

***Ordonnancement de chaînes indépendantes sur
processeurs uniformes avec délais de
communication***

Wieslaw Kubiak and Bernard Penz
GILCO, ENSGI-INPG, 46 avenue F. Viallet
38031 Grenoble Cedex 1, France
Denis Trystram
LMC-IMAG, Domaine Universitaire, BP 53
38041 Grenoble cedex, France

No 3576

december 1998

THÈME 1



***rapport
de recherche***

Ordonnancement de chaînes indépendantes sur processeurs uniformes avec délais de communication

Wieslaw Kubiak* and Bernard Penz
GILCO, ENSGI-INPG, 46 avenue F. Viallet
38031 Grenoble Cedex 1, France
Denis Trystram
LMC-IMAG, Domaine Universitaire, BP 53
38041 Grenoble cedex, France

Thème 1 — Réseaux et systèmes
Projet Apache

Rapport de recherche n° 3576 — december 1998 — 24 pages

Résumé : Nous montrons dans ce rapport que le problème qui consiste à ordonnancer un ensemble de chaînes de tâches unitaires indépendantes sur une machine parallèle à processeurs uniformes est NP-difficile au sens fort (en tenant compte ou non des délais de communications). Nous présentons un algorithme linéaire qui calcule une solution optimale pour le cas de 2 processeurs avec délai de communication, lorsqu'un processeur est a fois plus rapide que l'autre (avec a entier). Enfin, nous dérivons une heuristique dans le cas d'un nombre fixé de processeurs avec une bonne garantie de performance.

Mots clés : Ordonnancement - Processeurs Uniformes - Chaînes

(Abstract: pto)

* On leave from the Faculty of Business Administration, Memorial University of Newfoundland, St. John's, Canada

Scheduling Chains on Uniform Processors with Communication Delays

Abstract: We show that the problem of scheduling chains of UET jobs on uniform processors with communication delays to minimize makespan is NP-hard in the strong sense. We also give a heuristic that generates solutions with known, and relatively small, absolute error for this problem. The NP-hardness result holds even for the case without communication delays, and complements earlier result of Gonzalez and Sahni who gave a polynomial time algorithm for preemptive jobs of arbitrary length. We also study the structure of optimal solutions for the two processor problem of scheduling chains of UET jobs with communication delays, where one processor is a (integer) times faster than the other. This investigation leads to a linear time optimization algorithm for this case.

Keywords: Scheduling - Uniform Processors - Chains

1 Introduction

The seminal works dealing with the problem of scheduling jobs on uniform processors, i.e. processors with different speeds, were done in the seventies, see [6] for review, and the eighties, [4], [7] and [9]. There is today a renewed interest in studying such problems because of the popularization of networks of workstations of the same type. These scalable systems may include several micro-processors of different generations, which only differ by their clock speeds (or frequency), or several types of clusters (mono, bi or quadri multi-processors).

The evolution of network architectures has lead to three main architectures: dedicated parallel machines (or supercomputers), homogeneous clusters of processors and general networks of workstations. The first are very expensive, while the third require computing resources on the Web to carry out larger scale computations. We are interested in the second architecture because it realizes a good trade-off between price and performance for large applications. In such an architecture, various implementations exist at the moment. For instance, several components of SMP (Symmetric Multi-Processors) or CCUMA machines (Cache Coherence Uniform Memory Accesses) connected according to a controlled and structured interconnection topology called Systems Area Networks. An example of such a system is the dedicated network built within the *APACHE* project to support the *Athapascan* parallel programming environment [1]. It is composed of one SMP quadri-processor (Pentium 166 Mhz), two SMP bi-processors (Pentium 333 Mhz) and four mono-processors (Pentium 133 Mhz) interconnected by Myrinet and Ethernet 100 Mb/sec.

Therefore, a reasonable model of job scheduling in such systems should capture both the different speeds of processors and the delaying effect that the interconnections between processors may have on communications between jobs. A natural (first level) of approximation is to consider uniform processors (homogeneous processors with homothetic speed ratio) and unit communication delays.

1.1 Problem description and notation

We consider a set of K chains Ch_1, \dots, Ch_K , $K \geq 1$, made up of unit execution time (UET) jobs. Chain Ch_i consists of N_i jobs, $i = 1, \dots, K$. The total number of jobs N equals $\sum_{i=1}^K N_i$. These chains are to be processed on a set P_1, \dots, P_M , $M \geq 2$, of uniform processors so that the completion time of the last job, i.e. schedule makespan C_{max} , is as small as possible. Processor P_j takes a_j units of time to perform a UET job, $j = 1, \dots, M$. We shall assume that all a_j 's are rational numbers. Therefore, without loss of generality we may assume that $a_j = 1$ for some processor j , and $a_j \geq 1$, $j = 1, \dots, M$. Equivalently, we could define the speed v_j of processor P_j as being equal to $\frac{1}{a_j}$, $j = 1, \dots, M$, however it will be more convenient for us to work with the anti-speeds a_j in this paper.

A chain may be switched from one processor to another during its execution if this shortens the execution of the set of chains; consider for example two chains with three UET jobs each and two processors with their anti-speeds 1 and 2. Starting both chains at 0, and then switching their processors at 2 would result in a makespan of 4. Obviously, any solution without processor switching would be longer in this example.

Any switching can only occur between two neighbouring jobs in a chain, in other words no job preemption is allowed in a feasible schedule. If a switch occurs, then a chain completes its job on processor P_i and continues on P_j for some $i \neq j$. The switch requires c_{ij} time units to exchange data and control between the two parts of the chain. We shall call c_{ij} a communication delay, and assume that $c_{ij} = 1$ for all i and j .

The problem will be denoted by $Q|c_{i,j} = 1, chains, p_j = 1|C_{max}$.

1.2 Related Works

The problem $Q | p_j = 1 | C_{max}$ without precedence constraints between jobs can be solved in $O(N \log M)$ time (Lawler et al. [9]) even for the objective functions $\sum_{i=1}^N f_i(C_i)$ and $\max_{i \in \{1, \dots, N\}} f_i(C_i)$ where f_i is a monotone function of the finish time C_i of job i . Gonzalez and Sahni [6] give an $O(N)$ algorithm for problem $Q|pmtn|C_{max}$ with N independent jobs of arbitrary length on M uniform processors.

Despite the fact that there is an $O(N^6)$ -algorithm for problem $Q2 | pmtn, prec, r_j | L_{\max}$, where jobs with arbitrary processing times, release times, and arbitrary precedence constraints are to be processed preemptively on two uniform machines to minimize maximum lateness (Lawler [9]), only few polynomial algorithms have been developed for special precedence constraints. Namely, Kubiak [7] gives a polynomial time algorithm for problem $Q2 | tree, p_j = 1 | C_{\max}$ with one processor a times faster than the other, where a is integer. Gabow [4] tackles the same problem for $a = 1 + 1/k$, where k is an integer. The complexity of the corresponding problem with arbitrary rational a is unknown. Brucker et al [3] give an algorithm for problem $Q2 | chains, p_j = 1 | C_{\max}$ with two uniform processors, UET jobs, chain precedence constraints, and makespan minimization. The algorithm works in $O(K)$ time.

Blazewicz et al [2] give an algorithm for scheduling complete in-trees on two uniform processors with anti-speeds 1 and a (integer) and communication delays.

1.3 Organisation of the paper

In Section 2, we study the computational complexity of problem $Q | c_{i,j} = 1, chains, p_j = 1 | C_{max}$ with arbitrary number of processors and arbitrary anti-speeds. We show that this problem is NP-hard in the strong sense. Actually, we prove a stronger result, namely, that problem $Q | chains, p_j = 1 | C_{max}$ without communication delays is NP-hard in the strong sense. Then, in Section 3, we present an optimization algorithm for two uniform processors, $Q2 | c_{i,j} = 1, chains, p_j = 1 | C_{max}$, with anti-speeds 1 and integer a . In Section 4, we present a heuristic for $Q | c_{i,j} = 1, chains, p_j = 1 | C_{max}$ and prove that it generates solutions within $2M - 1$ units from optimum.

2 The problem with arbitrary number of processors

We first study the complexity of the case with arbitrary number of processors.

Theorem 1 *Problem $Q|chains, p_j = 1|C_{max}$ is NP-hard in the strong sense.*

Proof. The transformation will be from the Numerical 3 Dimensional Matching (N3DM) problem defined as follows (Garey and Johnson [5]).

INPUT: Sets $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_n\}$, and $Z = \{z_1, \dots, z_n\}$. Positive integer weights $s(x_i)$ for each $x_i \in X$, $s(y_j)$ for each $y_j \in Y$, and $s(z_i)$ for each $z_k \in Z$. A positive integer B .

QUESTION: Is there a partition of $X \cup Y \cup Z$ into three element disjoint subsets $\{x_i, y_j, z_k\}$, such that $x_i \in X$, $y_j \in Y$ and $z_k \in Z$, and $s(x_i) + s(y_j) + s(z_k) = B$ in each subset?

This problem remains NP-hard in the strong sense even if the following two conditions are met:

- (1) $s(x_1) \leq s(x_2) \leq \dots \leq s(x_n) < s(y_1) \leq s(y_2) \leq \dots \leq s(y_n)$
- (2) For any two $y_j, y_{j'} \in Y$ and $z_k \in Z$ we have $s(y_j) + s(y_{j'}) + s(z_k) > B$;

Given an instance of N3DM, we define an instance of $Q|chains, p_j = 1|C_{max}$ with $N = nB^6 + 8n + \sum_{i=1}^n f(\bar{x}_i) + \sum_{j=1}^n g(\bar{y}_j)$ jobs, $K = 7n$ chains, and $M = 7n$ processors. All details of this instance are given in Tables 1 and 2. Furthermore, we set the threshold value for makespan equal to $y = B^6$. We have the following two simple propositions about the instance that will be used later in the proof.

Proposition 1 $g(\bar{y}_j) > B^3$, for $j = 1, \dots, n$.

Proof. By definition $g(\bar{y}_j) > y/(g(y_j) - 1) - 3$, for $i = 1, \dots, n$. We need to show that $y > (B^3 + 3)(g(y_j) - 1)$, for $j = 1, \dots, n$. Since $g(y_j) - 1 < B^2 + B$, we only need to show that $y > (B^3 + 3)(B^2 + B)$. It can readily be checked that the last inequality holds for $B \geq 3$ which proves the proposition. ■

Proposition 2 $f(\bar{x}_i) > B^3$, for $i = 1, \dots, n$.

Proof. The proof is similar to that of Proposition 2 and will be omitted. ■

(if part) Consider permutations π and σ of $1, \dots, n$ such that $s(x_{\pi(k)}) + s(y_{\sigma(k)}) + s(z_k) = B$ for $k = 1, \dots, n$. Schedule the first job of chain $x_{\pi(k)}$ to start at 0 on very slow \bar{X} -processor $\pi(k)$ and to continue on fast processor k in

TAB. 1 – Chains

Chains		Number of jobs (N_i)
X-chains	x_1	$f(x_1) = B^2 + s(x_1) + 1$
	.	.
	x_i	$f(x_i) = B^2 + s(x_i) + 1$
	.	.
	x_n	$f(x_n) = B^2 + s(x_n) + 1$
Y-chains	y_1	$g(y_1) = B^2 + s(y_1) + 1$
	.	.
	y_j	$g(y_j) = B^2 + s(y_j) + 1$
	.	.
	y_n	$g(y_n) = B^2 + s(y_n) + 1$
Z-chains	z_1	$y - 2B^2 - B + s(z_1) + 2$
	.	.
	z_k	$y - 2B^2 - B + s(z_k) + 2$
	.	.
	z_n	$y - 2B^2 - B + s(z_n) + 2$
\bar{X} -chains	\bar{x}_1	$f(\bar{x}_1) = \lfloor y/(f(x_1) - 1) \rfloor - 2$
	.	.
	\bar{x}_i	$f(\bar{x}_i) = \lfloor y/(f(x_i) - 1) \rfloor - 2$
	.	.
	\bar{x}_n	$f(\bar{x}_n) = \lfloor y/(f(x_n) - 1) \rfloor - 2$
\bar{Y} -chains	\bar{y}_1	$g(\bar{y}_1) = \lfloor y/(g(y_1) - 1) \rfloor - 2$
	.	.
	\bar{y}_j	$g(\bar{y}_j) = \lfloor y/(g(y_j) - 1) \rfloor - 2$
	.	.
	\bar{y}_n	$g(\bar{y}_n) = \lfloor y/(g(y_n) - 1) \rfloor - 2$
\overline{XX} -chains	\overline{xx}_1	2
	.	.
	\overline{xx}_i	2
	.	.
	\overline{xx}_n	2
\overline{YY} -chains	\overline{yy}_1	2
	.	.
	\overline{yy}_i	2
	.	.
	\overline{yy}_n	2

TAB. 2 – Processors

Processors		Time to do one UET job (a_j)
Very slow \overline{X} processors	1	$y - (f(x_1) - 1)$
	.	.
	i	$y - (f(x_i) - 1)$
	.	.
	n	$y - (f(x_n) - 1)$
Very slow $\overline{X\overline{X}}$ processors	1	$y - (f(x_1) - 1)$
	.	.
	i	$y - (f(x_i) - 1)$
	.	.
	n	$y - (f(x_n) - 1)$
Very slow \overline{Y} processors	1	$y - (g(y_1) - 1)$
	.	.
	j	$y - (g(y_j) - 1)$
	.	.
	n	$y - (g(y_n) - 1)$
Very slow $\overline{Y\overline{Y}}$ processors	1	$y - (g(y_1) - 1)$
	.	.
	j	$y - (g(y_j) - 1)$
	.	.
	n	$y - (g(y_n) - 1)$
Slow X processors	1	$f(x_1) - 1$
	.	.
	i	$f(x_i) - 1$
	.	.
	n	$f(x_n) - 1$
Slow Y processors	1	$g(y_1) - 1$
	.	.
	j	$g(y_j) - 1$
	.	.
	n	$g(y_n) - 1$
Fast processors	1	1
	.	.
	k	1
	.	.
	n	1

$[y - (f(x_{\pi(k)}) - 1), y]$. Schedule chain $y_{\sigma(k)}$ on fast processor k in $[0, g(y_{\sigma(k)}) - 1]$ and its last job to finish at y on very slow \overline{Y} -processor $\sigma(k)$. Schedule the first job of z_k on slow Y -processor $\sigma(k)$ to start at 0, and its last job on slow X -processor $\pi(k)$ to finish at y . Schedule all remaining jobs of z_k on fast processor k in $[g(y_{\sigma(k)}) - 1, y - (f(x_{\pi(k)}) - 1)]$.

Finally, schedule chains $\overline{x}_{\pi(k)}$ and $\overline{y}_{\sigma(k)}$ on slow X -processor $\pi(k)$ and slow Y -processor $\sigma(k)$, respectively, in intervals $[f(x_{\pi(k)}) - 1, y - (f(x_{\pi(k)}) - 1)]$ and $[g(y_{\sigma(k)}) - 1, y - (g(y_{\sigma(k)}) - 1)]$, respectively. This leaves intervals $[0, f(x_{\pi(k)}) - 1]$ on slow X -processor $\pi(k)$ and $[y - (g(y_{\sigma(k)}) - 1), y]$ on slow Y -processor $\sigma(k)$ available for $\overline{X\overline{X}}$ -chain $\pi(k)$ and for $\overline{Y\overline{Y}}$ -chain $\sigma(k)$, respectively. These two chains may continue on very slow $\overline{X\overline{X}}$ -processor $\pi(k)$ and $\overline{Y\overline{Y}}$ -processor $\sigma(k)$, respectively. By repeating these steps for all chains z_k , $k = 1, \dots, n$, we obtain a feasible schedule with makespan equal y .

(only if part) Consider a schedule with $C_{max} \leq y$. We show how to build a required matching. We have the following simple results for the schedule.

Claim 1 *There are at most ny jobs on fast processors.*

Proof. The claim follows immediately from the fact that there are n fast processors and it takes one unit of time to process a UET job on any fast processor. ■

Claim 2 *There are at most $g(\overline{y}_j) + 2$ jobs on slow Y -processor j , $j = 1, \dots, n$.*

Proof. By contradiction. Assume that there are $g(\overline{y}_j) + e$, where $e \geq 3$, jobs on some slow Y -processor j . Then, we have $T_j \geq (g(\overline{y}_j) + e)(g(y_j) - 1)$ for the total load T_j on j . By definition of $g(\overline{y}_j)$ we obtain $T_j \geq y + \epsilon(g(y_j) - 1)$, where $\epsilon > 0$. This, however, leads to a contradiction since $T_j \leq C_{max} \leq y$. ■

Claim 3 *There are at most $f(\overline{x}_i) + 2$ jobs on slow X -processor i , $i = 1, \dots, n$.*

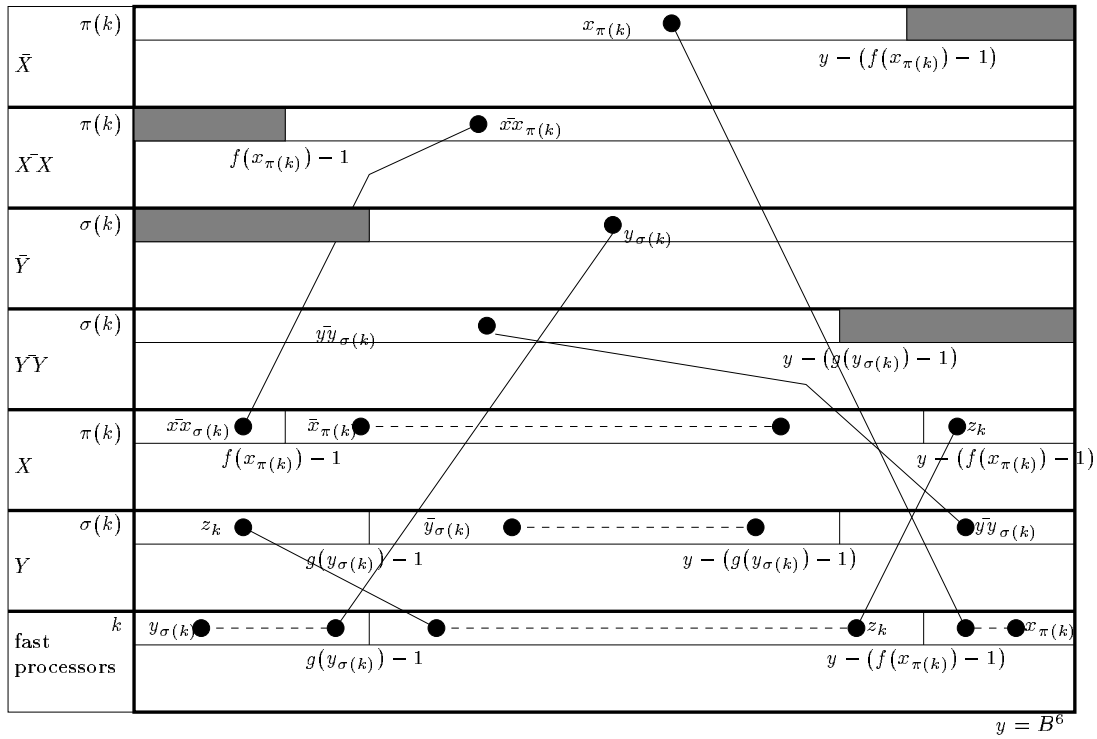


FIG. 1 – Principle of allocation in the optimal schedule.

Proof. The proof is similar to that of Claim 2 and will be omitted. ■

Claim 4 *There is exactly one job on each very slow processor.*

Proof. First, we prove that there is at most one job on each very slow processor. Otherwise, the total load would be at least $2y - 2(f(x_i) - 1)$ for some very slow \overline{X} - or \overline{XX} -processor i , or $2y - 2(g(y_j) - 1)$ for some very slow \overline{Y} - or \overline{YY} -processor j . Either value is greater than y which leads to a contradiction. Second, the fast and slow processors can process at most $\sum_{i=1}^n (f(\overline{x}_i) + 2) + \sum_{j=1}^n (g(\overline{y}_j) + 2) + ny$ jobs, which follows from Claims 1-3. This leaves at least $4n$ jobs for $4n$ very slow processors, thus, the claim holds. ■

Claim 5 *Neither Z - nor \overline{X} - nor \overline{Y} - chains have their jobs on very slow processors.*

Proof. By Claim 4, it is sufficient to show that executing at least one job of either Z - or \overline{X} - or \overline{Y} -chain on a very slow processor would result in a schedule longer than y . This, however, easily follows from Propositions 1 and 2, and the definitions in Table 1. ■

Claim 6 *Each Y -chain has exactly one job on a very slow \overline{Y} - or \overline{YY} - processor, and its remaining jobs on fast processors. Also, each X -chain has exactly one job on a very slow processor, and its remaining jobs on fast processors.*

Proof. First, notice that no chain may have more than one of its jobs on very slow processors, otherwise the chain could not finish by y . On the other hand, by Claims 4 and 5, each X - and Y -chain must have at least one of its jobs on very slow processors. Thus, any X - and Y -chain must have exactly one job on very slow processors. Furthermore, none of the chain's remaining jobs can be on a slow processor in order for the chain to complete by y . Thus, all must be on fast processors.

Finally, no Y -chain may be processed by a slow \overline{X} - or \overline{XX} - processor. Otherwise, the total processing time of some Y -chain, say y_j , would be at least $y - (f(x_i) - 1) + g(y_j) - 1 = y + s(y_j) - s(x_i)$ for some $i = 1, \dots, n$, and y_j would not finish by y since, by (1), $s(y_j) - s(x_i) > 0$. ■

Claim 7 *Exactly two jobs of each Z - chain are executed on slow processors. Furthermore, neither \overline{X} - nor \overline{XX} - nor \overline{Y} - nor \overline{YY} -chain is done on fast processors.*

Proof. First, we show that no Z -chain z_k can afford to execute more than two of its jobs on slow processors. Otherwise, it would take at least $B^6 - 2B^2 - B + s(z_k) + 2 + eB^2 - e$, to complete chain z_k with $e > 2$ jobs on slow processors. This value equals $y + (e - 2)(B^2 - 1) - B + s(z_k)$ and is greater than y since $(e - 2)(B^2 - 1) - B + s(z_k)$ is positive for $e \geq 3$ and $B \geq 3$.

Second, by Claim 6, there are $\sum_{j=1}^n f(x_i) + \sum_{j=1}^n g(y_j) - 2n = 2nB^2 + nB - \sum_{j=1}^n s(z_k) + 2n - 2n$ jobs from X - and Y - chains on fast processors. By Claim 1, this leaves at most $nB^6 - 2nB^2 - nB + \sum_{j=1}^n s(z_k)$ time on fast processors for Z -chains. Thus, at least $2n$ jobs from Z -chains must be processed on slow processors. As we showed at the beginning of the proof, no more than two jobs of any Z - chain can be executed on slow processors, thus, we have that exactly two jobs of each Z - chain are executed on slow processors. Therefore, there are exactly ny jobs from X -, Y - and Z -chains on fast processors, which leaves no room for either \overline{X} - or \overline{XX} - or \overline{Y} - or \overline{YY} -chain to be scheduled on fast processors. This proves the claim. ■

Claim 8 *Each \overline{XX} -chain and \overline{YY} -chain has one of its jobs on a very slow processor and the other on a slow X -or Y -processor.*

Proof. By Claim 4, each very slow processor must process exactly one job. By Claim 5, this job must not belong to Z -, \overline{X} -, or \overline{Y} - chains. By Claim 6, exactly

$2n$ out of $4n$ very slow processors are occupied by jobs from X - and Y -chains. This leaves $2n$ very slow processors for $2n$ \overline{XX} - and \overline{YY} -chains. Thus, each \overline{XX} - and \overline{YY} -chain has one of its jobs on a very slow processor. Furthermore, by Claim 7, the other job must not be on a fast processor. Thus, it must be on either a slow X -processor or a slow Y -processor. ■

Claim 9 *There are at least n \overline{XX} - or \overline{YY} -chains on slow X -processors.*

Proof. By Claim 6 and 8, there are at least n \overline{XX} - or \overline{YY} -chains on very slow \overline{X} - and \overline{Y} -processors. By (1), none of these chains may continue on slow Y -processors to complete by y . Thus all must be on slow X -processors. ■

Claim 10 *Exactly one job of each Z -chain is executed on a slow X -processor and exactly one on a slow Y -processor.*

Proof. A Z -chain can not have two of its jobs processed on slow Y -processor since its processing time would then be at least

$y - 2B^2 - B + s(z_k) + g(y_j) - 1 + g(y_i) - 1 = y - B + s(z_k) + s(y_j) + s(y_i)$ which, by (2), is greater than y .

We now show that each Z -chain has exactly one job on a slow Y -processor. The proof will be by contradiction. Assume that there are $n - e$ Z -chains, $e \geq 1$, with one job on a slow Y -processor and one on a slow X -processor. Thus, by Claim 7, there are e Z -chains with two jobs on slow X -processors.

Let s_i and t_i be jobs processed on slow X -processor i , $i = 1, \dots, n$ in the intervals $[0, f(x_i) - 1]$ and $[y - (f(x_i) - 1), y]$, respectively. We shall call these jobs "ends". We observe that there are $2n$ "ends". By Claim 9, at least n out of the $2n$ "ends" belong to \overline{XX} - or \overline{YY} -chains. Furthermore, at least $n - e$ "ends" belong to the Z -chains with one job on a slow Y -processor. To prove this it is sufficient to show that at least $y - 2B^2 - 2B$ jobs of any Z -chain must be processed on fast processor between the chain's jobs processed on slow processors. First, observe that the processing of all X - and Y -chains on very slow processors overlaps at any moment in the interval $o = [g(y_{max}) - 1, y - (g(y_{max}) - 1)]$. Thus, only Z -chains can be processed on fast processors in

o. Since there are n Z -chains and n fast processors, then no Z -chain can have its job processed on slow processors in o . Therefore, any Z -chain can switch to slow processors in the intervals $[0, g(y_{max}) - 1]$ and $[y - (g(y_{max}) - 1), y]$ only. Furthermore, neither interval can accommodate more than one job on slow processors. Finally, $y - 2(g(y_{max}) - 1) \geq y - 2B^2 - 2B$.

We now proved that there are at most e "ends" that belong to the e Z -chains with two jobs on slow X -processors. Thus, $2e$ jobs from these chains are on slow X -processors, but at most e of them are "ends". Therefore, there is a Z -chain with a job processed on a slow processor in o which leads to a contradiction. Notice that $2(f(x_{min}) - 1) > g(y_{max}) - 1$.

We now proved that $e = 0$ and thus the claim holds. \blacksquare

Consider Z -chain z_k , $k = 1, \dots, n$. By Claim 10, one of its jobs is on a slow X -processor, say $m(k)$, and the other on a slow Y -processor, say $l(k)$. Thus, it takes at least

$$B^6 - 2B^2 - B + s(z_k) + g(y_{m(k)}) - 1 + f(x_{l(k)}) - 1 = B^6 - B + s(z_k) + s(y_{m(k)}) + s(x_{l(k)})$$

units of time to complete z_k . We must have

$$B^6 - B + s(z_k) + s(y_{m(k)}) + s(x_{l(k)}) \leq y$$

or

$$s(z_k) + s(y_{m(k)}) + s(x_{l(k)}) \leq B.$$

Consequently

$$\sum_{k=1}^n s(x_{l(k)}) + \sum_{k=1}^n s(y_{m(k)}) \leq nB - \sum_{k=1}^n s(z_k) = \sum_{i=1}^n s(x_i) + \sum_{j=1}^n s(y_j).$$

Thus, if both l and m are permutations, then we have $s(z_k) + s(y_{m(k)}) + s(x_{l(k)}) = B$ and the proof will be complete. We now show that both l and m are permutations.

Claim 11 *Both l and m are permutations.*

Proof. Let us index all inequalities in (1) from 1 (the leftmost) to $2n - 1$ (the rightmost). Let $i_1 < \dots < i_r$, for $r \geq 1$, be all the " $<$ " inequalities in this order. Define l_i and L_i to be the number of X - and Y -chains, and \overline{XX} - and \overline{YY} -chains, respectively, processed on the $2i$ fastest very slow processors. Obviously, we have $l_i + L_i = 2i$. Also, we have $l_i \geq L_i$, for $i = i_1, \dots, i_r$. Otherwise, $l_j < L_j$ for some $j = i_1, \dots, i_r$, and then, $l_j < j$. However, the longest j X - and Y -chains

can only be scheduled on the $2j$ fastest very slow processors to complete by y . Thus, when $l_j < j$, at least one of these chains is outside of these processors and so completes after y . Therefore, we get a contradiction, which proves that $l_i \geq L_i$, for $i = i_1, \dots, i_r$. If $l_i = L_i$, for $i = i_1, \dots, i_r$, then each slow X - and Y -processor processes exactly one job of either a \overline{XX} -chain or a \overline{YY} -chain. Consequently, no two different Z -chains have jobs on the same slow X - or Y -processor. Thus, both l and m are permutations.

Thus, it remains to prove that we never have $l_j > L_j$, for some $j = i_1, \dots, i_r$. Let A and C be the sets of all very slow \overline{X} -processors with X -chains and \overline{XX} -chains, respectively. Let B and D be the sets of all very slow \overline{Y} -processors with Y -chains and \overline{YY} -chains, respectively. If $l_j > L_j$, for some $j = i_1, \dots, i_r$, then we have

$$\sum_{i \in A} (y - (f(x_i) - 1)) + \sum_{j \in B} (y - (g(y_j) - 1)) < \sum_{i \in C} (y - (f(x_i) - 1)) + \sum_{j \in D} (y - (g(y_j) - 1))$$

or

$$\sum_{i \in C} s(x_i) + \sum_{j \in D} s(y_j) < \sum_{i \in A} s(x_i) + \sum_{j \in B} s(y_j).$$

Since

$$\sum_{i \in C} s(x_i) + \sum_{j \in D} s(y_j) + \sum_{i \in A} s(x_i) + \sum_{j \in B} s(y_j) = 2\left(\sum_{i=1}^n s(x_i) + \sum_{j=1}^n s(y_j)\right)$$

we have

$$\sum_{i \in C} s(x_i) + \sum_{j \in D} s(y_j) < \sum_{i=1}^n s(x_i) + \sum_{j=1}^n s(y_j).$$

Let E and F be the sets of all X - and Y -processors, respectively with with \overline{XX} - or \overline{YY} -chains.

We have

$$\sum_{i \in E} s(x_i) + \sum_{j \in F} s(y_j) + \sum_{k=1}^n s(x_{l(k)}) + \sum_{k=1}^n s(y_{m(k)}) = 2\left(\sum_{i=1}^n s(x_i) + \sum_{j=1}^n s(y_j)\right),$$

and, furthermore,

$$\sum_{i \in E} s(x_i) + \sum_{j \in F} s(y_j) \leq \sum_{i \in C} s(x_i) + \sum_{j \in D} s(y_j)$$

which means

$$\sum_{k=1}^n s(x_{l(k)}) + \sum_{k=1}^n s(y_{m(k)}) > \sum_{i=1}^n s(x_i) + \sum_{j=1}^n s(y_j)$$

and we get a contradiction. Therefore, $l_i = L_i$, for $i = i_1, \dots, i_r$ and the claim holds. \blacksquare

Theorem 2 *Problem $Q|c_{ij} = 1, chains, p_j = 1|C_{max}$ is NP-hard in the strong sense.*

Proof. We show a simple transformation from $Q|chains, p_j = 1|C_{max}$ to $Q|c_{ij} = 1, chains, p_j = 1|C_{max}$. Actually, we restrict ourselves to the instances of $Q|chains, p_j = 1|C_{max}$ constructed in Theorem 1 as they make an NP-hard in the strong sense subproblem of $Q|chains, p_j = 1|C_{max}$. (Therefore, we actually take the Numerical 3 Dimensional Matching problem as a point of departure in our construction.) In these instances, we reduce the anti-speed of each very slow \overline{X} - and \overline{Y} -processor by 1, we do the same for slow X - and Y -processors. These reductions will allow for communications between two parts of each X -, Y -, $\overline{X\overline{X}}$ - and $\overline{Y\overline{Y}}$ -chain, as well as between three parts of each Z -chain. Therefore, it will be easy to obtain a schedule with communication delays which is not longer than y for a given schedule without communication delays not longer than y , see the first part of the proof of Theorem 1. Furthermore, we increase the number of jobs in \overline{X} -chain \overline{x}_i to $\lfloor y/(f(x_i) - 2) \rfloor - 2$ and the number of jobs in \overline{Y} -chain \overline{y}_j to $\lfloor y/(g(y_j) - 2) \rfloor - 2$ to compensate for shorter anti-speeds on slow processors. The threshold value of the makespan will remain equal to y . We observe that after these changes Propositions 1 and 2, as well as Claims 1-10 hold for any schedule with communication delays which is not longer than y . Thus, the first and the last job on each slow processor belongs to either a Z -chain or $\overline{X\overline{X}}$ -chain or $\overline{Y\overline{Y}}$ -chain, and all the remaining jobs on slow processors belong to either extended \overline{X} -chains or extended \overline{Y} -chains. Thus, increasing the anti-speed of each slow processor by 1 and replacing the extended chains by the regular \overline{X} - and \overline{Y} -chains will result in a schedule without communication delays which is not longer than y . ■

3 The two-machines problem with integer anti-speeds

We will present an optimization algorithm for the case of two uniform processors. We consider fast processor, denoted by P_f , and slow processor, denoted by P_s , with anti-speeds 1 and a (integer greater than 1) respectively.

We first assume a number of chains greater than three ($K \geq 3$), and then discuss the case of no more than 2 chains.

3.1 A lower bound

We now establish a lower bound on the makespan.

Proposition 3 *The minimal time to schedule a set of K chains on two uniform processors with anti-speeds 1 and a (integer) is at least: $LB = \lceil \frac{aN}{1+a} \rceil$.*

Proof. We have $N = (a+1)x + r$, for some integer x and $0 \leq r < a+1$. Thus, $LB = ax + r$. On the other hand, we have $Na = (a+1)ax + ar$. Consequently, $\frac{Na}{a+1} = ax + r - \frac{r}{1+a}$ and $\lceil \frac{aN}{1+a} \rceil = ax + r$. Therefore, $LB = \lceil \frac{aN}{a+1} \rceil$. ■

3.2 Scheduling more than 2 chains

We now present an algorithm for scheduling a set of K ($K \geq 3$) chains on two uniform processors. We assume that the chains are sorted by decreasing order of their lengths, in other words $N_1 \geq N_2 \geq \dots \geq N_K > 0$.

Let N_i^s , for $i = 1, \dots, K$, be the largest integer p such that $N_i - p + ap < LB$. Define $N_i^f = N_i - N_i^s$.

Algorithm (A)

```

|   Compute  $LB = \lceil \frac{aN}{1+a} \rceil$ 
| 1   if  $(N_1 \geq LB)$  then
|     Allocate  $Ch_1$  to  $P_f$  and all the remaining chains to  $P_s$ 
|   else
|     if  $N = LB + 1$  then
| 2     if  $N_K = 1$  then
|       Allocate  $Ch_K$  to  $P_s$  and all the remaining chains to  $P_f$ 
| 3     if  $N_K > 1$  and  $(a + N_K \leq LB)$  then
|       Allocate the first job of  $Ch_K$  to  $P_s$ .
|       Allocate chains  $Ch_1, \dots, Ch_{K-1}$  to  $P_f$  to start at 0. Allocate
|       the remaining jobs of chain  $Ch_K$  to  $P_f$  to complete at  $LB$ .

```

```

4   if  $N_K > 1$  and  $(a + N_K > LB)$  then
    Allocate all chains to  $P_f$ .
5   if  $N > LB + 1$  then
    Determine the longest chain with  $N_{i_m}^s > 0$ .
    Allocate the first  $N_{i_m}^f$  tasks of  $Ch_{i_m}$  to  $P_f$  to start at 0.
    Allocate the last  $N_{i_m}^s$  tasks to  $P_s$  to finish at  $LB$ .
    Allocate all the remaining chains in the decreasing order of their
    lengths from left to right to  $P_f$  until it is
    possible to complete them by  $LB$ .
    If a chain, say  $Ch_{i_0}$ ,  $1 \leq i_0 \leq k$ ,  $i_0 \neq i_m$ , needs to be split, then
    the last  $LB - N_{i_m}^f - \sum_{i=1, i \neq i_m}^{i_0-1} N_i$  tasks of  $Ch_{i_0}$  are allocated to  $P_f$ 
    to complete at  $LB$ , while the remaining are allocated to
     $P_s$  to start at 0.
    The remaining chains are allocated to  $P_s$  from left to right
    after  $Ch_{i_0}$ .
End

```

Proposition 4 *Algorithm (A) builds an optimal schedule.*

Proof.

It is easy to check that all disjunctive cases considered in the algorithm represent all possible cases for this scheduling problem.

The condition in line 1 holds. Then no communication occurs, and since $N_1 \geq a(\sum_{i=2}^k N_i)$ the length of chain Ch_1 determines the schedule makespan.

In the three following cases, the shortest schedule is obtained by allocating at most one task to P_s .

The condition in line 2 holds, then the lower bound LB is reached and no communication occurs.

The condition in line 3 holds. Then, the algorithm splits chain Ch_K . Since $K > 2$, then Ch_K does not start before $a + 1$ on P_f . Furthermore, Ch_K has

exactly one job on P_s and this one completes at a , which leaves enough time for communication between the two parts of Ch_K . The lower bound LB is reached.

The condition in line 4 holds. Then $a + (N_K - 1) \geq LB$. This condition means that allocating one task of the shortest chain to P_s will exceed LB . Obviously, then, the best solution is obtained by allocating all chains to P_f and the resulting optimal makespan equals $LB + 1$.

The condition in line 5 is met. Consider the time interval between $N_{i_m}^f$ and $LB - aN_{i_m}^s$. By definition of $N_{i_m}^s$, we have $0 < LB - (aN_{i_m}^s + N_{i_m}^f) = d < a$, which allows for a unit time communication between the two parts of Ch_{i_m} , and makes that the last job, if any, to complete by $LB - aN_{i_m}^s$ on P_s starts by $N_{i_m}^f - 1$. This job must be from Ch_K (see Figure 2). Therefore, if Ch_K is on P_s only, then the chain split between P_s and P_f , if any, does not overlap and at least one unit of time elapses between its P_s part and P_f part which allows for a unit time communication between this two parts. If Ch_K is split between P_s and P_f , then $\alpha > 0$ of its jobs is scheduled on P_s from time 0 and $\beta > 0$ of its jobs is scheduled on P_f to complete at LB . If the two parts did not make a feasible schedule, then $N_k > \beta \geq aN_{i_m}^s > a$. Consequently, $N_i \geq N_k > a$, where Ch_i is the first chain to start just after Ch_{i_m} on P_f . Thus, chain Ch_i would occupy P_f in the interval $[N_{i_m}^f, N_{i_m}^f + a]$, which means that Ch_K could not start earlier than $LB - aN_{i_m}^s + 1$. Then, however the two parts of Ch_K would not overlap and the unit between $N_{i_m}^f + a - 1$ and $N_{i_m}^f + a$ could be used for communication. Therefore, we get a contradiction, and consequently the two parts of Ch_K make a feasible schedule, which proves that the algorithm yields an optimal schedule if condition in line 5 is met. ■

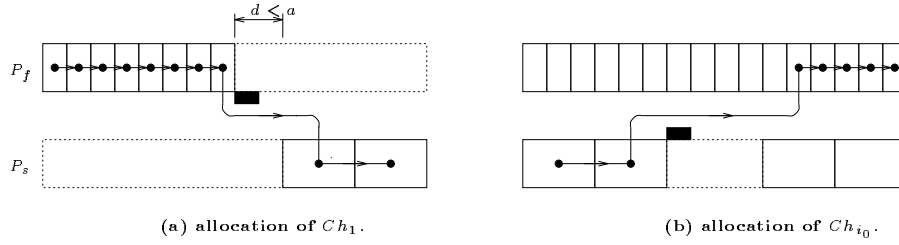


FIG. 2 – Principle of the main case of the algorithm (condition 5).

3.3 Scheduling 1 and 2 chains

Proposition 5 *An optimal solution for scheduling 1 or 2 chains on two uniform processors with anti-speeds of 1 and a can be obtained in constant time.*

Proof.

The proof is constructive. $\mathbf{K=1}$:

Of course, the best strategy is to allocate the only chain to P_f .

$\mathbf{K=2}$:

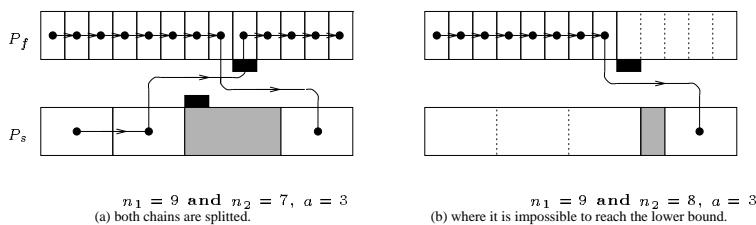
In this case, we have to distinguish three cases:

$N_1 < a$: Then, $\min\{N_1 - 1 + a, N_2 - 1 + a\} = N_2 - 1 + a \geq N_1 + N_2$. Thus, the optimal solution is to schedule all jobs on P_f , which results in a makespan of $N_1 + N_2$.

$N_1 \geq aN_2$: Then the allocation of Ch_1 to P_f and Ch_2 to P_s results in an optimal schedule with makespan of N_1 .

$a \leq N_1 < aN_2$: Allocate the first N_1^f tasks of Ch_1 to P_f to start at 0. Allocate the last N_1^s tasks to P_s to finish at LB . Allocate the jobs of chain Ch_2 to P_f until LB , the remaining jobs of Ch_2 are allocated to P_s to start at 0.

Consider the time interval between N_1^f and $LB - aN_1^s$ in this schedule. By definition of N_1^s , we have $0 < LB - (aN_1^s + N_1^f) < a$. Thus, if Ch_2 completes by $N_1^f - 1$ on P_s , then the schedule is feasible and optimal (all jobs finish by LB , see Figure 3a). If Ch_2 completes at N_1^f or later, then the schedule is not feasible since either Ch_2 overlaps on P_f and P_s or there is no time for communication between different parts of Ch_2 , see Figure 3b. In this infeasible schedule, however, the last job of Ch_2 completes by $LB - aN_1^f$, which follows

FIG. 3 – Example of the algorithm for $K = 2$.

from the definition of LB . Thus, moving one job of Ch_2 from P_s to P_f makes the schedule feasible and increases the makespan from LB to $LB + 1$.

Let N'_2 be the number of jobs from chain Ch_2 on P_f in this schedule. Obviously, there are N_1^f jobs from chain Ch_1 on P_f , and $N_1^f + N'_2 = LB + 1$. By definition of N_1^f , reducing the number of jobs from Ch_1 on P_f would increase the completion time of Ch_1 beyond LB . Since we have $N_2^f \geq N'_2$, then reducing the number of jobs from Ch_2 on P_f would increase the completion time of Ch_2 beyond LB as well. Therefore, the schedule is optimal. ■

4 A simple heuristic and a simple observation

We now show a simple heuristic for the general problem of scheduling UET chains on uniform processors. The heuristic guarantees that its solutions are within $2M - 1$ units from optimum. It is based on an algorithm given by Gonzalez and Sahni [6] for scheduling preemptible jobs of arbitrary length on uniform processors.

Theorem 3 *There is an $O(K \log K)$ heuristic H for $Q|c_{ij} = 1, chain, p_j = 1|C_{max}$ that guarantees an absolute worst case behaviour of $2M - 1$.*

Proof. Let us consider chains Ch_1, \dots, Ch_K as preemptible jobs $1, \dots, K$ with their lengths equal N_1, \dots, N_K respectively. Apply the algorithm of Gonzalez

and Sahni [6] for $Q|pmtn|C_{max}$ to these K jobs. In the resulting schedule, say S , replace job i by its corresponding chain Ch_i , preempting the UET jobs of CH_i if necessary. Thus obtaining schedule S' . From S' , delete any UET job that either is preempted or starts too early for a communication delay of one time unit to be fit between the job and its immediate predecessor scheduled on another processor. The algorithm of Gonzalez and Sahni ensures that the number of UET jobs deleted from S' does not exceed $2(M - 1)$. Thus, if we add $2(M - 1)$ jobs at the end of schedule S' to start at $C_{S'} + 1$ and finish by $C_{S'} + 2M - 1$ on the fastest processor with its anti-speed equal to 1, then we can easily re-arrange the UET jobs in each chain to obtain a feasible schedule for $Q|c_{ij} = 1, chain, p_j = 1|C_{max}$ with makespan not exceeding $C_{S'} + 2M - 1$. The above approach can be implemented in time $O(K \log K)$ which is essentially the time required by the algorithm of Gonzalez and Sahni. ■

We end this section with a simple observation that, contrary to optimal scheduling of UET chains on uniform processors with unit communication delays, optimal scheduling of UET chains on identical processors with unit communication delays is not affected by the delays.

Observation 1 *Unit communication delays never affect optimal makespans for $P|c_{ij} = 1, chain, p_j = 1|C_{max}$.*

Proof. Let us consider the well-known McNaughton's lower bound of $\max\{N_1, \lceil N/M \rceil\}$ for the problem [8]. Any chain Ch_i with $N_i = \max\{N_1, \lceil N/M \rceil\}$ can be scheduled on a single processor, and thus, will require no communication. Furthermore, the wrap-around rule leaves at least one time unit between the jobs of any chain Ch_i with $N_i < \max\{N_1, \lceil N/M \rceil\}$ scheduled on two different processors which allows enough time for communication. Thus, the McNaughton's lower bound does not need to be exceeded in optimal schedules for $P|c_{ij} = 1, chain, p_j = 1|C_{max}$. ■

5 Concluding Remarks

We showed in this paper that the problem of scheduling chains of UET jobs on uniform processors with communication delays to minimize makespan is NP-hard in the strong sense. We proposed a simple heuristic that generates solutions within $2M - 1$ units from optimum for the problem, and are currently working on a heuristic with smaller than $2M - 1$ absolute error.

We also investigated the structure of optimal solutions for the two processor problem of scheduling chains of UET jobs with communication delays, where one processor is a (integer) times faster than the other. This investigation lead to a linear time optimization algorithm for this case. However, the complexity of the problem for arbitrary rational a remains open. So is the complexity of the two processor problem of scheduling trees of UET jobs with communication delays.

Acknowledgment

This research of the first author has been supported by NSERC Grant OGP0105675, and by a grant from the Fondation Scientifique de Lyon et du Sud-Est.

Références

- [1] An overview of the Apache project, <http://www-apache.imag.fr>
- [2] J. Błażewicz, F. Guinand, B. Penz, and D. Trystram. Scheduling complete trees on two uniform processors with arbitrary speed ratios and communication delays. submitted to *Information Processing Letters*.
- [3] P. Brucker, J. Hurink, and W. Kubiak. Scheduling jobs with unit processing requirements and chain precedences on two uniform machines. Technical Report 186, Osnabrücker schriften zur Mathematik, 1996. To appear in *Mathematical Methods of OR*, 1999.
- [4] Gabow, H.N. An almost-linear algorithm for two-processor scheduling. *Journal Assoc. Comput. Mach.* 29, 1982, pp. 766-780.

- [5] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman, New York, 1979.
- [6] T. Gonzalez and S. Sahni. Preemptive Scheduling of Uniform Processor Systems. *Journal Assoc. Comput. Mach.* 25, 1978, pp. 766-780.
- [7] W. Kubiak. Optimal scheduling of unit-time tasks on two uniform processors under tree-like precedence constraints. *Zeitschrifts für Operations Research*, 33:423–437, 1989.
- [8] R. McNaughton Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.
- [9] Lawler, E.L. [1982]: Preemptive scheduling of precedence-constrained jobs on parallel machines, in: Dempster, M.A.H., Lenstra, J.K., Rinnooy Kan, A.H.G.: *Deterministic and stochastic scheduling*, Reidel, Dordrecht.



Unit e de recherche INRIA Lorraine, Technop ole de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS L ES NANCY
Unit e de recherche INRIA Rennes, Irista, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit e de recherche INRIA Rh one-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

 diteur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
<http://www.inria.fr>
ISSN 0249-6399