s_i(p_i(t)), \quad x_i(0) = 0,\ x_i(C_i) = w_i \tag{4}$$

where

- $x_i(t)$—is the state of job i at time t, i.e. the amount of data processed by t for job i,
- $s_i(\cdot)$—is a continuous, strictly concave function, $s_i(0) = 0$,
- $p_i(t)$—is an amount of power allotted to job i at time t.

Model (4) was studied in Węglarz (1981) in the problem of allocating continuously-divisible, doubly constrained resource among project activities. In this problem power $p(t)$ allotted to a job at time t is doubly constrained, i.e.

$$\sum_{i=1}^{n} p_i(t) \le P \quad \text{for every } t > 0 \tag{5}$$

$$\sum_{i=1}^{n} \int_0^{C\max} p_i(t)dt \le E \tag{6}$$

where $P > 0$, $E > 0$ are known.

The constraint (5) guarantees that a total usage of power at every moment does not exceed the available amount P, whereas the constraint (6) ensures that a total consumption of energy up to the end of the last job ($C_{\max}$) in a schedule meets the known limit E.

It is easy to notice that the models (1) and (4), (5), (6) are equivalent if

$$s_i(\cdot) = s(\cdot) = p^{-1}(\cdot) = p_i^{1/\alpha}, \quad i = 1, 2, \ldots, n, \alpha > 1$$

However, the model (4), (5), (6) is much more general. First of all, time t enters directly into this model, and thus power allotted to jobs can vary over time. Secondly, functions $s_i(\cdot)$ can be different for different jobs. This is very important since power consumption per-cycle may differ for various instructions executed by a processor designed basing on one of modest architectures (see e.g. Boyer et al. 2006).

Moreover, functions $s_i(\cdot)$ can be also not only power functions but arbitrary continuous, strictly concave ones. Thus this model may be useful even when future technologies and microcomputer architectures will require another type of functions.

Since power is limited: $p_i(t) \in [0, P]$ in model (4), (5), (6), thus temperature of a microprocessor executing computational jobs is limited too.

3 Problem formulation

Consider a set of n precedence-related, preemptive jobs and m parallel identical processors. All jobs are ready to be processed at time 0. Each job requires for its processing a processor and an amount of power. Each job is performed by at most one processor at a time and a processor is able to process at most one job at a time. Processing rate of a job depends on the amount of power $p_i(t)$ allotted to job i at a time t and this relation is expressed by (4). Job i is characterized by the speed function $s_i(\cdot)$ and the size w_i. Precedence relations among jobs are represented by a task-on-arc digraph. Both power and energy are limited and available in amounts P and E, respectively.

This formulation allows to model practical situations, where a set of dependent modules of programs have to be executed on a multiprocessor portable device with processors driven by the common energy source—a battery of limited capacity. To prevent the computer system from overheating a power usage limit is established. We assume that processing rate function of job i in model (4) is increasing and strictly concave since this corresponds to the real relation between the temporal power usage and the processing rate in contemporary microprocessor systems.

The objective is to find a vector function $\boldsymbol{p}^*(t) = [p_1^*(t), p_2^*(t), \ldots, p_n^*(t)]$, $p_i^*(t) \ge 0$, $i = 1, 2, \ldots, n$ which, under the constraints imposed, minimizes the schedule length $C_{\max} = T$. Knowing $\boldsymbol{p}^*(t)$ we are able to calculate the energy consumption for job i by integrating $p_i^*(t)$ up to the completion time C_i. We will show that finding $\boldsymbol{p}^*(t)$ may be decomposed into two strongly interrelated subproblems: the subproblem of determining of the optimal sequence of jobs on processors and the subproblem of optimal power allocation to jobs already sequenced.

4 Independent jobs

Assume hypothetically that a job does not require any processor to be performed and may be processed using an amount of power only. In consequence, we deal with the subproblem of T-minimal allocation (respecting limits P and E) of power to jobs only and the following theorem is valid (Węglarz 1981).

Theorem 1 *If jobs are independent and $s_i(\cdot)$, $i = 1, 2, \ldots, n$ are strictly concave then all jobs finish simultaneously and are processed using constant amounts of power given by the formula*

$$p_i^*(t) = p_i^* = s_i^{-1}(w_i/T^*), \quad i = 1, \ldots, n, \ t \in \langle 0, T^* \rangle \tag{7}$$

where T^* is the (unique) positive root of the equation:

$$T \sum_{i=1}^{n} s_i^{-1}(w_i/T) = E \quad \text{if} \sum_{i=1}^{n} s_i^{-1}(w_i/T) \leq P \tag{8}$$

or

$$\sum_{i=1}^{n} s_i^{-1}(w_i/T) = P \tag{9}$$

otherwise. $T^*(w_1, w_2, \ldots, w_n)$ is a convex function.

Notice that (8) and (9) are direct consequence of (6) and (5). In the first case the active constraint is from the side of E, not P. It means that (7), with T^* calculated from (8), is the optimal solution of the problem since it does not exceed the power limit P. It may happen however that P is a critical constraint for the given instance of the problem, and then the minimum schedule length is found from (9).

Let us comment now on Theorem 1 from the view point of finding optimal solutions. To find the schedule of the minimum length it is better to start with (8), since it is usually of simpler form than (9). For example, for functions:

$$s_i(\cdot) = c_i p_i^{1/\alpha_i}, \quad c_i > 0, \ \alpha_i \in \{2, 3, 4\} \tag{10}$$

both (8) and (9) are analytically solvable algebraic equations of an order less than or equal to 4, however (8) is of an order less by 1, and thus is analytically solvable also for $\alpha_i = 5$. If the sum of power allocated to jobs using (8) exceeds available amount P, we have to solve (9), but the information obtained by solving (8) is valuable anyway.

Let us pass now to the problem formulated in Sect. 3 assuming that jobs are independent. It is easy to notice that if $n \leq m$, then the optimal schedule is defined by (7), (8), (9) with $n = m$.

However, if $n > m$, then for finding the optimal schedule we have to consider all m-element combinations from the set of n jobs. Each combination represents a single group of jobs that may be performed in parallel on limited number of processors. Since the processors are identical, an order of jobs in such combination may be neglected. Although the jobs are preemptable, we assume that preemption of a job within a combination is forbidden. However, a job is still able to migrate among processors because in consecutive combinations it may be allotted to different processors at no additional cost. Denote combinations by Z_k, $k = 1, 2, \ldots, r = \binom{n}{m}$. Denote also by K_i, $i = 1, 2, \ldots, n$, the set of indices of Z_k's

containing job i and w_{ik} part of job i processed in Z_k. Let T_k and E_k, $k = 1, 2, \ldots, r$, denote respectively: the length of the part of the schedule corresponding to Z_k and an amount of energy consumed for processing parts of jobs from Z_k. Notice that for a given level of P, $T_k = T_k(\{w_{ik}\}_{i \in Z_k}, E_k)$, $k = 1, 2, \ldots, r$, where w_{ik}, E_k are unknown a priori. Since for each Z_k we have $n = m$, we can apply Theorem 1. For the simplest situation of processing speed functions (10) where $\alpha_i = \alpha > 1$, $c_i = 1$, $i = 1, 2, \ldots, n$, T_k is calculated from:

$$T_k = \max\left(\sqrt[\alpha-1]{\frac{1}{E_k} \sum_{i \in Z_k} (w_{ik})^\alpha},\ \sqrt[\alpha]{\frac{1}{P} \sum_{i \in Z_k} (w_{ik})^\alpha} \right)$$

Finally, we get the following corollary for the general case of arbitrary strictly concave processing speed functions.

Corollary 2 *For strictly concave $s_i(\cdot)$, $i = 1, 2, \ldots, n$, the optimal schedule for independent, preemptable jobs is obtained by solving the following convex programming problem*:

Minimize:

$$T = \sum_{k=1}^{r} T_k \left(\{w_{ik}\}_{i \in Z_k}, E_k \right) \tag{11}$$

Subject to:

$$\sum_{k=1}^{r} E_k \leq E \tag{12}$$

$$\sum_{k \in K_i} w_{ik} = w_i \quad i = 1, 2, \ldots, n \tag{13}$$

$$w_{ik} \geq 0, \qquad T_k \geq 0, \quad i = 1, 2, \ldots, n; k \in K_i \tag{14}$$

where T_k are calculated as functions of E_k and $\{w_{ik}\}$ for $i \in Z_k$ using Theorem 1. Of course, in (8), (9) it has to be assumed that $n = m$. Values of $\boldsymbol{p}^*(t)$ (i.e. p^*_{ik}, $i \in Z_k$ and $k = 1, 2, \ldots, r$) are calculated for the optimal values of w_{ik}, E_k and T_k, using (7) for each Z_k.

Corollary 2 describes the simplest way for finding an optimal solution of the problem of scheduling independent jobs for arbitrary strictly concave $s_i(\cdot)$. Of course, it may be applied to small instances only, since the number of variables in (11)–(14), grows exponentially with n.

5 Dependent jobs

We assume that a task-on-arc digraph represents the precedence relation in the set of jobs. Moreover, the nodes are numbered in such a way that, if arc (j_a, j_b) represents a job, then $j_a < j_b$. Nodes j_a, j_b are called respectively *head* and *tail* ones for this job. Of course, for each directed graph without cycles such an ordering is always possible, and may be found in $O(n^2)$ time. We assume, without loss of generality, that node 1 (beginning node) is the

only node without predecessors, and node q (terminal node) is the only node without successors. Moreover, two nodes may be linked by at most one arc in a task-on-arc digraph. Let Q_j, $j = 1, 2, \ldots, q-1$, denote the set of jobs that may be processed between the occurrences of events represented by nodes j and $j+1$. Such sets are called *main sets.* Of course, parts of jobs processed in particular Q_j are independent. Notice that there exist only one sequence $S = [Q_1, Q_2, \ldots, Q_{q-1}]$ of main sets for a particular ordering of nodes O. An example of the task-on-arc digraph and related main sets is presented in Fig. 1.

5.1 Optimal solution for a given ordering of nodes

Let us assume that the sequence S is given representing ordering of nodes O. Since parts of jobs processed in particular Q_j's are independent, we can apply the approach described in Sect. 4 for each Q_j, $j = 1, 2, \ldots, q-1$. This means that we have to consider all m-element combinations from all Q_j's for which $|Q_j| > m$. Let us keep for these combinations the denotations Z_k, $k = 1, 2, \ldots, r$. If we denote by A the set of main sets with the size greater than m, then total number r of all combinations Z_k for a given ordering of nodes O equals:

$$r = q - |A| - 1 + \sum_{Q_j \in A} \binom{|Q_j|}{m}$$

An example of combinations Z_k for a given sequence of main sets and $m = 2$ is showed in Fig. 2.

In consequence, for finding an optimal schedule for a given ordering of nodes O, one has to solve the problem (11)–(14) where Z_k, $k = 1, 2, \ldots, r$ are constructed as presented above.

Using Theorem 1 one can formulate two properties of an optimal schedule.

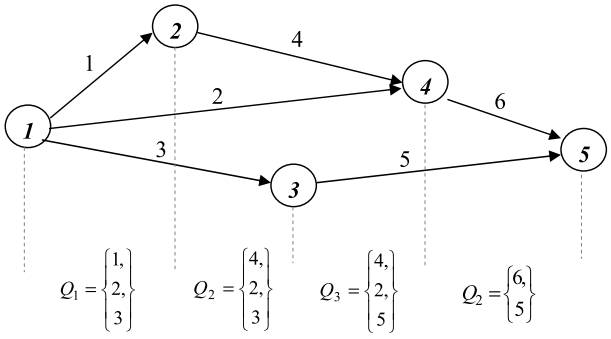

Fig. 1 Example of a sequence of main sets

$$Q_1 = \left\{\begin{matrix}1,\\2,\\3\end{matrix}\right\}, \quad Q_2 = \left\{\begin{matrix}4,\\2,\\3\end{matrix}\right\}, \quad Q_3 = \left\{\begin{matrix}4,\\2,\\5\end{matrix}\right\}, \quad Q_4 = \left\{\begin{matrix}6,\\5\end{matrix}\right\},$$

$$\overbrace{Z_1 \quad Z_2 \quad Z_3}^{} \quad \overbrace{Z_4 \quad Z_5 \quad Z_6}^{} \quad \overbrace{Z_7 \quad Z_8 \quad Z_9}^{} \quad | \; Z_{10}$$

$$\left\{\begin{matrix}1,\\2\end{matrix}\right\}, \left\{\begin{matrix}1,\\3\end{matrix}\right\}, \left\{\begin{matrix}2,\\3\end{matrix}\right\}, \quad \left\{\begin{matrix}4,\\2\end{matrix}\right\}, \left\{\begin{matrix}4,\\3\end{matrix}\right\}, \left\{\begin{matrix}2,\\3\end{matrix}\right\}, \quad \left\{\begin{matrix}4,\\2\end{matrix}\right\}, \left\{\begin{matrix}4,\\5\end{matrix}\right\}, \left\{\begin{matrix}2,\\5\end{matrix}\right\}, \quad \left\{\begin{matrix}6,\\5\end{matrix}\right\}$$

Fig. 2 Example of a set of combinations Z_k

Corollary 3 *The minimum energy level which ensures the minimum schedule length for a given power level P, ordering of nodes O and corresponding combinations Z_k, is equal to*:

$$E_{\min} = \sum_{k=1}^{r} T_k^* \sum_{i \in Z_k} s_i^{-1}(w_{ik}^*/T_k^*) \tag{15}$$

where $T_k^*, w_{ik}^*, k = 1, 2, \ldots, r, i \in Z_k$ are the optimal values obtained by solving problem (11), (13), (14) where T_k^* are calculated for the level of power P from the equations:

$$\sum_{i \in Z_k} s_i^{-1}(w_{ik}/T_k) = P, \quad k = 1, 2, \ldots, r$$

Corollary 4 *The minimum level of power which ensures the minimum schedule length for a given energy level E, ordering of nodes O, and combinations Z_k, is equal to*:

$$P_{\min} = \max_k \left\{ \sum_{i \in Z_k} s_i^{-1}(w_{ik}^*/T_k^*) \right\} \tag{16}$$

where $T_k^*, w_{ik}^*, k = 1, 2, \ldots, r$, $i \in Z_k$ are the optimal values obtained by solving problem (11)–(14) in which "=" should be put in (12), and T_k^* are calculated from the equations:

$$T_k \cdot \sum_{i \in Z_k} s_i^{-1}(w_{ik}/T_k) = E_k, \quad k = 1, 2, \ldots, r$$

Let us underline another interesting aspect of numerical solving of the convex programming problem (11)–(14). It is easy to observe, that quite often a situation may appear where for a given sequence of main sets S, two or more processor feasible sets contain the same combination of job indices. Such a situation may also be observed in Fig. 2 (e.g. see combinations Z_3 and Z_6). It is easy to show that an optimal schedule obtained by replacing all duplicated combinations by a single combination will never be longer than the original one. Thus it is practically justified to propose a simple procedure (Algorithm 1) of generating combinations without repetitions.

The procedure in Algorithm 1 assumes in step 4 that effective algorithm of generating of all combinations in lexicographic order is known. An example of such an algorithm may be found in Er (1985).

5.2 Heuristics for nodes ordering

To find an optimal schedule for the problem with dependent jobs it is necessary to find an ordering O^* of graph nodes which leads to an optimal sequence of main sets S^*. To this end, the full enumeration approach over the full spectrum of feasible node orderings is needed. Alternatively, one can apply a metaheuristic approach where searching over the set of all feasible node orderings is directed by algorithm-specific rules. A metaheuristic method produces results relatively fast but quality of found solutions is unpredictable and highly depends on the proper preliminary tuning of parameters and/or operators of the algorithm. Of course, both approaches are applicable for small instances of the problem only or for the instances with a dense task-on-arc precedence graph, since for each ordering a problem (11)–(14) has to be solved.

Algorithm 1

1. $j = 1$; $B = \emptyset$
2. Let set C contains all the jobs beginning in node j,
3. Index anew (starting from 1) all the jobs from C in arbitrary order, if $B \neq \emptyset$ index jobs from B consecutively (starting from $|C| + 1$) in arbitrary order.
4. if $(|B| + |C| \leq m)$
   then
   generate only one combination of all jobs from B and C,
   otherwise
   generate all combinations of m jobs over set $B \cup C$ in lexicographic order of its indices until the first job from set B appears on the first position in a new combination (the combination triggering stopping criteria is excluded from the current combination set).
5. if $(j < q)$
   then
   $j = j + 1$; let the set B contain all the jobs not completed in node j; go back to step 2;
   otherwise
   stop.

Below we propose several simple constructive heuristics which create a single ordering of nodes for a given problem instance.

Notice that the processing time of a job is unknown a priori in the considered scheduling problem. Instead, the size of a job may be taken as its base metrics used in a heuristic. Unfortunately, the relation between power allocation for a job and its effective processing time defined by (4) is ignored in this approach. Therefore, a job size may have only an auxiliary meaning when a power allocation is unknown. Thus, a preliminary allocation of power to jobs is necessary in order to respect processing times of jobs in a heuristics. We propose the approach where the preliminary allocation of power is found by solving (11)–(14) for an arbitrary feasible ordering of nodes in a graph. In consequence, we are able to calculate both: a preliminary processing time, as well as an energy consumption for each job. These two metrics may be further directly utilized in the heuristics.

We will consider three variants of the proposed heuristics, each of them taking into account one of the following parameters as a job metrics:

variant 1—size w_i,

variant 2—processing time τ_i,

variant 3—energy consumption e_i.

It is worth stressing that the metrics of jobs in variants 2 and 3 base on some preliminary ordering O. It is an open question, how far a particular ordering O determines the final ordering produced by a heuristics. We will try to estimate this relation on the bases of the computational experiment described in Sect. 6.

5.2.1 Level based heuristics

Levord is a method which has proved its efficiency for the classical problem of scheduling preemptable jobs on parallel identical processors (Józefowska et al. 2004). It exploits the intuition that jobs with greater sizes should be performed longer and thus they have to be present in a greater number of main sets than the jobs with small sizes.

We propose two methods of calculating a node level.

Let us denote by *pLevord*(a) the method where the level of node j is calculated as the length of the longest path from the node j to node q assuming that the weight of each arc on this path is equal to the chosen metrics a, $a \in \{w_i, \tau_i, e_i\}$ of the represented job.

Second method, denoted by *sLevord*(a), finds the level of node j as the sum of weights a of all arcs belonging to at least one path from the node j to node q.

5.2.2 Load based heuristic

An impact of the limited number of processors m on a constructed schedule is not respected in the level based heuristics described in Sect. 5.2.1. We will use the *Nodord* algorithm proposed in Słowiński (1978) as a load driven heuristic for node ordering. The algorithm was originally developed for the resource constrained project scheduling problem and further successfully applied to the aforementioned machine scheduling problem (Józefowska et al. 2004). We propose three variants of the algorithm denoted by *Nodord*(a) for the considered problem, each one basing on a single job metrics a, $a \in \{w_i, \tau_i, e_i\}$. Let us present the original algorithm (Słowiński 1978) with necessary modifications.

Algorithm 2 *Nodord*(a)

1. Number the nodes in the directed graph from 1 to q, so that for each arc (j_a, j_b), $j_a < j_b$.
2. Number the arcs (jobs) in the graph from 1 to n according to the increasing number of their head nodes, and if two or more arcs share the same head node, solve ties according to the increasing tail-node number.
3. Using a heuristic method for scheduling nonpreemptable jobs on m identical parallel processors and assuming that processing time of a job is equal to its value of metrics a, find job start times t_i $i = 1, 2, \ldots, n$ that satisfy precedence constraints and the nonpreemptability condition.
4. For each node except the beginning and terminal ones calculate the value of the parameter

$$d_j = \min_{i \in D_j}\{t_i\}, \quad j = 2, 3, \ldots, q-1,$$

   where D_j is the set of jobs beginning in node j and d_j is the latest possible time of occurrence of node j for the task start times defined in step 3.
5. Number the nodes from 2 to $q-1$ according to increasing values of parameter d_j.

The heuristic method used in step 3 determines the quality of the *Nodord*(a) algorithm. It constructs a nonpreemptable schedule on m machines via a series of partial schedules (Józefowska et al. 2004). A precedence feasible job scheduled as next in a partial schedule is chosen basing on its priority. Applying the results of Józefowska et al. (2004), we calculate a priority of a job as the total length (defined by metrics a) of all its successors.

6 Computational experiment

To evaluate efficiency of the proposed heuristic approaches we have performed a computational experiment. The algorithms were implemented in C++ and run on SUN Fire V490 equipped with UltraSPARC IIIi processor and 32 GB RAM memory. Due to the fact that no

standard library of instances is available for the considered problem, the test instances were generated randomly.

The experiment was limited to small instances since we were interested in comparing the results with the optimal ones. Thus we assumed that the number of nodes $q \in \{6, 7\}$, and the number of processors m equals 2 and 3. The density μ of the graph was set to the three values: 0.3, 0.5, 0.7. Of course, the number of jobs n is related to the number of nodes q and the density of the graph μ and was calculated as $\text{Round}(q(q-1)\cdot\mu/2)$. For each set of parameters m, q, μ, (instances for $q = 7$ were limited only to the case of $\mu = 0.5$) 10 instances were generated. The job sizes were generated uniformly from the interval [0,100]. Moreover, each job i was characterized by a processing speed function $s_i(\cdot)$ of the form: $s_i(\cdot) = p_i^{1/\alpha_i}$, where α_i have been set to the one of values: 2 or 3, with equal probability.

Different combinations of amounts of power P and energy E were tested. First, we set these values in the way that each one of constraints (5) or (6) was active exclusively. Moreover, we established P and E to the levels where both the constraints remained active for tested instances.

We used CFSQP 2.5 solver (Lawrence et al. 1997) for solving the non-linear programming problem (11)–(14). Algorithm 1 was applied to minimize the number of the variables in (11)–(14).

We utilized optimal solutions as the reference results in our experiment. Optimal schedules have been found by a full enumeration procedure which explored the set of all precedence feasible job orderings for a given problem instance. In Table 1 we show data representing the relative range R of schedule lengths for different problem sizes estimated basing

Table 1 Relative range of schedule lengths and average computational time ($E = 1000$)

#	q/m	μ	R [%] ($AvgTime$ [s])		
			$P = 5$	$P = 30$	$P = 100$
1	6/2	0.3	1.57 (1.60)	1.67 (2.53)	2.11 (2.34)
2	6/2	0.5	7.20 (22.52)	7.65 (9.91)	5.0 (7.76)
3	6/2	0.7	4.31 (51.75)	4.22 (16.94)	3.16 (18.89)
4	6/3	0.3	3.7 (1.33)	3.6 (0.82)	3.8 (1.07)
5	6/3	0.5	4.6 (139.21)	5.1 (33.12)	7.0 7.0 (31.40)
6	6/3	0.7	3.1 (572.83)	4.5 (406.43)	4.6 (379.53)
7	7/2	0.5	13.74 (295.42)	8.22 (165.17)	10.88 (153.49)
8	7/3	0.5	10.09 (293.72)	16.70 (1120.31)	18.86 (1089.71)

on all precedence feasible nodes ordering. This value was calculated as:

$$R = \frac{1}{H} \sum_{h=1}^{H} \frac{C^W_{h\max} - C^*_{h\max}}{C^*_{h\max}} \cdot 100\%$$

where
$C^*_{h\max}$—is the best schedule length for instance h
$C^W_{h\max}$—is the worst schedule length for instance h
H—is the number of tested instances.

In Tables 2, 3, 4 the results obtained by heuristics are presented. In consecutive rows the following metrics for each algorithm are placed:

– average relative deviation from the optimum,
– maximum relative deviation from the optimum,
– number of instances for which the algorithm found an optimal solution.

Each single row in Tables 2, 3, 4 represents the values of above metrics for a specific problem size.

For each instance size we mark the best value of a particular metric by bold font.

The full enumeration approach applied for finding optimal orderings of nodes in a graph allows to recognize characteristics of a search space of all precedence feasible orderings from the schedule length point of view. Schedules most distant from the optimum were obtained for instances with an average value of graph density. The graphs with density μ set to 0.5 led to the variability of schedule lengths, measured by R, up to approx. 19%. It may be justified by the fact that the number of different feasible schedules takes the highest values in this case. Another interesting property is that the smaller amount of available power P we set in a instance, the smaller value of average computational time for finding optimum we get. Since the main part of computational time is consumed for the solution of problem (11)–(14), it may be interpreted in such a way that finding the optimal power allocation for instances with relatively small value of P by a non-linear solver is probably simpler.

A general observation resulting from the computational experiment is that efficiency of the proposed heuristics do not differ much. It is especially visible if we take into account the maximum relative deviation from the optimum. One can also conclude that in most cases all the heuristics are misled by the same instance of the problem. Notice that in the worst case (see the seventh instance size in Table 4) the schedule related to the node ordering generated by all heuristics was longer by 12.4% than the optimal one. It is obvious, that various mechanisms implemented in our heuristics are insufficient for such malicious problem instances. However, all the heuristics found the results worse on average by approx. 2.5% only than the optimal ones. In general, the number of the optimal orderings found ranges from 10% to 100% and it depends mainly on the density of the graph and amounts of P and E.

Let us analyze sensitivity of proposed heuristics on the values of P and E. It is easy to observe that when the amount of power P is relatively small compared to an amount of energy E, then the heuristics found relatively better results than in the case of big P. It may denote that the instances with an amount of energy E as most tight limit are harder to solve then instances where power P should be respected only.

It is difficult to compare a computational time of the proposed heuristic algorithms since their computations are significantly dominated by solving the non-linear programming problem (11)–(14). The time needed by heuristics for constructing a single, potentially optimal ordering may be taken as negligibly small.

Table 2 Power related instances ($P = 5$, $E = 1000$)

#	q/m	μ	$pLevord(w_i)$	$pLevord(\tau_i)$	$pLevord(e_i)$	$sLevord(w_i)$	$sLevord(\tau_i)$	$sLevord(e_i)$	$Nodord(w_i)$	$Nodord(\tau_i)$	$Nodord(e_i)$
1	6/2	0.3	0.0016	**0.0**	0.0016	0.0016	**0.0**	0.0016	0.0016	**0.0**	0.0016
			0.0157	**0.**	**0.0157**	**0.157**	**0.0**	**0.0157**	**0.0157**	**0.0**	**0.0157**
			9	**10**	9	9	**10**	9	9	**10**	9
2		0.5	**0.001**	0.0011	**0.0010**	**0.0010**	0.0011	**0.0010**	**0.0010**	0.0011	**0.0010**
			0.021	0.021	**0.021**	**0.021**	0.021	**0.021**	**0.021**	0.021	**0.021**
			8	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
3		0.7	**0.003**	0.004	0.004	0.006	0.006	0.006	**0.003**	0.004	**0.003**
			0.01	**0.01**	**0.01**	0.025	0.025	0.0251	**0.01**	**0.01**	**0.01**
			8	7	7	6	6	7	**8**	6	7
4	6/3	0.3	0.0057	**0.003**	0.0053	0.0057	**0.003**	0.0053	0.0053	**0.003**	0.0053
			0.0304	**0.0304**	**0.0304**	**0.0304**	**0.0304**	**0.0304**	**0.0304**	**0.0304**	**0.0304**
			7	**9**	7	8	**9**	8	8	**9**	8
5		0.5	**0.015**	0.017	0.023	**0.015**	0.017	0.023	**0.015**	0.017	0.023
			0.045	**0.045**	0.080	**0.045**	**0.045**	0.080	**0.045**	**0.045**	0.080
			3	2	2	**3**	2	2	**3**	2	2
6		0.7	0.012	0.018	0.011	0.018	0.018	0.014	0.011	0.015	**0.010**
			0.032	0.044	**0.032**	0.044	0.044	0.033	**0.032**	0.044	**0.032**
			3	3	**4**	3	3	**4**	3	3	**4**
7	7/2	0.5	0.014	0.013	0.015	0.014	**0.012**	0.014	0.014	**0.012**	0.014
			0.02	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**
			1	**2**	1	1	**2**	1	1	**2**	1
8	7/3	0.5	0.015	0.018	0.022	0.015	0.015	0.015	0.015	0.015	**0.013**
			0.023	0.025	0.025	**0.023**	**0.023**	**0.023**	**0.023**	**0.023**	0.024
			3	3	3	3	3	3	3	3	**4**

Table 3 Mixed, power/energy related instances ($P = 30$, $E = 1000$)

#	q/m	μ	$pLevord(w_i)$	$pLevord(\tau_i)$	$pLevord(e_i)$	$sLevord(w_i)$	$sLevord(\tau_i)$	$sLevord(e_i)$	$Nodord(w_i)$	$Nodord(\tau_i)$	$Nodord(e_i)$
1	6/2	0.3	0.004	0.007	**0.0**	0.004	0.007	**0.0**	**0.0**	0.004	0.004
			0.0375	0.0375	**0.0**	0.0375	0.0375	**0.0**	**0.0**	0.0375	0.0375
			9	8	**10**	9	8	**10**	**10**	9	9
2		0.5	**0.018**	0.019	**0.018**	**0.018**	0.019	**0.018**	**0.018**	0.019	**0.018**
			0.031	**0.031**	**0.031**	**0.031**	**0.031**	**0.031**	**0.031**	**0.031**	**0.031**
			7	6	**7**	**7**	6	**7**	**7**	6	**7**
3		0.7	0.004	**0.003**	0.004	0.014	0.014	0.015	0.004	**0.003**	0.004
			0.030	**0.030**	**0.030**	0.114	0.114	0.114	**0.030**	**0.030**	**0.030**
			7	**8**	7	8	8	7	7	**8**	7
4	6/3	0.3	0.0043	0.0053	**0.0037**	0.0043	0.0053	**0.0037**	**0.0037**	0.0053	0.0043
			0.0241	**0.0241**	**0.0241**	**0.0241**	**0.0241**	**0.0241**	**0.0241**	**0.0241**	**0.0241**
			6	5	**7**	6	5	**7**	**7**	5	6
5		0.5	**0.015**	0.016	**0.015**	**0.015**	0.018	**0.015**	**0.015**	0.016	**0.015**
			0.042	0.033	0.042	0.033	0.042	0.028	**0.022**	0.024	0.028
			4	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
6		0.7	0.0021	0.0024	0.0029	0.0025	0.0025	0.0033	**0.0016**	0.0021	0.0025
			0.01	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**
			7	**7**	6	**7**	**7**	5	**7**	**7**	6

Table 3 (*Continued*)

#	q/m	μ	$pLevord(w_i)$	$pLevord(\tau_i)$	$pLevord(e_i)$	$sLevord(w_i)$	$sLevord(\tau_i)$	$sLevord(e_i)$	$Nodord(w_i)$	$Nodord(\tau_i)$	$Nodord(e_i)$
7	7/2	0.5	0.017	0.018	**0.016**	0.018	0.017	**0.016**	0.017	**0.016**	**0.016**
			0.034	**0.034**	**0.034**	**0.034**	**0.034**	**0.034**	**0.034**	**0.034**	**0.034**
			1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
8	7/3	0.5	0.014	0.017	0.017	0.014	0.014	0.014	0.014	0.014	**0.007**
			0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	**0.025**
			2	2	2	2	2	2	2	**3**	**3**

Table 4 Energy related instances ($P = 100$, $E = 1000$)

#	q/m	μ	$pLevord(w_i)$	$pLevord(\tau_i)$	$pLevord(e_i)$	$sLevord(w_i)$	$sLevord(\tau_i)$	$sLevord(e_i)$	$Nodord(w_i)$	$Nodord(\tau_i)$	$Nodord(e_i)$
1	6/2	0.3	**0.0**	0.0051	0.0042	**0.0**	0.0051	0.0042	0.0042	0.0051	**0.0**
			0.0	**0.0408**	**0.0408**	**0.0**	**0.0408**	**0.0408**	**0.0408**	**0.0408**	**0.0**
			10	9	9	**10**	9	9	9	9	**10**
2		0.5	**0.011**	0.013	**0.011**	**0.011**	0.012	**0.011**	**0.011**	0.012	**0.011**
			0.047	**0.047**	**0.047**	**0.047**	**0.047**	**0.047**	**0.047**	**0.047**	**0.047**
			7	6	**7**	**7**	6	**7**	**7**	6	**7**
3		0.7	0.018	0.017	0.018	**0.015**	**0.015**	0.016	0.018	0.017	0.023
			0.099	**0.099**	**0.099**	**0.099**	**0.099**	**0.099**	**0.099**	**0.099**	**0.099**
			3	**4**	3	**4**	**4**	3	3	**4**	3
4	6/3	0.3	**0.004**	0.005	0.0042	**0.004**	0.005	**0.004**	**0.004**	0.005	**0.004**
			0.038	**0.038**	**0.038**	**0.038**	**0.038**	**0.038**	**0.038**	**0.038**	**0.038**
			8	7	7	**8**	7	7	7	7	**8**
5		0.5	0.035	**0.034**	**0.034**	**0.034**	0.035	**0.034**	**0.034**	0.037	**0.034**
			0.105	**0.105**	**0.105**	**0.105**	**0.105**	**0.105**	**0.105**	**0.105**	**0.105**
			3	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
6		0.7	0.030	0.048	0.028	0.049	0.032	0.030	0.031	0.030	**0.012**
			0.104	0.104	0.092	0.104	0.104	0.082	0.104	0.104	**0.081**
			2	2	**3**	2	2	**3**	2	2	**3**
7	7/2	0.5	0.034	0.033	0.034	0.034	0.033	0.034	0.034	0.034	**0.026**
			0.124	**0.124**	**0.124**	**0.124**	**0.124**	**0.124**	**0.124**	**0.124**	**0.124**
			1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
8	7/3	0.5	0.021	0.037	0.038	0.021	0.021	0.021	0.021	**0.015**	0.031
			0.124	0.125	0.125	**0.124**	**0.124**	**0.124**	**0.124**	**0.124**	**0.124**
			2	2	2	2	2	2	2	**3**	2

Even if it is difficult to indicate the best heuristic for the considered problem, we propose the following recommendation. Use *Nodord* algorithm in variant 1, 2 and 3, for each instance of the problem. From among three orderings obtained choose the one giving the smallest schedule length. As it is ease to notice in Tables 2, 3, 4, such a combining effect of the three heuristics guarantees the best values of algorithm metrics for each problem size and combination of parameters P/E.

7 Summary

In this paper the problem of power-aware scheduling of preemptable jobs on parallel identical processors to minimize the schedule length was studied. Two job processing models: one based on a power vs. processor speed function, and the second based on a job processing speed vs. temporary amount of power function have been compared. In the sequel the second, more general model, has been applied to study basic properties of optimal schedules for independent and precedence-related jobs. Methods for finding optimal schedules using these properties have been presented. Although these methods, based on solving convex programming problems, can be practically solved for rather small problem sizes, they are important from the viewpoint of evaluating heuristic vs. optimal solutions. Heuristics for finding ordering of nodes in a task-on-arc digraph which led to possibly short schedules from among all possible orderings in the general case of precedence constraints among jobs have been proposed and tested experimentally.

Further research can be directed towards the construction of heuristics for finding schedules for independent jobs, as well as for dependent ones for a given ordering of nodes. Some concepts of the heuristics proposed in Józefowska et al. (2002) for the case of a nonpreemptive scheduling of independent tasks seem to be a good starting point for such interesting study. Combinations of similar heuristics with the ones presented in this paper, tested experimentally, should lead for an efficient finding of good quality power-aware schedules in the general case. Let us underline that situation where the non-linear solver is substituted by a good heuristic may also be profitable in combination with a metaheuristic approach effectively exploiting dependencies among jobs.

Acknowledgements We are grateful to anonymous referees for their helpful comments. The research was partially supported by a grant from the Ministry of Science and Higher Education, Poland. The computational resources for the experiment were granted by Poznan Supercomputing and Networking Center.

References

Bansal, N., Kimbrel, T., & Pruhs, K. (2005). Dynamic speed scaling to manage energy and temperature. In *Lecture notes in computer science* (Vol. 3404, pp. 460–471).

Bansal, N., Chan, H. L., Lam, T. W., & Lee, L. K. (2008). Scheduling for speed bounded processors. In *Proceedings of international colloquium on automata, languages and programming (ICALP)* (pp. 409–420).

Boyer, F. R., Epassa, H. G., & Savaria, Y. (2006). Embedded power-aware cycle by cycle variable speed processor. In *IEE Proceedings: Vol. 153/4. Computers and digital techniques* (pp. 283–290).

Błażewicz, J., Ecker, K., Pesch, E., Schmidt, G., & Węglarz, J. (2007). *Handbook on scheduling*. Berlin: Springer.

Bunde, D. P. (2006). Power-aware scheduling for makespan and flow. In *Proceedings of the eighteenth annual ACM symposium on parallelism in algorithms and architectures* (pp. 190–196). Cambridge, MA, USA.

Er, M. C. (1985). Lexicographic ordering, ranking and unranking of combinations. *International Journal of Computer Mathematics*, *3*, 277–283.

Irani, S., & Pruhs, K.b (2005). Algorithmic problems in power management. *SIGACT News*, *36*(2), 63–76.

Józefowska, J., Mika, M., Różycki, R., Waligóra, G., & Węglarz, J. (2002). A heuristic approach to allocating the continuous resource in discrete-continuous scheduling problems to minimize the makespan. *Journal of Scheduling*, *5*(6), 487–499.

Józefowska, J., Mika, M., Różycki, R., Waligóra, G., & Węglarz, J. (2004). An almost optimal heuristic for preemptive $C_{\max}$ scheduling of dependent tasks on identical parallel processors. *Annals of Operation Research*, *129*, 205–216.

Lawrence, C., Zhou, J. L., & Tits, A. L. User's guide for CFSQP version 2.5: A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints (1997). Available at: http://www.isr.umd.edu/Labs/CACSE/FSQP/c_manual.ps.

Pruhs, K., van Stee, R., & Uthaisombut, P. (2005). Speed scaling of tasks with precedence constraints. In *Proceedings of WAOA* (pp. 307–319).

Słowiński, R. (1978). A node ordering heuristic for network scheduling under resource constraints. *Foundations of Control Engineering*, *3*, 19–27.

Węglarz, J. (1981). Project scheduling with continuously-divisible doubly constrained resources. *Management Science*, *27*(9), 1040–1053.