Theoretical Computer Science 819 (2020) 24–34

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Flow shop for dual CPUs in dynamic voltage scaling ☆

Vincent Chau [a,c], Xin Chen [b,*], Ken C.K. Fong [c], Minming Li [c], Kai Wang [c]

[a] Shenzhen Institutes of Advanced Technology, Shenzhen, China
[b] School of Electronics and Information Engineering, Liaoning University of Technology, China
[c] Department of Computer Science, City University of Hong Kong, Hong Kong

ARTICLE INFO

Article history:
Received 20 March 2017
Received in revised form 12 September 2017
Accepted 17 September 2017
Available online 22 September 2017

Keywords:
Flowshop
Speed scaling
Scheduling

ABSTRACT

We study the following flow shop scheduling problem on two processors. We are given n jobs with a common deadline D, where each job j has workload $p_{i,j}$ on processor i and a set of processors which can vary their speed dynamically. Job j can be executed on the second processor if the execution of job j is completed on the first processor. Our objective is to find a feasible schedule such that all jobs are completed by the common deadline D with minimized energy consumption. For this model, we present a linear program for the discrete speed case, where the processor can only run at specific speeds in $S = \{s_1, s_2, \cdots, s_q\}$ and the job execution order is fixed. We also provide a $m^{\alpha-1}$-approximation algorithm for the arbitrary order case and for continuous speed model where m is the number of processors and α is a parameter of the processor.

We then introduce a new variant of flow shop scheduling problem called sense-and-aggregate model motivated by data aggregation in wireless sensor networks where the base station needs to receive data from sensors and then compute a single aggregate result. In this model, the first processor will receive unit size data from sensors and the second processor is responsible for calculating the aggregate result. The second processor can decide when to aggregate and the workload that needs to be done to aggregate x data will be $f(x)$ and another unit size data will be generated as the result of the partial aggregation which will then be used in the next round aggregation. Our objective is to find a schedule such that all data are received and aggregated by the deadline with minimum energy consumption. We present an $O(n^5)$ dynamic programming algorithm when $f(x) = x$ and a greedy algorithm when $f(x) = x - 1$.

Finally, we investigate the performance of the flowshop problem when the order of jobs is fixed by comparing it to the approximation algorithm with an arbitrary order. We show experimentally that the approximation ratio is close to 1 when there are few machines and when there are more jobs.

1. Introduction

Energy efficiency has become a major concern nowadays, especially when more and more data centers are used. One of the technologies used to save energy is speed scaling where the processors are capable of varying speed dynamically. The

☆ This work was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 117913].

* Corresponding author.
E-mail address: chenxin@lnut.edu.cn (X. Chen).

https://doi.org/10.1016/j.tcs.2017.09.014
0304-3975/

faster the processors run, the more energy they consume. The idea is to complete all the given jobs before their deadlines with the slowest possible speed to minimize the energy usage.

The energy minimization problem of scheduling n jobs with release times and deadlines on a single processor that can vary its speed dynamically where preemption is allowed has been first studied in the seminal paper by Yao et al. [13]. The time complexity has been improved in [8], [5] and [9]. More works along this line can be found in the surveys by Albers [1] and [2].

On the other hand, researchers have also studied various shop scheduling problems on multiple processors. For example, in the classical flow shop scheduling on m processors, there exist dependencies on execution order between processors for the same job, such that for each job j, it has to be completed on processor i before being executed on processor $i+1$ where $1 \leq i \leq m-1$. This problem is known to turn NP-hard by adding constraints [3]. One of the polynomial cases is when there are two processors and one aims to minimize the makespan, i.e. the completion time of the last job, which can be solved by a greedy algorithm and is proposed by Johnson [6].

Recently, Mu and Li [10] combined the classical flow shop scheduling problem with the speed scaling model. They proposed a polynomial time algorithm for the flow shop speed scaling problem for two processors when the order of jobs is fixed. They also showed that the order returned by Johnson's rule [6] is not optimal for the speed scaling environment. Fang et al. [4] studied the flow shop scheduling problem aiming to find an optimal order of jobs with a restriction on peak power consumption, i.e. the energy consumption is bounded by a constant for a fixed size interval at any time in the schedule.

Our contributions. In this paper, we first extend the study of combining the classical flow shop scheduling problem and the speed scaling model proposed by Mu and Li [10] to the discrete speed setting where the processor can only run at specific speeds in $S = \{s_1, s_2, \cdots, s_q\}$ with predefined job execution order. We present a linear program for this case and also give a simple $m^{\alpha-1}$-approximation algorithm for the general case of flow shop scheduling problem where job execution order is not given.

We then introduce a new variant of flow shop scheduling problem called sense-and-aggregate model motivated by data aggregation in wireless sensor networks where the base station needs to receive data from sensors and then compute a single aggregate result like minimum, maximum or average value of all the input. When dealing with these kinds of aggregation functions, we can either wait for all the input data to be ready, or instead of waiting for all the input data, we can collect part of the data, and compute the sub results. Eventually, we can compute the final aggregation result based on the sub results that we computed previously. The second approach will play a significant role in energy efficient computation.

In the sense-and-aggregate model, we simulate the aggregation calculation in the setting of two processors, where the first processor performs the data collection task, and the second processor is responsible for the calculations. Note that both processors can change speeds when executing workloads. We are given a set of n unit jobs in the first processor and the workload-consideration-function $f(x)$ to indicate the amount of workload that the second processor will process to aggregate x units of data and a deadline D. After each aggregation, one unit of data will be produced which will be used in the next partial aggregation. There is a trade off between waiting and calculating. The more we wait, the more we have to schedule faster (thus consuming more energy), but the less we have to schedule for some function f. Based on this trade off, our objective is to find a schedule with minimum energy consumption to aggregate all the data by the deadline D. For this model, we present an $O(n^5)$ dynamic programming algorithm when $f(x) = x$ and a greedy algorithm when $f(x) = x - 1$.

Finally, we investigate the performance of the flowshop problem when the order of jobs is fixed by comparing it to the approximation algorithm with an arbitrary order. We propose a linear program that achieves a $(1+\varepsilon)$-approximation when it is compared to the convex program. The size of the linear program depends on the accuracy of the solution. We show experimentally that the approximation ratio still depends on the number of machines and on the parameter of the processors α.

The remainder of this paper is organized as follows. In Section 2, we present the formulation of the standard flow shop speed scaling problem. In Section 3, we propose the solution for the flow shop problem with discrete speed scaling model where the job order is predefined. In Section 4, we present a lower bound and an approximation algorithm for the general setting of the flow shop speed scaling problem. In Section 5, we define the sense-and-aggregate model and present a greedy algorithm to solve the aggregation model when $f(x) = x - 1$, together with a dynamic programming approach when $f(x) = x$. In Section 6, we analyze the performance of the solution given by the linear program.

2. Formulation of standard flow shop with speed scaling

We are given m processors and n jobs with a common deadline D, where each job j has workload $p_{i,j}$ on processor i. Each job has to be completed on processor i before it can be performed on the next processor $i+1$. The energy consumption depends on the processor speed. If $s(t)$ is the speed of the processor at time t, then the energy consumption of the processor in the interval $[u, u']$ is $\int_u^{u'} P(s(t))dt$ where $P(s(t)) = s(t)^\alpha$ is the power function of the processor and α is a parameter of the processor, e.g. 1.11 for Intel PXA 270, 1.62 for Pentium M770 and 1.66 for a TCP offload engine [12]. We only assume that $\alpha > 1$ in order to keep the convexity of the power function P. The objective is to find a schedule (when to execute

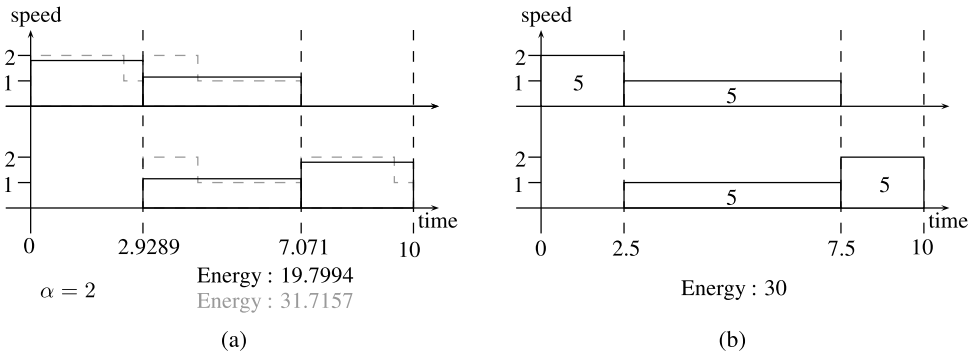

Fig. 1. Example of instance where the discrete optimal schedule cannot be obtained by first computing the continuous optimal schedule.

$$\min \quad \sum_{v\in S}\sum_{i}\sum_{j} v^{\alpha-1}x_{i,j,v} \tag{1}$$

$$\text{s.t.} \sum_{v\in S} x_{i,j,v} = p_{i,j} \qquad \forall i = 1,2,\ldots,m,\ \forall j = 1,\ldots,n \tag{2}$$

$$s_{i,j} + \sum_{v\in S} \frac{x_{i,j,v}}{v} = c_{i,j} \qquad \forall i, j \tag{3}$$

$$c_{m,n} \le D \tag{4}$$

$$c_{i,j} \le s_{i,j+1} \qquad \forall i = 1,\ldots,m,\ \forall j = 1,\ldots,n-1 \tag{5}$$

$$c_{i,j} \le s_{i+1,j} \qquad \forall i = 1,\ldots,m-1,\ \forall j = 1,\ldots,n \tag{6}$$

$$x_{i,j,v},\ s_{i,j},\ c_{i,j} \ge 0 \qquad \forall i, j, v \tag{7}$$

Fig. 2. Linear program for discrete speed setting flow shop.

what job at which speed) to finish all the jobs by the deadline D conforming to the flow shop requirement to minimize the total energy consumption of all the processors.

Given a schedule, we define *critical jobs* to be the jobs that are executed on the second processor as soon as they are completed on the first processor. Given a list of *critical jobs* $\{c_1, c_2, \ldots, c_k\}$ the total cost can be computed as follows [10]: $\frac{\left(p_{1,1}+\sum_{i=1}^{k-1}\sqrt[\alpha]{\left(\sum_{j=1+c_i}^{c_{i+1}} p_{1,j}\right)^{\alpha}+\left(\sum_{j=c_i}^{-1+c_{i+1}} p_{2,j}\right)^{\alpha}}+p_{2,n}\right)^{\alpha}}{D}$.

Note that jobs between two critical jobs $\{c_i + 1, c_i + 2, \ldots, c_{i+1}\}$ are executed at the same speed on the same machine.

3. Discrete speed setting with given job order

In this section, we have a fixed order of jobs that have to be executed on m processors.

Moreover, processors can only run at specific speeds in $S = \{s_1, s_2, \ldots, s_q\}$. The goal is to find a schedule with minimum energy to finish all jobs by the deadline D. The discrete setting of the classical speed scaling problem was studied in [7]. They solved the problem by first computing an optimal schedule in the continuous setting, and then perform the transformation to the solution in the discrete setting by adjusting the speeds. However, in the flow shop speed scaling problem, such a transformation technique cannot be used to obtain an optimal schedule. The example below with two processors demonstrates that computing the optimal schedule in the continuous setting by using the algorithm proposed by Mu and Li [10] may not be the optimal schedule in the discrete speed setting after the transformation.

The instance contains two jobs with $p_{i,j} = 5\ \forall i, j$. Moreover, we set $\alpha = 2$ and $D = 10$. In the discrete setting, we have the set of speeds $S = \{1, 2\}$. According to Fig. 1(a), we use the algorithm proposed by Mu and Li [10] to compute the result schedule with continuous speed, and convert the schedule to discrete speed setting with energy consumption of 31.7157. However, the optimal schedule only consumes 30 as illustrated in Fig. 1(b). This shows that adjusting the speeds obtained from the algorithm proposed by [10] to the adjacent available discrete speeds may not be an optimal solution in the discrete speed setting.

Therefore, it is necessary to design a new method to calculate the optimal solution for the discrete speed setting. We propose the following linear program in Fig. 2. Without loss of generality, we assume that jobs are scheduled in the following order $1, \ldots, n$.

Let $x_{i,j,v}$ be the workload done for job j on processor i at speed v. Let $s_{i,j}$ (resp. $c_{i,j}$) be the starting time (resp. completion time) of job j on processor i. The constraints (2) are to ensure that all jobs are fully executed while the constraints (3), (4), (5), (6) ensure that deadlines are not violated. Especially, since the execution order is fixed, for each

processor, jobs need to be scheduled in the order $1, \ldots, n$. Constraints (5) ensure this order. Job j needs to be completed before job $j+1$ starts. Similarly, constraints (6) ensure that a job j must to be finished on processor i before starting on the next processor $i+1$. Constraints (3) ensure that the completion time of job j on processor i is exactly after its processing time $\sum_{v \in S} \frac{x_{i,j,v}}{v}$ from its starting time $s_{i,j}$. This linear program is correct as we can obtain a feasible schedule in the given order of jobs with deadlines satisfied and jobs scheduled entirely. Given the solution of the linear program, we know exactly the processing time of each job on each processor, i.e. the processing time of job j on processor i is $\sum_{v \in S} x_{i,j,v}$. Then we simply schedule jobs on the first processor with the corresponding processing time. Each job j on the second processor is scheduled either immediately after the previous job $j-1$, or exactly at the time it is completed on the first processor. The constraint (4) guarantees that the returned scheduled has a makespan at most D.

Theorem 1. *The linear program in Fig. 2 has a running time of $O((qnm)^{2.5}L)$ where q is the number of allowed speeds, n is the number of jobs, m is the number of processors, with a precision of $O(L)$ bits.*

Proof. According to the work of Vaidya [11], the running time of a linear program is $O((N+M)^{1.5}NL)$ where N is the number of variables, M is the number of constraints and with a precision of $O(L)$ bits. In the linear program in Fig. 2, we have $O(qnm)$ variables and $O(nm)$ constraints. Thus, the running time of this linear program is $O((qnm)^{2.5}L)$. □

4. Continuous speed setting with arbitrary order

In this section, we investigate the flow shop scheduling problem where the execution order of jobs is not predefined.

Theorem 2. *[10] Computing the minimum energy feasible schedule for the flow shop with arbitrary speeds given the order of execution of jobs on two processors is polynomial.*

We propose a linear time approximation algorithm for the flowshop scheduling problem for an arbitrary number of processors.

Let $V_i = \sum_{j=1}^{n} p_{i,j}$ be the processing volume on the i-th processor.

Proposition 1. $\sum_i (V_i)^\alpha D^{1-\alpha}$ *is a valid lower bound.*

Proof. By omitting the precedence constraints, we can schedule jobs with uniform speed. Since the workload is fixed on each processor, the minimum energy for processor i is $V_i^\alpha D^{1-\alpha}$. □

Proposition 2. $\sum_i (V_i)^\alpha D^{1-\alpha} \geq m(\sum_i V_i/m)^\alpha D^{1-\alpha}$.

Proof. Since the power function is convex, the minimum energy is given by executing the same workload on all processors. □

The main idea of the algorithm is to schedule jobs at speed $\frac{\sum_i V_i}{D}$ in any order. In fact, it is sufficient to schedule jobs on the first processor, then schedule jobs on the next processor once the last job ends on the previous processor. In other words, we schedule jobs in arbitrary order on each processor i in $[\sum_{j=0}^{i-1} V_j, V_i + \sum_{j=0}^{i-1} V_j]$. In this way, all precedence constraints are satisfied and no job misses its deadline. Therefore it is a feasible schedule. The energy consumption of this schedule is exactly $(\sum_i V_i)^\alpha D^{1-\alpha}$.

Algorithm 1 Approximation algorithm for the flowshop speed scaling problem.

```
t ← 0
s ← (Σ_i Σ_j p_{i,j}) / D
for i = 1, ..., m do
    for j = 1, ..., n do
        Schedule job j of machine i at speed s time t to t + p_{i,j}/s
        t ← t + p_{i,j}/s
    end for
end for
```

Theorem 3. *Algorithm 1 has an approximation ratio of $m^{\alpha-1}$ and has a running time of $O(nm)$.*

Proof. We analyze the approximation ratio by dividing the energy cost of the returned schedule by the lower bound.

$$\frac{(\sum_i V_i)^\alpha D^{1-\alpha}}{m(\sum_i V_i/m)^\alpha D^{1-\alpha}} = \frac{(\sum_i V_i)^\alpha}{m(\sum_i V_i/m)^\alpha} = \frac{(\sum_i V_i)^\alpha}{(\sum_i V_i)^\alpha m(1/m)^\alpha} = m^{\alpha-1}.$$

With the discussion above, we can get a feasible schedule in linear time. □

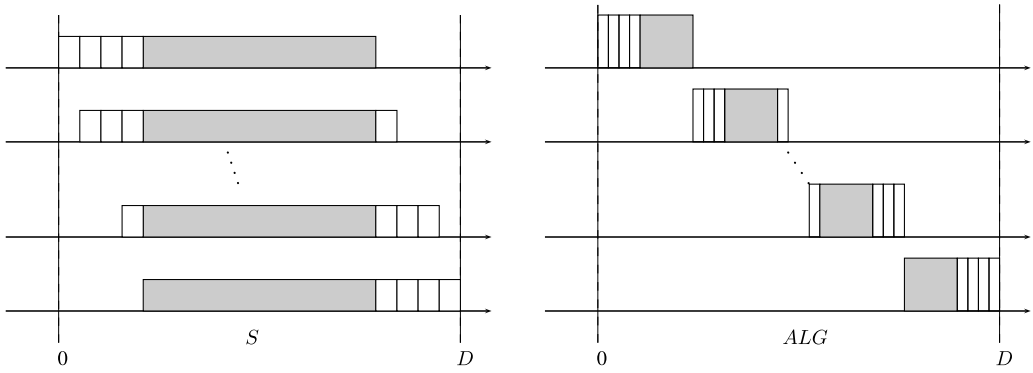

Fig. 3. Tight Example: first job has workload $(1, \ldots, 1, K)$, and the last job has workload $(K, 1, \ldots, 1)$. The gray block represents a workload of K of each job.

Proposition 3. *The bound of the algorithm is tight for m processors.*

Proof. We create an instance with m jobs and m processors. The first job has workload $(1, \ldots, 1, K)$. The second job has workload $(1, \ldots, 1, K, 1)$ and so on. The last job has workload $(K, 1, \ldots, 1)$, with $K > 0$.

Let S be a schedule such that jobs are scheduled in the following order $1, 2, \ldots, n$. We formulate the cost of this schedule such that each job is a critical job. Note that this cost may be larger than the optimal cost. Since we have $m^{\alpha-1} \geq \frac{ALG}{OPT} \geq \frac{ALG}{S}$. We only need to show that $\frac{ALG}{S}$ is at least $m^{\alpha-1}$.

The energy consumption of the schedule S (shown in Fig. 3) is

$$\begin{aligned} S &= (\sqrt[\alpha]{1} + \sqrt[\alpha]{2} + \ldots + \sqrt[\alpha]{m-1} + \sqrt[\alpha]{mK^\alpha} + \sqrt[\alpha]{m-1} + \ldots + \sqrt[\alpha]{1})^\alpha D^{-1} \\ &= \left(2\sum_{i=1}^{m-1} \sqrt[\alpha]{i} + \sqrt[\alpha]{mK^\alpha}\right)^\alpha D^{-1} \end{aligned}$$

while the energy usage in our algorithm is $ALG = (m(m-1+K))^\alpha D^{-1}$.

$$\frac{ALG}{S} = \frac{(m(m-1+K))^\alpha D^{-1}}{\left(2\sum_{i=1}^{m-1} \sqrt[\alpha]{i} + \sqrt[\alpha]{mK^\alpha}\right)^\alpha D^{-1}} = \left(\frac{m(m-1+K)}{2\sum_{i=1}^{m-1} \sqrt[\alpha]{i} + \sqrt[\alpha]{mK^\alpha}}\right)^\alpha$$

Then

$$\lim_{K\to+\infty} \left(\frac{m(m-1+K)}{2\sum_{i=1}^{m-1} \sqrt[\alpha]{i} + \sqrt[\alpha]{mK^\alpha}}\right)^\alpha = \lim_{K\to+\infty} \left(\frac{m(\frac{m-1}{K}+1)}{\frac{2\sum_{i=1}^{m-1} \sqrt[\alpha]{i}}{K} + \sqrt[\alpha]{m}}\right)^\alpha = m^{\alpha-1}$$

Finally, we have $m^{\alpha-1} \geq \frac{ALG}{OPT} \geq \frac{ALG}{S} \geq m^{\alpha-1} - \varepsilon$ for small $\varepsilon > 0$ since the limit is asymptotic. Thus, the analysis is tight. □

5. Sense-and-aggregate model

In this section, we present a new variant of the flow shop speed scaling problem denoted as sense-and-aggregation model which is motivated by data aggregation and computation tasks in wireless sensor networks.

In wireless sensor networks, the station needs to receive data from the sensors and compute the single aggregation result like minimum, maximum, or average of all inputs. For these kinds of aggregation functions, we have two ways to deal with it. We can either wait for all inputs to be ready or we can collect part of the input data and compute the sub results. Eventually, we can compute the final result based on the sub results we computed previously. We introduce the sense-and-aggregation model to model the second approach.

In the sense-and-aggregation model, we are given a set of n unit jobs and two processors where the first processor handles the data collection and aggregation tasks, and the second processor is responsible for the calculations. All jobs need to be processed in the first processor before they can aggregate to the second processor for computations. Moreover, the speeds of the jobs may vary independently and all jobs need to be completed before their common deadline D.

Once the jobs are completed in the first processor, they can be aggregated to the second processor for calculation. For this aggregation process, we need to make use of the workload-consideration-function $f(x)$ to indicate the amount of workload that the second processor will process to aggregate x units of data. Then based on the workload-consideration-function, we can compute how fast we need to schedule the jobs in order to complete all jobs before the deadline D. The main idea of

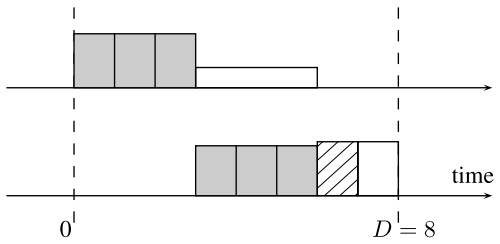

Fig. 4. Optimal schedule of 4 jobs for sense-and-aggregate model with $f(x) = x$.

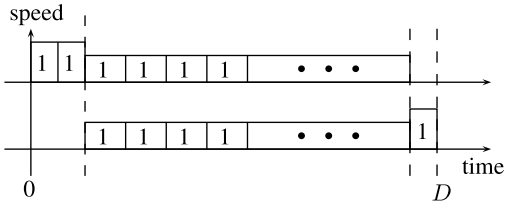

Fig. 5. Optimal schedule when the workload-consideration-function is $f(x) = x - 1$.

performing partial aggregation is to compute the sub results in order to spread the computation process. After each partial aggregation, one unit of data will be produced for the sub result which will be used in the next partial aggregation.

Observe that there is a trade off between waiting and calculating. The more we wait, the more we have to schedule faster (thus consuming more energy), but the less we have to schedule for some function f. Our objective is to find a feasible schedule such that all jobs are completed before their common deadline D in both processors with the minimum usage of energy.

Without loss of generality, we assume that each aggregation on the second processor has at least two units of workload.

In the example showed in Fig. 4, we are going to find out the optimal schedule of 4 jobs with $f(x) = x$. We decide to collect 3 unit jobs on the first processor before scheduling them on the second processor (in gray). Then we schedule the last job on the first processor and the second processor aggregates it with the output data from the previous aggregation, a total workload of 2 for the second aggregation. The output of the first aggregation is shown hatched.

We first analyze the case $f(x) = x - 1$ and propose a greedy algorithm to schedule the jobs with minimum energy consumption. Then we present a dynamic programming approach to solve the problem when $f(x) = x$.

5.1. $f(x) = x - 1$

For aggregation functions such as sum, min and max, we have $f(x) = x - 1$. We first show that the total workload on the second processor is not affected by the decision of the second processor.

Proposition 4. *The workload on the second processor is $n - 1$.*

Proof. Suppose we have k aggregations with sizes $B_1, B_2, \ldots, B_k$. Then $\sum_{i=1}^{k} B_i = n + k - 1$, since the first $k - 1$ aggregations will generate one unit of data each to be the input of subsequent aggregations. Therefore, $\sum_{i=1}^{k} f(B_i) = \sum_{i=1}^{k} (B_i - 1) = n - 1$. □

The idea is to schedule jobs as soon as possible on the second processor since the total workload on the second processor remains unchanged. We obtain the minimum energy consumption by scheduling jobs in the following way.

We first schedule two unit jobs on the first processor. Then we aggregate these jobs on the second processor and generate one unit workload ($f(2) = 1$). At the same time, we continue to schedule jobs on the first processor for the next aggregation (see Fig. 5). It is easy to see that the precedence constraints are guaranteed and it is a feasible schedule. Note that it is not possible to get another schedule with lower energy consumption. Indeed, it is not possible to schedule on the second processor when the first two jobs are scheduled. Similarly, we cannot schedule any job on the first processor when we schedule the last aggregation. Finally, the minimum energy consumption is $\frac{\left(2+\sqrt[\alpha]{(n-2)^\alpha + (n-2)^\alpha} + 1\right)^\alpha}{D}$ according to [10].

5.2. $f(x) = x$

The idea of the algorithm for this case is based on [10]. Indeed, the calculation of the energy consumption depends on critical jobs. A job is critical when the completion time of a job on the first processor is equal to the

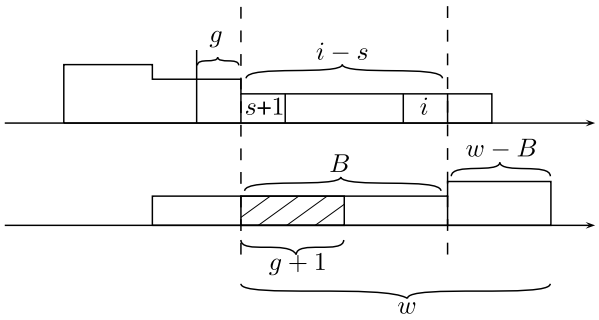

Fig. 6. Illustration of Algorithm 2 and Definition 1.

starting time of the same job on the second processor. Moreover, we only need to focus on schedules in which jobs are executed without preemption (each job is scheduled in one piece) and the execution order is the same on both processors. Recall that given a list of critical jobs $\{c_1, c_2, \ldots, c_k\}$, the total cost can be computed as follows [10]: $\frac{\left(p_{1,1}+\sum_{i=1}^{k-1}\sqrt[\alpha]{\left(\sum_{j=c_i+1}^{c_{i+1}} p_{1,j}\right)^{\alpha}+\left(\sum_{j=c_i}^{c_{i+1}-1} p_{2,j}\right)^{\alpha}}+p_{2,n}\right)^{\alpha}}{D}$.

We adapt the algorithm proposed in [10] for our case. We have to guess the total workload on the second processor since the number of the aggregations is not known in advance. In the following, we suppose that the total workload on the second processor is W. Finally, we show that W is polynomial.

Definition 1. (Fig. 6) Let $F(s, w, g)$ be the minimum cost of the jobs $s+1, \ldots, n$ with a workload of w on the second processor and a pending workload of g on the first processor (before job $s+1$).

Note that the objective function is $\min_{\substack{1\le j\le n\\ 1\le w\le W}} \frac{(F(j+1,w,j))^{\alpha}}{D}$.

Since we aim to find the minimum pending workload on the first processor at the end of each critical interval. In fact, it is equivalent to find the maximum workload that is already processed (aggregated) on the second processor.

Algorithm 2 Flowshop with accumulation.

```
Procedure F(s, w, g)
Min ← +∞
for i = s + 1, ..., n do
    for B = 1, ..., w do
        //test whether s to i can be a segment in the optimal schedule with the minimum pending workload on the first processor
        k ← A(i − s, g, B, B)
        Min = min{Min, ∛((i − s)^α + B^α) + F(i + 1, w − B, (i − s) − k)}
    end for
    return Min
end for
```

Definition 2. $A(i, g, B, e)$ is the maximum workload on the first processor that can be aggregated such that:

- there is at most a workload of i on the first processor
- there is at most a workload of B on the second processor
- There is a pending workload of g (already scheduled on the first processor on a previous critical interval)
- the second processor has already scheduled a workload of e.

We suppose without loss of generality that the schedule starts at time 0 and the pending workload g is before time 0 (see Fig. 7). With the definitions above, we can construct the dynamic program for computing function A.

Proposition 5. *(Fig. 7)*

$$A(i,g,B,e) = \max\begin{cases} A(i,g,B,e-1) \\ \max\limits_{\substack{0\le e'<e \\ \frac{A(i,g,B,e')+(e-e'-1)}{i}\le\frac{e'}{B}}} A(i,g,B,e')+(e-e'-1) \end{cases}$$

$$A(i,g,B,0) = -g$$

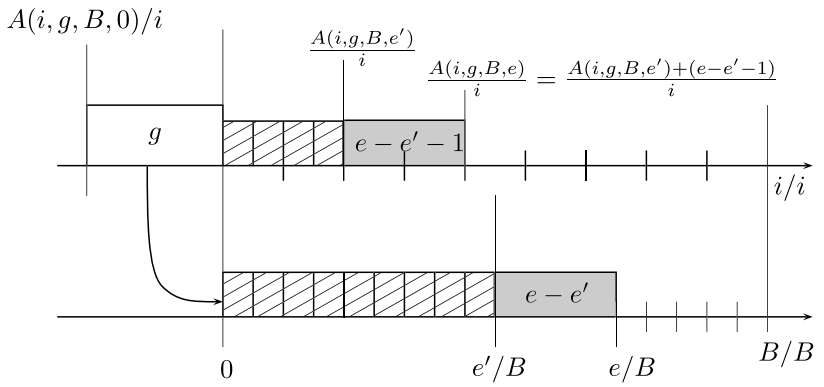

Fig. 7. Illustration of Proposition 5.

Proof. The conditions under the maximization are to ensure that we get a feasible schedule. The initialization means that we have g pending units of workload. Without loss of generality, we assume that the current schedule starts at time 0. Note that when the value of the table A is negative, it means that there is still pending workload that is not transferred to the second processor.

By choosing the value of e', we schedule a workload of $e - e'$ on the second processor in the interval $[e'/B, e/B)$ and we need a workload of $e - e' - 1$ on the first processor. The only condition is to make sure that the workload that we gather on the first processor has to be finished before e'/B.

The first case is when there is no job on the second processor in $[\frac{e-1}{B}, \frac{e}{B})$. It corresponds to the first line of the equation, then the cost is exactly $A(i, g, B, e - 1)$.

Let $A' = A(i, g, B, e') + (e - e' - 1)$ for some e' be the cost of the second case.

We first prove that $A(i, g, B, e) \leq A'$ (Feasibility). Let $\mathcal{S}$ be a schedule (on both processors) that realizes $A(i, g, B, e')$ for some e'. Let e be a value such that $(A(i, g, B, e') + (e - e' - 1))/i \leq e'/B \leq 1$. We build a schedule from time 0 to $\frac{A(i,g,B,e')+(e-e'-1)}{i}$ on the first processor (note that the last value may be negative and is still feasible) from the schedule $\mathcal{S}$ on the same time interval. On the second processor, we construct from 0 to e'/B with the schedule $\mathcal{S}$ within the same time interval and we schedule the gathered jobs (in gray in Fig. 7) from time e'/B to time e/B. This is a feasible schedule and its cost is exactly $A(i, g, B, e') + (e - e' - 1)$. Thus we have $A(i, g, B, e) \leq A'$.

We then prove that $A(i, g, B, e) \geq A'$ (Optimality). We know that the schedule stops at time e/B on the second processor. The grouped jobs are necessary a integer workload. Therefore, we know that the aggregated job has a workload of $e - e'$. The completion time on the first processor will increase by $\frac{e-e'-1}{i}$. In order to get a feasible schedule, we must have $A'/i \leq e'/B$.

As we try among all feasible schedule such that $0 \leq e' \leq e$, the schedule that maximize the completion time on the first processor is necessary one of them. Thus we have $A(i, g, B, e) \geq A'$. □

Proposition 6. *The flow shop problem with aggregation can be solved in $O(n^3W^2)$ time.*

Proof. We first show that the time complexity of the dynamic program in Proposition 5 is $O(n^3W^2)$. The size of the dynamic program table is $O(n^2W^2)$. The maximization is over the values of e'. Actually, e' can be expressed as $e - f(y)$ for $y = 1, \ldots, n + 1$. Otherwise, there is at least $1/B$ unit time which is idle and we can consider $A(i, j, B, e'')$ instead of $A(i, j, B, e')$ with $e'' > e$. Therefore the running time of the maximization is $O(n)$. Thus, the total running time of the dynamic program is $O(n^3W^2)$.

We can precompute the values of $A(i, g, B, e)$ which requires a running time of $O(n^3W^2)$. In Algorithm 2, the size of the dynamic program table $F(s, w, g)$ is $O(n^2W)$. The minimization is over the values i and B and the number of possibilities for these two variables is $O(nW)$. Since the values of $A(i, g, B, e)$ are precomputed, we do not need to compute the values $A(i, g, B, e)$ at each time. To sum up, the total running time is $O(n^3W^2)$. □

Corollary 1. *When the aggregation function is $f(x) = x$, the time complexity is $O(n^5)$.*

Proof. When the workload-consideration-function is $f(x) = x$, we have $W \leq 2n - 1$. It is easy to see that there is at least a workload of n to schedule on the second processor. The maximum workload that we can create after groupings is $n - 1$. Indeed, each aggregation should contain at least one unit of workload from the first processor and at most one unit of workload from the previous aggregation. Finally, we have a maximum workload of $2n - 1$ on the second processor. □

$$\min \quad \sum_i \sum_j v_{i,j}^{\alpha-1} p_{i,j} \tag{8}$$

$$\text{s.t.} \quad s_{i,j} + \frac{p_{i,j}}{v_{i,j}} = c_{i,j} \qquad \forall i, j \tag{9}$$

$$c_{m,n} \leq D \tag{10}$$

$$c_{i,j} \leq s_{i,j+1} \qquad \forall i = 1, \ldots, m,\ \forall j = 1, \ldots, n-1 \tag{11}$$

$$c_{i,j} \leq s_{i+1,j} \qquad \forall i = 1, \ldots, m-1,\ \forall j = 1, \ldots, n \tag{12}$$

$$v_{i,j},\ s_{i,j},\ c_{i,j} \geq 0 \qquad \forall i, j \tag{13}$$

Fig. 8. Convex program for the flowshop problem in the setting of continuous speed when order of jobs is given.

With the previous proof, we can deduce that W is strongly related to the workload-consideration-function $f(x)$. If there are k aggregations and x_i is the workload of the i-th aggregation, the total workload on the second processor is $W \leq \max_k \sum_{i=1}^{k} f(x_i + 1)$ such that $\sum_i x_i = n$. We can roughly bound this with $\max_k \sum_{i=1}^{k} f(x_i + 1) \leq n(\max_{x \in [0,n]} f(x))$.

Thus we have $W \leq n(\max_{x \in [0,n]} f(x))$.

Corollary 2. *The running time of the flow shop problem with aggregation with a given function $f(x)$ can be solved in $O(n^5(1 + \max_{x \in [0,n]} f(x))^2)$ time.*

6. Experiments

In this section, we investigate the performance of the flowshop problem when the order of jobs is fixed by comparing it to the approximation algorithm with an arbitrary order. In particular, Mu and Li [10] showed that the Johnson rule [6] is not optimal for the two machines flowshop problem in the setting of speed scaling. We aim to evaluate the performance of the schedule when the order of jobs is given compared to the two machines flowshop problem for an arbitrary order. The problem can be formulated as a convex program (Fig. 8) when the order of jobs is given.

Let $v_{i,j}$ be the speed of job j on machine i.

We showed that this convex program can be transformed into a linear program losing a factor of $(1 + \varepsilon)$ in the cost.

Lemma 1. *There exists a linear program that is a $(1 + \varepsilon)$-approximation for the flowshop problem with speed scaling when the order of jobs is fixed.*

Proof. The idea is to transform the continuous speed model to the discrete speed model. We discretize the speed in the following way. Let s_{LB} (resp. s_{UB}) be the lower bound (resp. upper bound) of the speed. We get the following set of speeds: $[s_{LB}, (1+\delta)s_{LB}, (1+\delta)^2 s_{LB}, \ldots, (1+\delta)^k s_{LB}]$ with $k = \min\{i \mid (1+\delta)^i s_{LB} > s_{UB}\}$.

Let s be the optimal speed of a given job such that $(1+\delta)^i s_{LB} < s \leq (1+\delta)^{i+1} s_{LB}$. By scaling the speed s to $(1+\delta)^{i+1} s_{LB}$, the energy consumption increases at most by a factor of $(1+\delta)^{\alpha-1}$. Indeed, the energy consumption of this job is $s^{\alpha-1}w$ in the continuous model where w is the processing requirement of this job. After scaling the speed of this job, the energy consumption can be bounded as follows $s^{\alpha-1}w \leq (1+\delta)^{i+1} s_{LB}^{\alpha-1} w \leq ((1+\delta)s)^{\alpha-1} w = (1+\delta)^{\alpha-1} s^{\alpha-1} w$. By repeating such modifications, we conclude that we get a $(1+\delta)^{\alpha-1}$-approximation schedule.

Moreover, we have a feasible schedule since the adjusted speed are higher than that in the optimal solution (the processing time of jobs are shorter).

Let us consider the linear program in Fig. 2 with the following set of speed $[s_{LB}, (1+\delta)s_{LB}, (1+\delta)^2 s_{LB}, \ldots, (1+\delta)^k s_{LB}]$. We simply define $s_{LB} = \min_i \left(\sum_j \frac{p_{i,j}}{D} \right)$ and $s_{UB} = \sum_i \sum_j \frac{p_{i,j}}{D}$ in order to get a feasible schedule as we showed in Section 4 that jobs do not need to be scheduled with a speed higher than s_{UB} or slower than s_{LB}.

Finally, with δ small enough, i.e. $\delta = \sqrt[\alpha]{1+\varepsilon} - 1$, we get a $(1+\varepsilon)$-approximation. □

We then compare this linear program with the lower bound that we proposed in Section 4. The ratio of the obtained solution and the lower bound gives us the approximation ratio. We compare different settings of the environment, such as different value of $\alpha = [2, 2.5, 3]$, for the number of jobs $n = [100, 200, \ldots, 1000]$ and the number of machines $m = [10, 20, \ldots, 100]$. For each combination of these parameters, we run a set of 100 test cases. Moreover, we set $\delta = 0.1$, i.e. the linear program has an approximation ratio of 1.1^{α}. Recall that the proposed approximation ratio in Section 4 is $m^{\alpha-1}$.

We observe that given a number of machines m, the approximation ratio decreases when the number of jobs n increases (Figs. 9, 10 and 11). Moreover, the approximation ratio increases when the value of α or the number of machines increases. This goes the same way as the approximation ratio we proposed in Section 4.

Finally, the results show that the approximation ratios are much better than Algorithm 1. For example, for 10 machines and for $\alpha = 3$, the approximation ratio of Algorithm 1 is 100, but the approximation ratio obtained by the linear program is less than 1.4 (Fig. 11).

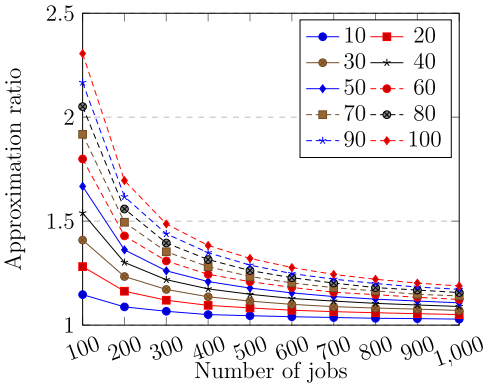

Fig. 9. Approximation ratio for a given order of jobs with $\alpha = 2$ for different number of machines m.

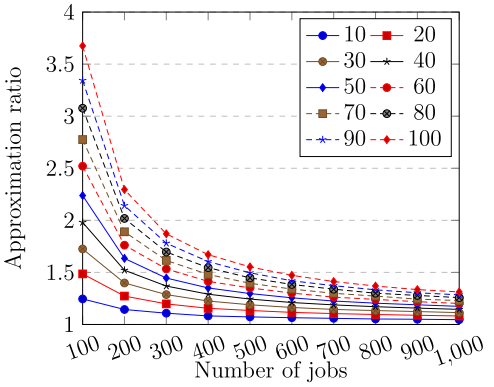

Fig. 10. Approximation ratio for a given order of jobs with $\alpha = 2.5$ for different number of machines m.

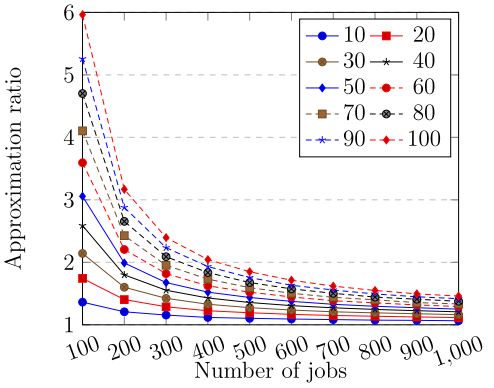

Fig. 11. Approximation ratio for a given order of jobs with $\alpha = 3$ for different number of machines m.

7. Conclusion and future directions

In this work, we proposed a new model called sense-and-aggregate motivated by data aggregation in wireless sensor networks where the base station needs to receive data from sensors and then compute a single aggregate result. Depending on the model, the aggregation function can be different, and we proposed an algorithm whose complexity time depends on the aggregation function. A natural direction is to find a polynomial time algorithm for any aggregation function.

For the speed scaling flowshop problem, the complexity status for two machines still remains open. We proposed a simple greedy algorithm which achieves an approximation ratio of $m^{\alpha-1}$. Finally, we investigate the performance of the schedule in which the order of jobs is given. Since we did not enumerate all the possible orders of execution of jobs, we based our computation on a lower bound. Another direction is to improve this lower bound and thus improve the approximation ratio.

References

[1] S. Albers, Energy-efficient algorithms, Commun. ACM 53 (5) (2010) 86–96, http://doi.acm.org/10.1145/1735223.1735245.
[2] S. Albers, Algorithms for dynamic speed scaling, in: 28th STACS 2011, in: LIPIcs, vol. 9, 2011, pp. 1–11.
[3] P. Brucker, Scheduling Algorithms, vol. 3, Springer, 2007.
[4] K. Fang, N.A. Uhan, F. Zhao, J.W. Sutherland, Flow shop scheduling with peak power consumption constraints, Ann. Oper. Res. 206 (1) (2013) 115–145, http://dx.doi.org/10.1007/s10479-012-1294-z.
[5] B. Gaujal, N. Navet, Dynamic voltage scaling under EDF revisited, Real-Time Syst. 37 (1) (2007) 77–97.
[6] S.M. Johnson, Optimal two-and three-stage production schedules with setup times included, Nav. Res. Logist. Q. 1 (1) (1954) 61–68.
[7] W.-C. Kwon, T. Kim, Optimal voltage allocation techniques for dynamically variable voltage processors, ACM Trans. Embed. Comput. Syst. 4 (1) (2005) 211–230.
[8] M. Li, A.C. Yao, F.F. Yao, Discrete and continuous min-energy schedules for variable voltage processors, Proc. Natl. Acad. Sci. USA 103 (11) (2006) 3983–3987.
[9] M. Li, F.F. Yao, H. Yuan, An $O(nˆ2)$ algorithm for computing optimal continuous voltage schedules, CoRR, 2014, http://arxiv.org/abs/1408.5995.
[10] Z. Mu, M. Li, DVS scheduling in a line or a star network of processors, J. Comb. Optim. 29 (1) (2015) 16–35, http://dx.doi.org/10.1007/s10878-013-9668-y.
[11] P.M. Vaidya, Speeding-up linear programming using fast matrix multiplication, in: 30th Annual Symposium on Foundations of Computer Science, 1989, IEEE, 1989, pp. 332–337.
[12] A. Wierman, L.L.H. Andrew, A. Tang, Power-aware speed scaling in processor sharing systems: optimality and robustness, Perform. Eval. 69 (12) (2012) 601–622, http://dx.doi.org/10.1016/j.peva.2012.07.002.
[13] F.F. Yao, A.J. Demers, S. Shenker, A scheduling model for reduced CPU energy, in: 36th FOCS, IEEE Computer Society, 1995, pp. 374–382.