Worst case analysis of decomposed software pipelining for cyclic unitary RCPSP with precedence delays

Abir Benabid, Claire Hanen

MISTA 2009 manuscript No.
(will be inserted by the editor)

Worst case analysis of Decomposed Software Pipelining for cyclic unitary RCPSP with precedence delays ⋆

Abir Benabid · Claire Hanen

the date of receipt and acceptance should be inserted later

Abstract In this paper we address a cyclic scheduling problem: finding a periodic schedule with minimal period for unitary resource constrained cyclic scheduling problem. The main originality is to cope with both precedence delays and complex resources settings which make the problem $\mathcal{NP}$-complete in general.

A guaranteed approach, called Decomposed Software Pipelining, has been proposed by Gasperoni and Schwiegelshohn [1], followed by the retiming method by Calland, Darte and Robert [2] to solve the problem assuming parallel identical processors and ordinary precedence constraints. In this paper, an extension of this approach to unitary resource-constrained cyclic scheduling problems with precedence delays is analyzed and its worst case performance ratio is provided.

1 Introduction

Cyclic scheduling problems [3] have numerous practical applications in production systems [4], [5], [6] as well as in embedded systems [7], [8], [3] used for devices such as mobile, automotive and consumer electronics. Our research in this field is particularly motivated by the advances in hardware technology, but our results are also valid for mass production systems. In fact, instruction scheduling, also known as software pipelining [9] is produced by the compiler in embedded architectures and aims to reduce the operating frequency given real-time processing requirements. The high-quality of such schedules is then a performance critical optimization that has a direct impact on the overall system cost and energy consumption. Most of today's high performance applications uses instruction level parallel processors such as VLIW processors [10].

⋆ *ORCYAE project supported by awards from DIGITEO, a research park in Ile-de-France dedicated to information science and technology.*

A. Benabid
LIP6, Pierre and Marie Curie University
E-mail: Abir.Benabid@lip6.fr

C. Hanen
Paris Ouest University and LIP6
E-mail: Claire.Hanen@lip6.fr

To benefit from these parallel structure, software pipelining techniques aim to exploit parallelism across more than one basic block of the code. Since most of the programs are composed with loops with a very large number of iterations (Image processing is a typical example), the parallelism should be extracted from loops. Hence, to achieve efficiency, instruction scheduling can be modeled by a cyclic scheduling problem given by a bivalued (uniform) cyclic precedence graph and resource constraints.

Among the different cyclic scheduling frameworks, modulo (periodic) scheduling [11], [3], [7] is the most successful in compilers (for example the LAO compiler of STMicroelectronics) and production systems [12],[5],[6] since it induces a short description of schedules that optimizes the memory requirements of the loop code provided by the compiler, as well as the way schedules are executed in production systems. In this approach, each task is repeated periodically every λ time units. The aim is then to provide a periodic schedule with minimal period λ. This problem is $\mathcal{NP}$-hard even if unitary processing times, no precedence delays and parallel processors are assumed [3]. If no resource constraints are involved, then the problem can be solved in polynomial time - all references can be found in ([8], p. 103-128). Notice that polynomial subproblems have been investigated when the precedence graph is acyclic [3], or when some decisions are made on the ordering of tasks on resources [5],[6].

In order to model the software pipelining problem, [7] introduces the RCMSP (Resource Constrained Modulo Scheduling Problem) as an extension of the usual resource constrained scheduling problem RCPSP [13] with precedence delays and provides an integer programming model. This choice is driven by the fact that RCMSP enables to model both ressource and precedence constraints induced by the features of VLIW processors. These processors are composed of functional units of different types. An instruction may use several types of functional units at the same time (for example memory access an floating point unit). We consider the special case, denoted unitary RCMSP, where the resource demands are unitary. This restriction is motivated by the VLIW structure, where usually no more than one functional unit of each type is needed by an instruction. Moreover, VLIW functional units are usually pipelined. This means that they can start a new instruction whereas the previous instruction is still in progress. Hence the precedence constraints induced by data dependences between instructions are subject to precedence delays in the RCMSP.

Although periodic scheduling heuristics are commonly used in instruction scheduling of modern VLIW architectures, in the classic modulo scheduling framework [11], [14], [15], [16], the theoretical efficiency of the corresponding schedules is not established. However, the noticeable Decomposed Software Pipelining (DSP) approach introduced in [1], [17], [2] has been proved to be among the most efficient in practice [18], and provides the only algorithms with a worst case performance bound. The idea of DSP is to decompose the periodic scheduling problem into two parts: a polynomially solvable graph problem, and a resource constrained non cyclic scheduling problem. The solutions of the two problems can then be recombined to get a feasible periodic schedule. Notice that the performance bound given in [2] assumes simple precedence constraints and parallel machines, then it is extended, in [18], to the case of precedence constraints with delays.

The goal of this paper is to explore more deeply the Decomposed Software Pipelining approach and to extend it to deal with unitary RCMSP resource settings and arbitrary non-negative start to start precedence delays. We then provide a worst case performance analysis that generalizes the ones of [1] and [2] which have been developed for parallel processors and simple precedences. Our performance guarantee outperforms the bound given by [18] and extends it to the case of specialized processors. We show that Decomposed Software Pipelining is still a guaranteed algorithm in this context although the theoretical worst case ratio increases with the number of resource types.

Section 2 presents the mathematical formulation of the cyclic unitary resource constrained modulo scheduling problem RCMSP with precedence delays, illustrated by a loop issued from the ST200 (ST Microelectronics) compiler. In section 3, we introduce the Decomposed Software Pipelining approach and we define a generic algorithm Extended DSP to solve our problem. Section 4 is devoted to the worst case analysis of the extended DSP algorithm. Section 5 concludes the paper.

2 Problem formulation

Let us now define more formally the problem adressed in this paper. Denotations are mainly issued from the classical cyclic scheduling papers [3].

2.1 Resources, tasks, precedences

An instance of a unitary resource-constrained cyclic scheduling problem can be defined by:

1. An architecture model characterized by k resource types. The availability of resource type $r \in \{1, \ldots, k\}$ is denoted by m_r. The resource type r may be viewed as m_r parallel processors.
2. A set $\mathcal{T}$ of n tasks or instructions $\{T_i\}_{1 \leq i \leq n}$ with integer processing time $\{p_i\}_{1 \leq i \leq n}$. To each task T_i is associated a binary vector $\{b_r^i\}_{1 \leq r \leq k}$ over the resource types, such that T_i uses b_r^i units of type r resource during its execution. Notice that a task might use several processors but of different types. Each task T_i must be performed an infinite number of times. We call T_i at iteration q and denote by (T_i, q) the q^{th} execution of T_i.
3. An inner loop modeled by a cyclic precedence bivalued graph $G = (\mathcal{T}, \mathcal{E})$ where $\mathcal{E}$ is a set of edges defining uniform dependence relations. An edge between two tasks $(T_i, T_j) \in \mathcal{E}$ is characterized by two integers:
   Its height h_{ij} is a non-negative integer.
   Its delay l_{ij} is an integer such that the scope $\delta_{ij} = p_i + l_{ij} \geq 0$.
   This edge models the fact that for any iteration number q, the task T_j at iteration $q + h_{ij}$ has to be issued at least $p_i + l_{ij}$ time units after the starting time of task T_i at iteration q.
   Usually, G is depicted by giving to each edge (T_i, T_j) the value $(p_i + l_{ij}, h_{ij})$ (see figure 2).

This model generalizes the classical parallel processors statements (in which $k = 1$ -i.e. there is a unique resource type) as well as typed tasks systems where each binary

vector $\{b_r^i\}_{1\leq r\leq k}$ has only one positive component. Moreover usual cyclic precedences [3], [1], [2] are defined by null delays.

As shown in the example section, positive delays allows to model data dependences between tasks performed by pipelined functional units, whereas negative delays l_{ij} such that $p_i + l_{ij} \geq 0$ may model synchronization between tasks.

2.2 Schedules and problem statement

A resource-constrained cyclic schedule σ assigns a starting time $\sigma(T_i, q)$ for each task occurrence (T_i, q) such that for all $r \in \{1, \cdots, k\}$ and for each time slot t, the number of tasks using resource type r at time t is at most equal to m_r, and

$$(T_i, T_j) \in \mathcal{E} \Rightarrow \sigma(T_i, q) + p_i + l_{ij} \leq \sigma(T_j, q + h_{ij}) \quad \forall q \in \mathbb{N}$$

Among the different software pipelining frameworks, periodic scheduling [11] is the most successful in compilers since their compact representation makes them easy to implement, shortening the program size. A periodic schedule σ is defined by its period λ and the schedule of the first iteration $\sigma(T_i, 0)_{i\in\{1,\ldots,n\}}$:

$$\forall i \in \{1, \cdots, n\}, \forall q \in \mathbb{N} : \quad \sigma(T_i, q) = \sigma(T_i, 0) + q \cdot \lambda$$

In this paper we consider the problem of finding a periodic schedule σ with minimal period λ such that:

1. Precedence constraints are met:
$$\forall (T_i, T_j) \in \mathcal{E}, \quad \sigma(T_j, 0) - \sigma(T_i, 0) \geq p_i + l_{ij} - \lambda h_{ij}$$
2. Resource constraints are met : for any time slot t and any resource type r, the number of tasks T_i using r (i.e. such that $b_r^i = 1$) that are beeing processed at time t is not greater than m_r.

2.3 Assumptions and notations

Let us denote, for any circuit c of G, by $H(c)$ the sum of the heights of the arcs of c, and by $L(c)$ the sum of the scopes of its arcs (processing times plus delays).

It is well known [8], [3] that the precedence graph is consistent (i.e. assuming no resource constraints, there exists an infinite schedule) if and only if for any circuit c of G, either $H(c) > 0$, or $H(c) = L(c) = 0$.

As we consider here models for loops, a circuit c with $H(c) = L(c) = 0$ is unlikely to exist. This would mean that all tasks of the circuit are to be scheduled at the same time in each iteration, and such constraints are not induced by data dependences nor by usual synchronizations. So we assume that for any circuit c of G, $H(c) > 0$.

We note σ^∞ an optimal periodic schedule for the problem considering unbounded resources (thus relaxing the resource constraints) and λ^∞ its period. This schedule can be computed in polynomial time using graph algorithms. Indeed, according to many authors (see [3] for references),

$$\lambda^\infty = \max_{c \ circuit \ of \ G} \frac{L(c)}{H(c)},$$

The computation of this value can be done in polynomial time. A review of efficient graph algorithms for this problem can be found in [19], and a polynomial parametric algorithm is given in [20]. Moreover, if we define the value of an arc (T_i, T_j) to be $p_i + l_{ij} - \lambda^\infty h_{ij}$, and the value of a path by the sum of values of its arcs, then $\sigma^\infty(T_i, 0)$ can be defined as the maximum value of a path to T_i.

In the rest of the paper, we use the following notations:

$$m_{max} = \max_{1 \le r \le k} m_r$$
$$l_{max} = \max(0, \max_{(T_i,T_j) \in \mathcal{E}} l_{ij})$$
$$p_{min} = \min_{1 \le i \le n} p_i, \quad p_{max} = \max_{1 \le i \le n} p_i$$
$$\rho = \frac{l_{max}}{p_{min}}$$

2.4 Example

Let us illustrate the problem adressed in this paper with the following example, given in Figure 1, which represents a C program source code and the corresponding ST200 VLIW processor operations ([7], p.270). We will work on this example throughout the paper.

```
int prod(int p, short a[], short b)
{ int s    0, i;
   for(i = 0; i < p; i + +){
     s+ = a[i] + b;
   }
   return s;
}
```

```
L?_0_8:
LDH_1     g131   =   0, G127        loads a[i] in register g131
MULL_2    g132   =   G126, g131     multiplies a[i] by b
ADD_3     G129   =   G129, g132     adds the result to s
ADD_4     G128   =   G128, 1        increments i
ADD_5     G127   =   G127, 2        computes the address of a[i]
CMPNE_6   b135   =   G118, G128     compares i and p
BRF_7     b315   ,   L?_0_8:        branches to the loop body
```

Fig. 1 A sample C program and inner loop body compiler representation.

The loop has $n = 7$ unit processing time tasks (instructions) displayed on the right and numbered from top to bottom. Each one is executed N times, where N is a given parameter representing the number of iterations that can be very large, and thus is assumed infinite.

Each instruction is composed with an operation code and register numbers. For example, $ADD\_3\ G129 = G129, g132$ means that registers $G129$ and $g132$ are added and the result is stored in register $G129$.

The resources availabilities and the resource requirements of each operation type are displayed in Table 1.

Resource	Issue	Memory	Control	Align
Availability	4	1	1	2
LDH	1	1	0	0
MULL	1	0	0	1
ADD	1	0	0	0
CMPNE	1	0	0	0
BRF	1	0	1	1

Table 1 Resources availabilities and operation requirements.

The resources are: Issue for the instruction issue width; Memory for the memory acces unit; and Control for the control unit. An artificial resource Align is also introduced to satisfy some encoding constraints. There are in our example 5 types of operations: ADD and CMPNE correspond to arithmetic operations. MULL, LDH and BRF are repectively multiply, memory and control operations.

The associated precedence graph $G(\mathcal{T}, \mathcal{E})$ is given in Figure 2. A dummy node 0 represents the start of an iteration and a dummy node $n+1 = 8$ represents the end of an iteration.

For example, consider the edge between nodes 1 and 2 with valuation $(3, 0)$. T_1 loads in register $g131$ the memory word $(a[i])$ whose address is in $G127$, whereas T_2 multiplies $g131$ (thus $a[i]$) by $G126$ in which b is stored. So T_1 at any iteration q must output its result in register $g131$ before T_2 at the same iteration reads it. As the loading operation is performed by a pipelined unit, the data in $g131$ is available only 3 time units after T_1 starts.

Consider now the edge between nodes 6 and 4 with valuation $(0, 1)$. T_6 compares the two registers containing values of i ($G128$) and p ($G118$), whereas T_4 increments $G128$. In order to observe the loop semantic, $(T_4, q + 1)$ cannot modify $G128$ before (T_6, q) compares it.

This model can also handle buffer constraints. Indeed, assume that $G127$ is a buffer of capacity 4. For any iteration q, (T_5, q) writes $G127$, so there must be a free position in the buffer. Hence the task T_1 at iteration $q - 4$ must have already read the buffer, leading to an arc from node 5 to node 1 with valuation $(1, 4)$.

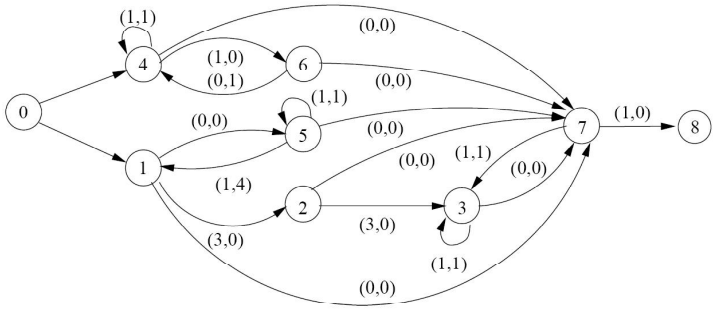

Fig. 2 A precedence graph $G(\mathcal{T}, \mathcal{E})$. Values $(p_i + l_{ij}, h_{ij})$ are displayed next to the corresponding arc.

Figure 3 displays a periodic schedule of $G(\mathcal{T}, \mathcal{E})$ with $\lambda = 2$, in which each operation is suffixed by the iteration number. The tasks of the first iteration are highlighted. Notice that if unbounded resources are assumed, the optimal period $\lambda^\infty = 1$, since there is no circuit c in the graph such that $\frac{L(c)}{H(c)} > 1$.

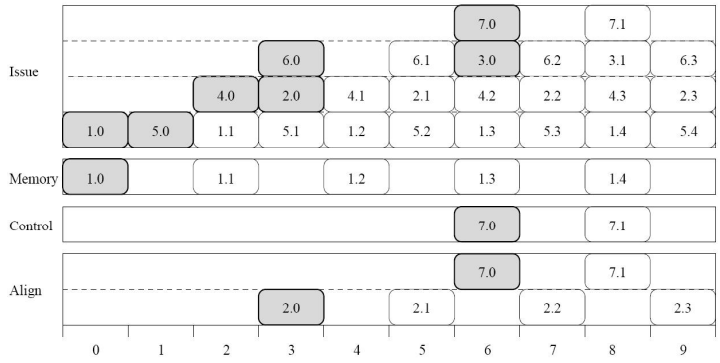

Fig. 3 An optimal periodic schedule of $G(\mathcal{T}, \mathcal{E})$.

3 Decomposed Software Pipelining

Generating an optimal resource constrained cyclic scheduling with minimal period is known to be $\mathcal{NP}$-hard [3]. The Decomposed Software Pipelining is a heuristic approach introduced simultaneously by [1] and [17]. The main idea is to decompose the problem into two subproblems: a graph problem, and a standard scheduling problem (acyclic) for which efficient algorithms are known.

The authors of [1] give an efficiency bound on the period λ for the problem with m identical processors and precedence without delays using this approach. Let λ^{opt} be the optimal (smallest) period, this bound is given by the following inequality:

$$\lambda \leq \left(2 - \frac{1}{m}\right) \lambda^{opt} + \left(1 - \frac{1}{m}\right) \left(\max_{1 \leq i \leq n} p_i - 1\right)$$

In [2], a heuristic based on circuit retiming algorithms improves slightly this efficiency bound.

In this paper, we generalize this approach for our problem. Several new elements have to be taken into account: extended resource constraints and precedence delays.

Notice first that the simplest way to compute a periodic schedule for a loop is to schedule the first iteration of the loop so that precedence relations within the first iteration - i.e. those for which $h_{ij} = 0$- and resource constraints are met. Then, it is always possible to find a value λ larger than the makespan of this schedule and repeat the first schedule every λ time units to get a feasible periodic schedule. This approach is particularly inefficient in general, since efficiency relies on the interleaving of iterations. Retiming algorithms, as introduced by [2], aimed to improve this simple approach.

We first use the definition of [2] for a legal retiming:

$$\mathcal{R} : \mathcal{T} \to \mathbb{Z}, \quad \forall (T_i, T_j) \in \mathcal{E},\ \mathcal{R}(T_j) + h_{ij} - \mathcal{R}(T_i) \geq 0$$

The intuition behind retiming is that (T_i, q), that is interpreted as the $q+1-$th execution of the first task $(T_i, 0)$, can also be interpreted as the $q+1+\mathcal{R}(T_i)-$th execution of a new first task $T_i' = (T_i, -\mathcal{R}(T_i))$. Changing the definition of first occurences of tasks allows to interleave tasks of different iterations into the new first iteration. Precedence relations also move: if (T_i, q) precedes $(T_j, q + h_{ij})$ then (T_i', q) precedes

$(T'_j, q + \mathcal{R}(T_j) + h_{ij} - \mathcal{R}(T_i))$. So the value $\mathcal{R}(T_j) + h_{ij} - \mathcal{R}(T_i)$ is the height of a new cyclic precedence relation.

Moreover, if $\mathcal{R}(T_j) + h_{ij} - \mathcal{R}(T_i) = 0$ then (T'_i, q) precedes (T'_j, q) for any large enough integer q. Otherwise, (T'_i, q) precedes an occurrence (T'_j, q') with $q' > q$. Hence for these new generic tasks $(T'_i)_{1 \leq i \leq n}$, the first iteration fulfills the precedence relations given by a graph called $G^{\mathcal{R}}$ computed from G by keeping only the arcs for which $\mathcal{R}(T_j) + h_{ij} - \mathcal{R}(T_i) = 0$. Notice that $G^{\mathcal{R}}$ is acyclic since G has no zero height circuit.

We now show how $G^{\mathcal{R}}$ can be used to get a periodic schedule of G. We add two dummy tasks T_{Start} and T_{Stop} with null processing times and no resource use. For each task T_i, we add arcs (T_{Start}, T_i) and (T_i, T_{Stop}) and we need to define valuations (delays) of these new arcs.

Let λ be any value. We say that the valuations are consistent with $G^{\mathcal{R}}$ and λ if

$$\forall (T_i, T_j) \in G \backslash G^{\mathcal{R}},\ l_{i,Stop} + l_{Start,j} \geq l_{ij} - \lambda(\mathcal{R}(T_j) + h_{ij} - \mathcal{R}(T_i) - 1). \quad (1)$$

Notice that consistent valuations with $G^{\mathcal{R}}$ and λ are also consistent with $G^{\mathcal{R}}$ and $\lambda' > \lambda$. Moreover, consistent valuations can always be defined as follows for each task T_i, with $\lambda = 0$, for example by setting :

$$\begin{cases} l_{Start,i} = 0 \\ l_{i,Stop} = \max\limits_{(T_i,T_j) \in G \backslash G^{\mathcal{R}}} l_{ij}. \end{cases} \quad (2)$$

In the last section, we will show that other values might be more efficient while the worst case bound can be obtained using condition (2).

Assume that we get such consistent valuations with a given λ and let π be any (non cyclic) schedule of $G^{\mathcal{R}}$ that fulfills the resource constraints as well as the precedences with delays induced by $G^{\mathcal{R}}$. We note π_i the start time of task T_i in this schedule. Then, setting $\lambda^{\mathcal{R}} = \max(\lambda, \pi_{Stop})$ and for any task T_i,

$$\sigma^{\mathcal{R}}(T_i, q) = \pi_i + (q + \mathcal{R}(T_i))\, \lambda^{\mathcal{R}}$$

we get the following result:

Lemma 1 *$\sigma^{\mathcal{R}}$ is a feasible periodic schedule of G with period $\lambda^{\mathcal{R}}$.*

Proof. First, we prove that, at any time slot t, the tasks scheduled at time t in $\sigma^{\mathcal{R}}$ meet the resource constraints. Let us note F_t the set of tasks for which one of occurrence is processed at time t. Let T_i and T_j be two tasks in F_t. We note q and q' their occurrences such that $\sigma^{\mathcal{R}}(T_i, q) \leq t < \sigma^{\mathcal{R}}(T_i, q) + p_i, \quad \sigma^{\mathcal{R}}(T_j, q') \leq t < \sigma^{\mathcal{R}}(T_j, q') + p_j$. Hence,

$$\begin{aligned} t = \sigma^{\mathcal{R}}(T_i, q) + s &= \sigma^{\mathcal{R}}(T_j, q') + s' \\ \pi_i + (q + \mathcal{R}(T_i))\, \lambda^{\mathcal{R}} + s &= \pi_j + \left(q' + \mathcal{R}(T_j)\right) \lambda^{\mathcal{R}} + s' \end{aligned}$$

$$\pi_i - \pi_j + s - s' - \left(\mathcal{R}(T_j) - \mathcal{R}(T_i) + q' - q\right) \lambda^{\mathcal{R}} = 0. \quad (3)$$

Since $\pi_i + s < \pi_i + p_i \leq \pi_{Stop} \leq \lambda^{\mathcal{R}}$ and similarly $\pi_j + s' < \lambda^{\mathcal{R}}$, $-\lambda^{\mathcal{R}} < \pi_i + s - \pi_j - s' < \lambda^{\mathcal{R}}$. Hence, the equality (3) gives:

$$\pi_i + s = \pi_j + s' \text{ and } \mathcal{R}(T_i) + q = \mathcal{R}(T_j) + q'$$

So, T_i and T_j are performed at the same time $\pi_i + s$ in the acyclic schedule π. Thus, $T_j \in F_{\pi_i+s}$ and then $F_t \subseteq F_{s+\pi_i}$. Since π fulfills the resource constraints induced by $G^{\mathcal{R}}$, F_{π_i} (and therefore F_t) meets the resource constraints.

For precedence constraints, we need to prove that, $\forall (T_i, T_j) \in \mathcal{E},\ \forall q \in \mathbb{N}$:

$$\begin{aligned} & \sigma^{\mathcal{R}}(T_i, q) + p_i + l_{ij} \ \leq \ \sigma^{\mathcal{R}}(T_j, q + h_{ij}) \\ \Leftrightarrow\ & \pi_i + (q + \mathcal{R}(T_i))\,\lambda^{\mathcal{R}} + p_i + l_{ij} \ \leq \ \pi_j + \left(q + \mathcal{R}(T_j) + h_{ij}\right)\lambda^{\mathcal{R}} \end{aligned}$$

Hence, we have to check that inequality (4) is satisfied for each (T_i, T_j) in $\mathcal{E}$ and for any $q \in \mathbb{N}$

$$\pi_i - \pi_j + p_i + l_{ij} \leq \left(\mathcal{R}(T_j) - \mathcal{R}(T_i) + h_{ij}\right)\lambda^{\mathcal{R}}. \tag{4}$$

Case 1 : $(T_i, T_j) \in G^{\mathcal{R}}$

Then, $\mathcal{R}(T_j) - \mathcal{R}(T_i) + h_{ij} = 0$. Since π fulfills the precedence constraints induced by $G^{\mathcal{R}}$,

$$\pi_i + p_i + l_{ij} \leq \pi_j.$$

Hence, inequality (4) is satisfied.

Case 2 : $(T_i, T_j) \notin G^{\mathcal{R}}$ Thus, $\mathcal{R}(T_j) - \mathcal{R}(T_i) + h_{ij} > 0$. Using condition (1), we get:

$\pi_i - \pi_j + p_i + l_{ij} \ \leq \ \pi_i + p_i + l_{i,Stop} + l_{j,Start} + \lambda^{\mathcal{R}}(\mathcal{R}(T_j) + h_{ij} - \mathcal{R}(T_i) - 1) - \pi_j$

As $\pi_j \geq l_{j,Start}$ and $\pi_i + p_i + l_{i,Stop} \leq \pi_{Stop} = \lambda^{\mathcal{R}}$, we get:

$$\begin{aligned} \pi_i - \pi_j + p_i + l_{ij} \ &\leq \ \lambda^{\mathcal{R}}(\mathcal{R}(T_j) + h_{ij} - \mathcal{R}(T_i) - 1) + \pi_{Stop} \\ &\leq \ \lambda^{\mathcal{R}}(\mathcal{R}(T_j) - \mathcal{R}(T_i) + h_{ij}) \end{aligned}$$

which achieves the proof. ■

Now, the idea, previously used by [1] and [2] is to choose a particular retiming, then to use a guaranteed algorithm to get a schedule π of $G^{\mathcal{R}}$, and finally to extend the guarantee to the induced periodic schedule.

List scheduling algorithms are the most used heuristics for scheduling with precedence and resource constraints, trying to minimize the makespan C_{max}. We thus propose the following generic algorithm 1 to solve our problem, by using a list algorithm to produce π.

Algorithm 1: Extended DSP

1. Find a legal retiming $\mathcal{R}$ for G;
2. **for** $(T_i, T_j) \in \mathcal{E}$ **do**
   if $\mathcal{R}(T_j) - \mathcal{R}(T_i) + h_{ij} = 0$ **then**
   keep (T_i, T_j) in $G^{\mathcal{R}}$; add nodes Start and Stop and valuations satisfying condition (1) for some λ not greater than a lower bound of $\lambda^{\mathcal{R}}$ (for example satisfying condition (2));
3. Perform a list scheduling on $G^{\mathcal{R}}$ coping with both precedence and resource constraints. Compute π_i the start time of task T_i in this schedule and $\lambda^{\mathcal{R}} = C_{max}(G^{\mathcal{R}}) = \pi_{Stop}$;
4. Define the cyclic schedule $\sigma^{\mathcal{R}}$ by:
   for $1 \leq q \leq N$ **do**
   for $T_i \in \mathcal{T}$ **do**
   $\sigma^{\mathcal{R}}(T_i, q) = \pi_i + \lambda^{\mathcal{R}}(q + \mathcal{R}(T_i))$;

A legal retiming for G of Figure 2 is given in Table 2. The algorithms that might be used to compute such retiming are discussed later in section 4.3.

Tasks	T_1	T_2	T_3	T_4	T_5	T_6	T_7
$\mathcal{R}(T_i)$	0	1	2	0	1	1	2

The only arcs of G remaining in $G^{\mathcal{R}}$ are (T_6, T_4) and (T_3, T_7)

Table 2 A retiming $\mathcal{R}$ of G.

The schedule built by algorithm 1 is depicted in Figure 4. The two arcs (T_6, T_4) and (T_3, T_7) of $G^{\mathcal{R}}$ are shown, as well as the arcs (T_1, T_2) and (T_2, T_3) which are not in $G^{\mathcal{R}}$ and thus link tasks in two consecutive periods. Since $p_1 + l_{12} = p_2 + l_{23} = 3$, the period $\lambda^{\mathcal{R}}$, associated to the retiming of Table 2, is at least equal to 3.

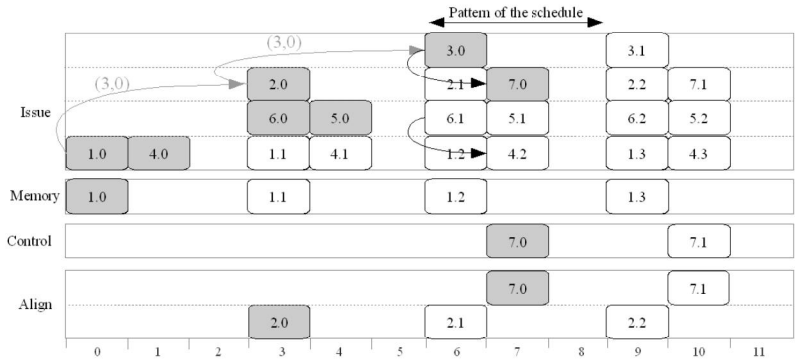

Fig. 4 A periodic cyclic schedule generated by algorithm 1: $\lambda^{\mathcal{R}} = 3$.

We remark that some idle cycles have been introduced by the scheduling algorithm. Hence, in order to define a worst case bound of algorithm 1, the idea is to bound the number of idle cycles occuring in the schedule. The worst case behavior of this algorithm is analyzed in the next section.

4 Worst case analysis

We first analyze the worst case performance of algorithm 1 in general. This is done by first analyzing the performance o the list algorithm on $G^{\mathcal{R}}$ (and in fact on any acyclic graph). In the case of m identical processors with non-negative precedence delays, [21] prove that list scheduling algorithms have the following worst case performance :

$$C_{max}^{list} \leq \left(2 - \frac{1}{m(\rho+1)}\right) C_{max}^{opt}$$

where $\rho = \dfrac{l_{max}}{p_{min}}$. For systems with unit execution time tasks, positive delays, and typed tasks, [22] give the following bound:

$$C_{max}^{list} \leq \left(k + 1 - \frac{1}{m_{max}(l_{max}+1)}\right) C_{max}^{opt}$$

We combine and extend the proofs of [22] and [21] to handle the fact that a task might use several resource types at the same time, and our slightly more general definition of precedence delays.

Then, we show that using some particular retimings, that can be computed in polynomial time, we can get an overall guarantee for the extended DSP algorithm.

4.1 Minimal length of pattern

Consider a dependence graph G. An acyclic graph $G^{\mathcal{R}}$ is obtained by a retiming $\mathcal{R}$. Then, we schedule $G^{\mathcal{R}}$ by a list algorithm and we generate a pattern π. We note $\phi^{\mathcal{R}}$ the length (sum of the delays and processing times) of the longest path in $G^{\mathcal{R}}$. Let λ^{opt} be the optimal period of G.

We consider two types of bounds obtained from resource and precedence constraints.

4.1.1 Resource bound

Lemma 2 *For each type r, let m_r be the number of machines of type r. Then,*

$$\lambda^{opt} \geq \max_{1 \leq r \leq k} \frac{\sum_{i=1}^{n} b_r^i \cdot p_i}{m_r}.$$

Proof. The shortest time required to complete the tasks using resources of type r on a single machine is $\sum_{i=1}^{n} b_r^i \cdot p_i$. Hence, on m_r parallel processors, the shortest time required is $\dfrac{\sum_{i=1}^{n} b_r^i \cdot p_i}{m_r}$. Furthermore, the optimal period λ^{opt} is not less than the time required to schedule these tasks once. ■

4.1.2 Precedence bounds

Let π be a schedule induced by a list algorithm on $G^{\mathcal{R}}$. In order to reveal the dependencies among tasks, we classify the time slots into three kinds:

1. A full slot t_f is a time slot in which at least all the processors of a certain type are executing tasks.
2. A partial slot t_p is a time slot in which at least one processor of each type is idle and this slot contains at least one non-idle processor.
3. A delay slot t_d is a time slot in which all processors are idle.

We note:

- p: the number of partial slots in π.
- d: the number of delay slots in π.

Lemma 3 *The partial-slots lemma:*
If π contains p partial slots and d delay slots, then $\phi^{\mathcal{R}} \geq p + d$.

Proof. We prove this lemma by finding a chain $h =< T_{j_c}, \cdots, T_{j_1} >$ in $G^{\mathcal{R}}$ such that the length of h is at least equal to $p + d$.

Let $T_{j_1} = T_{Stop}$ and assume that we already have a chain $< T_{j_{i-1}}, \cdots, T_{j_1} >$. Consider, if it exists, the predecessor T_{j_i} of $T_{j_{i-1}}$ such that $\pi_{j_i} + p_{j_i} + l_{j_i,j_{i-1}}$ is maximum. If such a predecessor cannot be found, set $c = i - 1$.

The construction of h leads to the following observation: All the slots between $\pi_{j_i} + p_{j_i} + l_{j_i,j_{i-1}}$ and $\pi_{j_{i-1}}$ (resp. before π_{j_c}) are full slots such that there is no available processor in a type of resource used by $T_{j_{i-1}}$ (resp. T_{j_c}). Otherwise, $T_{j_{i-1}}$ (resp. T_{j_c}) would have been scheduled earlier by the list algorithm.

Therefore, all the p partial slots and d delay slots are covered by the intervals $[\pi_{j_i}, \pi_{j_i} + p_{j_i} + l_{j_i,j_{i-1}})$, so that the length of h is not less than $p+d$. Thus, $\phi^{\mathcal{R}} \geq p+d$. ■

Lemma 4 *The delay-slots lemma:*
If $l_{max} > 0$ and if π contains d delay slots then $\phi^{\mathcal{R}} \geq d + \left\lceil \frac{d}{l_{max}} \right\rceil p_{min}$.

Proof. The schedule is computed by a list algorithm, then the number of any consecutive delay slots is not greater than l_{max}. Consider the chain h defined in the previous lemma. All the delay slots are included in $\cup_{1<i\leq c}[\pi_{j_i} + p_{j_i}, \pi_{j_i} + p_{j_i} + l_{j_i,j_{i-1}})$, since during interval $[\pi_{j_i}, \pi_{j_i} + p_{j_i})$, T_{j_i} is performed. Now the length of each interval $[\pi_{j_i} + p_{j_i}, \pi_{j_i} + p_{j_i} + l_{j_i,j_{i-1}})$ is less than l_{max}. So it holds that $c \cdot l_{max} \geq d$.

The length of h, which is $\sum_{i=c}^{2} p_{j_i} + l_{j_i,j_{i-1}} + p_{j_1}$, is thus not less than d plus the processing time of the chained tasks, which is greater than $c \cdot p_{min} \geq \left\lceil \frac{d}{l_{max}} \right\rceil p_{min}$. Thus, $\phi^{\mathcal{R}} \geq d + \left\lceil \frac{d}{l_{max}} \right\rceil p_{min}$. ■

4.2 Performance bound

In order to analyze the worst case performance of the algorithm, we need to decompose the schedule into idle and busy time units on each unit of the resources. An idle cycle (resp. non idle) on processor P is a time unit for which P is idle (resp. busy). Here we define the notations to be used below:

$M = \sum_{1\leq r\leq k} m_r$.

u_r: the number of non-idle cycles on processors of type r in π.

v_r: the number of non-idle cycles on processors of type r in partial slots in π.

$V = \sum_{1\leq r\leq k} v_r$: the number of non-idle cycles in partial slots in π.

We now prove a first bound on the algorithm performance based on the longest path in $G^{\mathcal{R}}$. The length of a path is assumed to be the sum of processing times of tasks and precedence delays along the path. We denote by $\phi^{\mathcal{R}}$ the maximal length of a path in $G^{\mathcal{R}}$.

Theorem 1 *Consider a dependence graph G. Let $\mathcal{R}$ be a legal retiming $\mathcal{R}$ on G. Then,*

$$\frac{\lambda^{\mathcal{R}}}{\lambda^{opt}} \leq k + \left(1 - \frac{1}{m_{max}(\rho+1)}\right)\frac{\phi^{\mathcal{R}}}{\lambda^{opt}}.$$

Proof. Consider the pattern π:

$$M\lambda^{\mathcal{R}} = \text{number of non-idle cycles} + \text{number of idle cycles.}$$

where the second term on the right hand side can be decomposed as the number of idle cycles occurring during delay slots, partial slots and full slots.

1. The number of idle cycles per processor occurring during delay slots is equal to Md.
2. The number of idle cycles per processor occurring during partial slots is not greater than $Mp - V$.
3. Using Lemma 2, we now give a bound on the number of idle cycles occurring during full slots.
   Notice that for a resource type r, there are at most $\frac{u_r - v_r}{m_r}$ full slots in which resource r is full. Thus the number of idle cycles in these slots is at most $(M - m_r)\frac{u_r - v_r}{m_r}$. Hence we get the following upper bound on the number of idle cycles occurring in full slots:
   $$\begin{aligned}
   &\leq \sum_{1\leq r\leq k}(M-m_r)\frac{u_r - v_r}{m_r}\\
   &\leq \sum_{1\leq r\leq k}(M-m_r)\frac{u_r}{m_r} - \sum_{1\leq r\leq k}(M-m_r)\frac{v_r}{m_r}\\
   &\leq \sum_{1\leq r\leq k}(M-m_r)\max_{1\leq r\leq k}\frac{\sum_{i=1}^{n} b_r^i \cdot p_i}{m_r} - (M-m_{max})\frac{\sum_{1\leq r\leq k} v_i}{m_{max}}\\
   &\leq (kM-M)\lambda^{opt} - (M-m_{max})\frac{V}{m_{max}}
   \end{aligned}$$

   Then,
   $$\begin{aligned}
   M\lambda^{\mathcal{R}} &\leq M\lambda^{opt} + (kM-M)\lambda^{opt} - (M-m_{max})\frac{V}{m_{max}} + Mp - V + Md\\
   &\leq kM\lambda^{opt} - (M-m_{max})\frac{V}{m_{max}} + Mp - V + Md\\
   &\leq kM\lambda^{opt} - M\frac{V}{m_{max}} + Mp + Md
   \end{aligned}$$

   V is the number of non-idle cycles in partial slots, since a partial slot contains at least one non-idle cycle, $V \geq p$. Thus,
   $$\begin{aligned}
   \lambda^{\mathcal{R}} &\leq k\lambda^{opt} - \frac{p}{m_{max}} + p + d\\
   &\leq k\lambda^{opt} + \left(1 - \frac{1}{m_{max}}\right)(p+d) + \frac{1}{m_{max}}d
   \end{aligned}$$

   Notice that if $l_{max} = 0$, then there is no delay slot and all slots are either full or partial in the schedule. Hence, $d = 0$ and using Lemma 3, we get
   $$\lambda^{\mathcal{R}} \leq k\lambda^{opt} + \left(1 - \frac{1}{m_{max}}\right)\phi^{\mathcal{R}}$$

   Thus, the bound of Theorem 1 holds in this case. Now, if $l_{max} > 0$,

$$\lambda^{\mathcal{R}} \leq k\lambda^{opt} + \left(1 - \frac{1}{m_{max}}\right)(p+d) + \frac{1}{m_{max}} d \left(1 + \frac{p_{min}}{l_{max}}\right) \left(\frac{1}{1 + \frac{p_{min}}{l_{max}}}\right)$$
$$\leq k\lambda^{opt} + \left(1 - \frac{1}{m_{max}}\right)(p+d) + \frac{1}{m_{max}} \left(d + \left\lceil \frac{d}{l_{max}} \right\rceil p_{min}\right) \left(\frac{\rho}{\rho+1}\right)$$

From Lemma 3, we have $\phi^{\mathcal{R}} \geq p + d$ and from Lemma 4,$\phi^{\mathcal{R}} \geq d + \left\lceil \frac{d}{l_{max}} \right\rceil p_{min}$. Then,

$$\lambda^{\mathcal{R}} \leq k\lambda^{opt} + \left(1 - \frac{1}{m_{max}}\right)\phi^{\mathcal{R}} + \frac{1}{m_{max}} \frac{\rho}{\rho+1} \phi^{\mathcal{R}}$$
$$\leq k\lambda^{opt} + \left(1 - \frac{1}{m_{max}(\rho+1)}\right)\phi^{\mathcal{R}}$$

which achieves the proof. ■

Notice that this theorem allows to extend of the bounds given by [22] and [21] to the non cyclic scheduling problem where unitary RCPSP resource constraints and precedence delays are considered. Let us denote by C^{list}_{max} the makespan of of a list schedule, and by C^{opt}_{max} the optimal makespan for such a non cyclic instance. Then:

Corollary 1

$$C^{list}_{max} \leq \left(k + 1 - \frac{1}{m_{max}(\rho+1)}\right) C^{opt}_{max}$$

Proof. In the proof of Theorem 1, $G^{\mathcal{R}}$ could be any acyclic graph. As $\lambda^{\mathcal{R}}$ is the makespan of the schedule, and as the resource bound of Lemma 2 is also met by C^{opt}_{max}, as well as the precedence bound (C^{opt}_{max} is not less than the length of the longest path in the graph, figured by $\phi^{\mathcal{R}}$ in the proof of Theorem 1), we come to the above inequality. ■

4.3 Choosing a good retiming

In order to improve the performance bound, it seems important to minimize the ratio between $\phi^{\mathcal{R}}$ and λ^{opt}. So if we have a good lower bound LB of λ^{opt}, using Theorem 7 in [23], we can check the existence of a legal retiming $\mathcal{R}'$ such that $\phi^{\mathcal{R}'} \leq LB \leq \lambda^{opt}$. If such a retiming exists, we have the following performance guarantee:

$$\frac{\lambda^{\mathcal{R}'}}{\lambda^{opt}} \leq k + 1 - \frac{1}{m_{max}(\rho+1)}.$$

Two approaches have been investigated for parallel processors and usual precedence constraints. The first one [1] bases a retiming on a schedule assuming unbounded resources. The second one [2] uses algorithms to compute a retiming $\mathcal{R}$ for which $\phi^{\mathcal{R}}$ is minimum. We investigate here the performances of these two approaches for our problem.

4.3.1 Retiming based on an optimal schedule for unbounded resources

In this subsection, we show that using a retiming associated with the optimal periodic schedule σ^{∞} of period λ^{∞} considering unbounded resources, one can define values of the delays associated with the edges connected to the dummy nodes so that the worst case performance ratio of the extended DSP algorithm can be provided.

According to [1],[2], σ^∞ can be used to produce a feasible retiming as follows: Let us define $\pi^\infty : \mathcal{T} \to [0, \lambda^\infty - 1]$ and $\mathcal{R}^\infty : \mathcal{T} \to \mathbb{Z}$ such that:

$$\sigma^\infty(T_i, 0) = \pi_i^\infty + \lambda^\infty \mathcal{R}^\infty(T_i),\ \forall T_i \in \mathcal{T}$$

Then, the precedence constraint for each edge $(T_i, T_j) \in \mathcal{E}$ is:

$$\begin{aligned} \sigma^\infty(T_i, 0) + p_i + l_{ij} &\leq \sigma^\infty(T_j, 0) + \lambda^\infty h_{ij} \\ \pi_i^\infty + p_i + l_{ij} &\leq \pi_j^\infty + \lambda^\infty \left(h_{ij} + \mathcal{R}^\infty(T_j) - \mathcal{R}^\infty(T_i)\right) \end{aligned}$$

Let us define $C_{max}^\infty = \max_{1 \leq i \leq n} \pi_i^\infty + p_i$. Notice that $C_{max}^\infty \leq \lambda^\infty + p_{max} - 1$, since for any task i, $\pi_i^\infty \leq \lambda^\infty - 1$.

The following lemma shows the properties of π^∞ with respect to the retiming $\mathcal{R}^\infty$ that will be useful to analyze the performance of the algorithm.

Lemma 5 *If we set for any task T_i, $l_{Start,i} = \pi_i^\infty$ and $l_{i,Stop} = C_{max}^\infty - \pi_i^\infty - p_i$ then these values satisfy condition (1). Moreover, π^∞ is a feasible schedule of $G^{\mathcal{R}^\infty}$ with makespan $C_{max}^\infty = \phi^{\mathcal{R}^\infty}$.*

Proof. Let us consider an edge $(T_i, T_j) \in \mathcal{E}$, such that $h_{ij} + \mathcal{R}^\infty(T_j) - \mathcal{R}^\infty(T_i) = 0$, we have

$$\pi_i^\infty + p_i + l_{ij} \leq \pi_j^\infty.$$

If now $h_{ij} + \mathcal{R}^\infty(T_j) - \mathcal{R}^\infty(T_i) > 0$, thus (T_i, T_j) is not an arc of $G^{\mathcal{R}^\infty}$, we know that

$$\pi_j^\infty \geq \pi_i^\infty + p_i + l_{ij} - \lambda^\infty \left(h_{ij} + \mathcal{R}^\infty(T_j) - \mathcal{R}^\infty(T_i)\right)$$

Thus, $l_{Start,j} \geq C_{max}^\infty - l_{i,Stop} + l_{ij} - C_{max}^\infty \left(h_{ij} + \mathcal{R}^\infty(T_j) - \mathcal{R}^\infty(T_i)\right)$. We come naturally to condition (1):

$$l_{Start,j} + l_{i,Stop} \geq l_{ij} - C_{max}^\infty \left(h_{ij} + \mathcal{R}\infty(T_j) - \mathcal{R}^\infty(T_i) - 1\right)$$

Now $\phi^{\mathcal{R}^\infty} \geq C_{max}^\infty$ according to the valuations of the incident arcs to the dummy nodes. Conversely, any path from T_{Start} to a node T_i has a valuation not greater than π_i^∞. Hence if we add the arc (T_i, T_{Stop}), the length of a path is not greater than $l_{i,Stop} + p_i + \pi_i^\infty = C_{max}^\infty$. ■

This lemma will lead to the following worst case bound for the schedule build using this approach. Notice that this bound coincides with the one of [1] when parallel processors ($k = 1$) and no delays ($\rho = 0$) are assumed.

Theorem 2 *Consider a dependence graph G. Let $\mathcal{R}^\infty$ be a retiming on G based on the schedule with unbounded resources.*

$$\lambda^{\mathcal{R}^\infty} \leq \left(k + 1 - \frac{1}{m_{max}(\rho + 1)}\right) \lambda^{opt} + \left(1 - \frac{1}{m_{max}(\rho + 1)}\right) (p_{max} - 1).$$

Proof. This is a simple outcome of Theorem 1, using Lemma 5: replacing $\phi^{\mathcal{R}^\infty}$ by C_{max}^∞ and using $C_{max}^\infty \leq \lambda^\infty + p_{max} - 1 \leq \lambda^{opt} + p_{max} - 1$. ■

4.3.2 Retiming minimizing $\phi^{\mathcal{R}}$

An another approach, introduced in [2], aims at minimizing the length of the longest path of the reduced graph. There are well-known retiming algorithms [23] to minimize $\phi^{\mathcal{R}}$, or to be more accurate, to minimize $\bar{\phi}^{\mathcal{R}}$, the length of a path excluding the two dummy nodes T_{Start} and T_{Stop}. Let $\mathcal{R}^{opt}$ be a retiming for which $\bar{\phi}^{\mathcal{R}}$ is minimum. We note $\phi^{opt} = \phi^{\mathcal{R}^{opt}}$. We prove that this retiming can be used to get a performance ratio of the extended DSP algorithm which is similar to the one of Theorem 2.

Lemma 6 *Let $\delta = \max_{(T_i,T_j)\in\mathcal{E}}(p_i + l_{ij})$ be the maximum scope in G. Then,*

$$\lambda^{\infty} + \delta - 1 \geq \phi^{opt}.$$

Proof. Consider the retiming $\mathcal{R}^{\infty}$ associated with the schedule σ^{∞} defined in the previous subsection. Let us define for any task T_i the values $l_{Start,i} = 0$ and $l_{i,Stop} = \max_{(T_i,T_j)\in G\setminus G^{\mathcal{R}^{\infty}}} l_{ij}$. Let $h =< T_{j_1}, \cdots, T_{j_c}, T_{Stop} >$ be a path in $G^{\mathcal{R}^{\infty}}$.

$$\pi^{\infty}_{j_i} + p_{j_i} + l_{j_i,j_{i+1}} \leq \pi^{\infty}_{j_{i+1}},\ \forall i \in \{1, \cdots, c-1\},$$

By summing up these $c-1$ inequalities, we have

$$\pi^{\infty}_{j_1} + \sum_{i=1}^{c-1}(p_{j_i} + l_{j_i,j_{i+1}}) \leq \pi^{\infty}_{j_c}$$

Thus, $\sum_{i=1}^{c-1}(p_{j_i} + l_{j_i,j_{i+1}}) \leq \pi^{\infty}_{j_c}$. This inequality is true for any chain of $G^{\mathcal{R}^{\infty}}$ in particular for the longest path in $G^{\mathcal{R}^{\infty}} - \{T_{Start}, T_{Stop}\}$. Thus $\bar{\phi}^{\mathcal{R}^{\infty}} \leq \pi^{\infty}_{j_c} \leq \lambda^{\infty} - 1$ As $\bar{\phi}^{opt} \leq \bar{\phi}^{\mathcal{R}^{\infty}}$ and $\phi^{opt} \leq \bar{\phi}^{opt} + \delta$, we get

$$\phi^{opt} \leq \lambda^{\infty} - 1 + \delta$$

■

Finally, from Lemma 6, we deduce our last performance ratio.

Theorem 3 *Consider a dependence graph G. Let $\mathcal{R}^{opt}$ be a retiming on G that minimize $\bar{\phi}^{\mathcal{R}}$. Then,*

$$\lambda^{\mathcal{R}^{opt}} \leq \left(k + 1 - \frac{1}{m_{max}(\rho+1)}\right)\lambda^{opt} + \left(1 - \frac{1}{m_{max}(\rho+1)}\right)(\delta - 1).$$

Proof. Using Theorem 1 applied to $\mathcal{R}^{opt}$, we get:

$$\begin{aligned}
\lambda^{\mathcal{R}^{opt}} &\leq k\lambda^{opt} + (1 - \frac{1}{m_{max}(\rho+1)})\phi^{opt} \\
&\leq k\lambda^{opt} + (1 - \frac{1}{m_{max}(\rho+1)})(\lambda^{\infty} + \delta - 1) \\
&\leq k\lambda^{opt} + (1 - \frac{1}{m_{max}(\rho+1)})(\lambda^{opt} + \delta - 1) \\
&\leq (k + 1 - \frac{1}{m_{max}(\rho+1)})\lambda^{opt} + (1 - \frac{1}{m_{max}(\rho+1)})(\delta - 1)
\end{aligned}$$

■

The two ways used to define an efficient retiming lead to similar performances, as in the parallel processor case. Although this second bound is slightly worse than the one of theorem 2, because of the additive constant, it is likely that improvements might be made to tighten this bound by refining Leiserson's algorithm in order to adjust the values of the incident arcs to the dummy nodes to get the same worst case performance.

4.4 Tightness

In order to discuss the tightness of our bounds, we consider the class of acyclic graphs given by [22] which asymptotically approaches the ratio $\left(k+1-\frac{1}{m_{max}(\rho+1)}\right)$ for makespan minimization on typed task systems with a special case of precedence delays. We consider an acyclic graph G^a in this class, with a dummy source T_{Start} and a dummy sink T_{Stop}, and set the height (h_{ij}) of its arcs to 0. We then add an edge from T_{Stop} to T_{Start} with $l_{Stop,Start} = 0$ and $h_{Stop,Start} = 1$. This last edge express the fact that tasks of the $q-$th iteration precede all tasks of iteration $q+1$.

It is easy to see that the retiming $\mathcal{R} = (0, \ldots, 0)$ equals $\mathcal{R}^{\infty}$ as well as $\mathcal{R}^{opt}$. According to algorithm 1 the pattern of the cyclic schedule is given by scheduling G^a by list algorithm, and then the period is equal to the makespan of G^a. Since the makespan of G^a provided by the list algorithm asymptotically reaches the worst case bound, the period of the cyclic schedule reaches this bound too. Hence we built a class of cyclic graphs for which the bound is asymptotically tight.

5 Conclusion

Instruction scheduling, which takes place when compiling applications for modern processors, affects critically the performance of the the overall system cost and energy consumption.

In this paper, we presented a generalized model of instruction scheduling but our results might also be used for applications in cyclic production systems.

We have built upon results of [1],[2], [22] and extended them to propose a guaranteed heuristic for unitary resource-constrained modulo scheduling problems. A worst case analysis of this heuristic is explored and a performance bound is established. It is the first guarantee derived for cyclic scheduling problems in the case of many different resources.

Now, what can be said if the resource demand is not unitary? Then, the Decomposed Software Pipelining algorithm still produces a feasible schedule. However, the use of list scheduling for solving the acyclic problem seems to be less interesting. Moreover our worst case analysis cannot be easily extended to handle general resource demands.

We point out that in all cases (unitary or not) it would be interesting to derive new algorithms, more sophisticated than list scheduling, to improve this performance bound. Finally, it would be worth to study the importance of choosing a good retiming and its impact on the worst case performance.

Acknowledgements Many thanks to the anonymous reviewers for their helpful comments.

References

1. Gasperoni, F., Schwiegelshohn, U.: Generating close to optimum loop schedules on parallel processors. Parallel Processing Letters **4**, 391–403 (1994)
2. Calland, P.Y., Darte, A., Robert, Y.: Circuit retiming applied to decomposed software pipelining. IEEE Transactions of Parallel Distribution Systems **9**(1), 24–35 (1998)
3. Hanen, C., Munier, A.: Cyclic scheduling on parallel processors: An overview. In: Scheduling theory and its applications. J. Wiley and sons (1994)
4. Proth, J.M., Xie, X.: Modélisation, analyse et optimisation des systèmes à fonctionnement cyclique. In: Les réseaux de Petri pour la conception et la gestion des systèmes de production. Masson (1995)
5. Alcaide, D., C.Chu, V.Kats, E.Levner, Sierksma, G.: Cyclic multiple-robot scheduling with time-window constraints using a critical path approach. European Journal of Operational Research **177**, 147–162 (2007)
6. Levner, E., Kats, V., de Pablo, D.A.L.: Cyclic scheduling in robotic cells: An extension of basic models in machine scheduling theory. In: E. Levner (ed.) Multiprocessor Scheduling: Theory and Applications, pp. 001–020. I-Tech Education and Publishing, Vienna Austria (2007)
7. Dupont de Dinechin, B., Artiques, C., Azem, S.: Resource constrained modulo scheduling. In: C. Artigues, S. Demassey, E. Neron (eds.) Resource-Constrained Project Scheduling: models, algorithms, extensions and applications, Control systems, robotics and manufacturing series, pp. 267–277. ISTE and John Wiley, London (2008)
8. Robert, Y., Vivien, F.: Introduction to Scheduling. CRC Press, Inc., Boca Raton, FL, USA (2009)
9. Allan, V.H., Jones, R.B., Lee, R.M., Allan, S.J.: Software pipelining. ACM Computer Survey **27**(3), 367–432 (1995)
10. Rau, B.R.: Dynamically scheduled VLIW processors. In: MICRO 26: Proceedings of the 26th annual international symposium on Microarchitecture, pp. 80–92. IEEE Computer Society Press, Los Alamitos, CA, USA (1993)
11. Rau, B.R.: Iterative modulo scheduling: an algorithm for software pipelining loops. In: MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture, pp. 63–74. ACM, New York, NY, USA (1994)
12. Kats, Levner, E.: Polynomial algorithms for periodic scheduling of tasks on parallel processors. In: L. Yang, M. Paprzycki (eds.) Practical Applications of Parallel Computing: Advances in Computation Theory and Practice, vol. 12, pp. 363–370. Nova Science Publishers, Canada (2003)
13. Brucker, P., Drexl, A., Mohring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research **112**(1), 3–41 (1999)
14. Huff, R.A.: Lifetime-sensitive modulo scheduling. In: In Proc. of the ACM SIGPLAN '93 Conf. on Programming Language Design and Implementation, pp. 258–267 (1993)
15. Llosa, J., González, A., Ayguadé, E., Valero, M.: Swing modulo scheduling: A lifetime-sensitive approach. pp. 80–86 (1996). URL citeseer.ist.psu.edu/llosa96swing.html
16. Lam, M.: Software pipelining: an effective scheduling technique for vliw machines. SIGPLAN Not. **23**(7), 318–328 (1988). DOI http://doi.acm.org/10.1145/960116.54022
17. Wang, J., Eisenbeis, C., Jourdan, M., Su, B.: Decomposed software pipelining: a new perspective and a new approach. International Journal of Parallel Program. **22**(3), 351–373 (1994)
18. Darte, A., Huard, G.: Loop shifting for loop compaction. International Journal of Parallel Programming **28**(5), 499 – 534 (2000)
19. Dasdan, A., Irani, S., Gupta, R.K.: Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. In: Design Automation Conference, pp. 37–42 (1999)
20. Levner, E., Kats, V.: A parametric critical path problem and an application for cyclic scheduling. Discrete Applied Mathematics **87**, 149–158 (1998)
21. Munier, A., Queyranne, M., Schulz, A.S.: Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In: IPCO, pp. 367–382 (1998)
22. Chou, H.C., Chung, C.P.: Upper bound analysis of scheduling arbitrary delay instruction on typed pipelined processors. International Journal of High Speed Computing **4**(4), 301–312 (1992)
23. Leiserson, C.E., Saxe, J.B.: Retiming synchronous circuitry. Algorithmica **6**, 5–35 (1991)