The resource-constrained modulo scheduling problem: an experimental study

Maria Ayala, Abir Benabid, Christian Artigues, Claire C. Hanen

The resource-constrained modulo scheduling problem: an experimental study

Maria Ayala[1,2], Abir Benabid[3], Christian Artigues[1,2] and Claire Hanen[3,4]

[1] CNRS; LAAS; 7 avenue du Colonel Roche, F-31077 Toulouse, France
[2] Université de Toulouse ; UPS, INSA, INP, ISAE, UT1, UTM, LAAS; F-31077 Toulouse, France
[3] LIP6, 4 place Jussieu, 75252 Paris cedex 05 France
[4] Université Paris Ouest Nanterre,200 av de la République, 92000 Nanterre, France

{Maria.Ayala, Christian.Artigues}@laas.fr
{Abir.Benabid,Claire.Hanen}@lip6.fr

Abstract

In this paper, we focus on the resource-constrained modulo scheduling problem, a general periodic scheduling problem, abstracted from the problem solved by compilers when optimizing inner loops at instruction level for VLIW parallel processors. Heuristic solving scheme have been proposed since many years to solve this problem, among which the decomposed software pipeling method. In this method, a cyclic scheduling problem ignoring resource constraints is first considered and a so-called legal retiming of the operations is issued. Second, a standard acyclic problem, taking this retiming as input, is solved through list scheduling techniques. In this paper, we propose an hybrid approach, which uses the decomposed software pipeling method to obtain a good retiming. Then the obtained retiming is used to build an Integer Linear Programming formulation of reduced size, which allows to solve it exactly. Experimental results show that a lot more problems are solved with this new approach. The gap to the optimal solution is really small (0 or 1%) on all the tested problem instances.

1 Introduction

A cyclic scheduling problem is specified by a set of generic operations that are processed an infinity of times and a set of constraints linked to precedence relations between the operations or to usage of limited resources. The objective is to generally maximize the throughput of scheduling. In this paper, we focus on the resource-constrained modulo scheduling problem (RCMSP), a general periodic scheduling problem, abstracted from the problem solved by compilers when optimizing inner loops at instruction level for VLIW parallel processors.

We briefly describe the context of the RCMSP in terms of parallel computing and very long instruction word (VLIW) architectures. We refer to [dD04, SMD04, RS02, ED97, EDA94, DAA08] for more details. In VLIW architectures, the instruction scheduling problem is a major compilation issue. To reduce the troughput, software pipelining is widely used. Software pipeling is a technique that allows to interleave the successive operations of a loop. Among software pipeling techniques, this paper focuses on the modulo scheduling technique, which implements software pipelining by generating periodic schedules. This is a suboptimal policy compared to k-periodic scheduling [HM94] but it is used in practice for ease of implementation. The goal is to find a valid schedule for the instructions of a loop (the local schedule) that can be overlapped with itself infinitely. In the modulo scheduling framework, the interval between two local schedules is called initiation interval (or period) and is the main indicator of the schedule quality. The modulo scheduling algorithm must take into account the constraints of the target processor, this is, latencies of operations, resources, and size of the register files. Also, it should consider optimizing secondary goals such as, minimizing the schedule length of a loop iteration, minimizing the register requirements of the resulting modulo schedule. Algorithms based on optimal solvers have been proposed, and are referred to as optimal modulo schedulers. In this study, register constraints and objectives are ignored. In this context, The RCMSP can be informally defined as a periodic scheduling problem consisting in minimizing the period, while satisfying precedence and resource constraints.

Solving the instruction scheduling problem at compilation phase in less time critical than for real time scheduling, integer linear programming (ILP) is a relevant technique for the RCMSP [dD04, ED97, EDA94, DAA08]. Hence, different ILP formulations for the RCMSP have been proposed and used in practice. These formulations can be presented as generalizations of the classical non preemptive time-indexed formulations of Pritsker *et al.*

[PWW69] and the tighter variant presented by Christofides *et al.* [CAVT87], for the (non periodic) resource constrained project scheduling problem (RCPSP). However, because of the periodic nature of the problem, the time-indexed formulation of the RCPSP can be extended in two different ways, yielding two categories of ILP formulations. Eichenberger et Davidson [ED97] proposed a first extension comprising both binary and integer variables. We call this category of formulations the decomposed formulations. Dupont de Dinechin [dD04, Dup07] proposed a second extension comprising only binary variables. We call the latter category the direct formulations. Ayala and Artigues [AA10] performed a series of computational experiments on a set of industrial and randomly-generated instances. They show that in terms of exact solving, the ILP formulations perform reasonably well although high computational times may be required to reach optimality for moderate size instances. For some hard instances optimum could not even be found after hours of computing.

Heuristic solving scheme have been proposed since many years to solve this problem, among which the decomposed software pipeling technique [Rau93, GS94, BdDH06, CDR98a]. This approach decomposed this problem into two subproblems solved consecutively. First, a cyclic scheduling problem ignoring resource constraints is considered and a so-called *legal retiming* of the operations is issued. Second, a standard acyclic problem, taking this retiming as input, is solved through list scheduling techniques. An extension of decomposed software pipelining has been proposed by [BH09a] for the resource-constrained cyclic scheduling problem with precedence delays. This approach has a very good performance in time and distance to optimum. However, even for the best variant of this approach, the optimum is not always reached, especially when the operations have non unit resource demands, which corresponds to the general case.

In this paper, we propose an hybrid approach, which uses the decomposed software pipeling method to obtain a good *retiming*. Then the obtained retiming is used to build a ILP formulation using the Eichenberger et Davidson [ED97] scheme. The formulation has a reduced size which allows to solve it exactly. Experimental results show that a lot more problems are solved with this new approach. The gap to the optimal solution is really small (0 or 1%) on all the tested problem instances.

Section 2 defines the considered resource-constrained modulo scheduling problem and presents the data intances used for the experiments. Section 3 presents the ILP formulations proposed by Eichenberger et Davidson [ED97] and Dupont de Dinechin [dD04, Dup07] and reports experimental results in terms of exact solving of the considered problem instances. In Section 4,

the approximation algorithms based on decomposed software pipeling are presented, as well as their results on the same problem instances. Section 5 details the proposed hybrid method and gives its results in comparison with the two previously-presented methods. Concluding remarks are drawn in Section 6.

2 The Resource-Constrained Modulo Scheduling Problem (RCMSP)

Instruction schedules, produced by the compiler in modern processors like VLIW (Very Long Insctruction Word) architectures, are a performance critical optimization that has a direct impact on the overall system cost and energy consumption. High-quality instruction schedules enable to reduce the operating frequency given real-time processing requirements.

This combinatorial problem is also known as software pipelining [AJLA95] and can be expressed by a cyclic scheduling problem with resource constraints. Among the different cyclic scheduling frameworks, modulo scheduling [Rau94] focuses on finding a periodic schedule with the minimal period λ and it is the most successful in production compilers. In order to model the software pipelining problem, [DAA08] proposes an extension of the classic modulo scheduling problem to Resource-Constrained Modulo Scheduling Problems RCMSP where the resources are adapted from the renewable resource of the Resource-Constrained Project Scheduling Problem RCPSP [BDM$^+$99].

2.1 Problem formulation

Modulo scheduling is equivalent to periodic scheduling with an integral period $\lambda \geq 1$. We consider a set $V = \{O_1, \ldots, O_n\}$ of n generic operations of unit duration. A schedule is defined by a mapping $\sigma : V \times \mathbb{N} \mapsto \mathbb{N}$, where $\sigma(O_i, q)$ represents the start time of the q^{th} instance (O_i, q) of generic operation O_i. For ease of notation, $\sigma(O_i, q)$ will be denoted as σ_i^q in the remainder of the paper. In addition, a periodic schedule satisfies:

$$\sigma_i^q = \sigma_i^0 + q\lambda, \forall i \in V, \forall q \in \mathbb{N} \tag{1}$$

Let E be the set that represents dependencies among operations, each dependence constraint can be written:

$$\sigma_i^q + \theta_i^j \leq \sigma_j^{q+\omega_i^j}, \forall (O_i, O_j) \in E, \forall q \in \mathbb{N}$$

where θ_i^j is the latency and represents dependence length while ω_i^j is the distance, representing the number of iterations separating the two operations involved.

Introducing σ_i as a shortcut for σ_i^0, the start time of the operation at the first iteration (defining the *local* schedule), and using (1) we obtain the following dependence constraints:

$$\sigma_i + \theta_i^j - \omega_i^j \lambda \leq \sigma_j, \forall (O_i, O_j) \in E \tag{2}$$

Each operation $i \in \{1, \ldots, n\}$ requires $b_i^s \in \mathbb{N}$ units of each resource $s \in \{1, \ldots, k\}$. Each resource s has a limited availability $m_s \in \mathbb{N}$. Note that since $\lambda \geq 1$, (1) implies that several instances of the same operation cannot be scheduled in parallel. Consequently, the set of operation instances in process at a time step $t \in \mathbb{N}$ is the set $A(t) = \{i \in \{1, \ldots, n\} | \exists q \in \mathbb{N}, \sigma_i^q = t\}$. The resource constraint can be written as follows:

$$\sum_{i \in A(t)} b_i^s \leq m_s, \quad \forall s \in \{1, \ldots, k\}, \forall t \in \mathbb{N}$$

Consider now a "generic" time step $\tau \in \{0, \ldots, \lambda - 1\}$ and the set $B(\tau) = \{i \in \{1, \ldots, n\} | \exists q \in \mathbb{N}, \sigma_i = \tau + q\lambda\}$ of operations having their start time σ_i equal to τ modulo λ. Thanks to the schedule periodicity, the resource constraint can be also simplified as it can be shown that there always exists $\mathcal{T} \in \mathbb{N}$ such that for $t \geq \mathcal{T}$, we have $A(t) = B(\tau)$ and for each $t < \mathcal{T}$ we have $A(t) \subset B(\tau)$ where $t = \tau + q\lambda$. The resource constraints can then be replaced by the following "modulo" resource constraints:

$$\sum_{i \in B(\tau)} b_i^s \leq m_s, \quad \forall s \in \{1, \ldots, k\}, \forall \tau \in \{0, \ldots, \lambda - 1\} \tag{3}$$

Finally the RCMSP aims at finding a schedule $\sigma \in \mathbb{N}^n$ that satisfies constraints (2-3) and minimizes the period λ.

2.2 Example

To illustrate the resource-constrained modulo scheduling problem that occurs in software pipeline, a sample C program and the corresponding ST200 VLIW processor operations are given in Figure 1. ([DAA08],p.270).

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

```
L?__0_8:
LDH_1   g131 = 0, G127
MULL_2  g132 = G126, g131
ADD_3   G129 = G129, g132
ADD_4   G128 = G128, 1
ADD_5   G127 = G127, 2
CMPNE_6 b135 = G118, G128
BRF_7   b135, L?__0_8
```

Figure 1: A sample C program and the corresponding ST200 operations.

The dependence graph corresponding to the example in Figure 1 is displayed in Figure 2. Each node represents a generic operation and each arc represents a dependence. Pairs of values for (θ_i^j, ω_i^j) are displayed close to their arc. A dummy node 0 represents the start of schedule and a dummy node $n+1=8$ represents the end of the schedule.

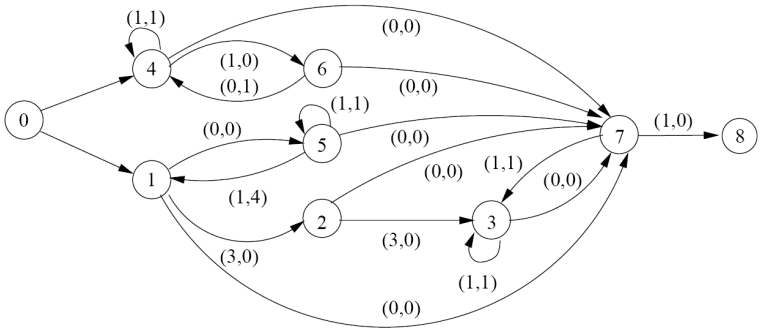

Figure 2: Generic dependencies of the modulo scheduling problem.

In Table 1 we display the resource[1] availabilities and the resource requirements of each operation class[2].

[1]The resources are: Issue for the instruction issue width; MEM for the memory access unit; and CTL for the control unit. An artificial resource LANE0 is also introduced to satisfy some encoding constraints.

[2]There are in our example, 10 classes of operations. ALU, MUL, MEM and CTL correspond repectively to the arithmetic, multiply, memory and control operations. The classes ALUX, MULX and MEMX represent the operations that require an extended

Resource	ISSUE	MEM	CTL	ODD	EVEN	LANE0
Availability	4	1	1	2	2	2
ALL	4	0	0	0	0	0
ALU	1	0	0	0	0	0
ALUX	2	0	0	1	1	0
CTL	1	0	1	1	1	0
ODD	1	0	0	1	0	0
ODDX	2	0	0	1	1	0
MEM	1	1	0	0	0	0
MEMX	2	1	0	1	1	0
PSW	1	1	1	1	1	0
EVEN	1	0	0	0	1	0

Table 1: Resources availabilities and operation requirements.

Figure 3 displays, on a Gantt chart, a modulo schedule in which each operation is suffixed by the instance number. The operations of the local schedule are highlighted.

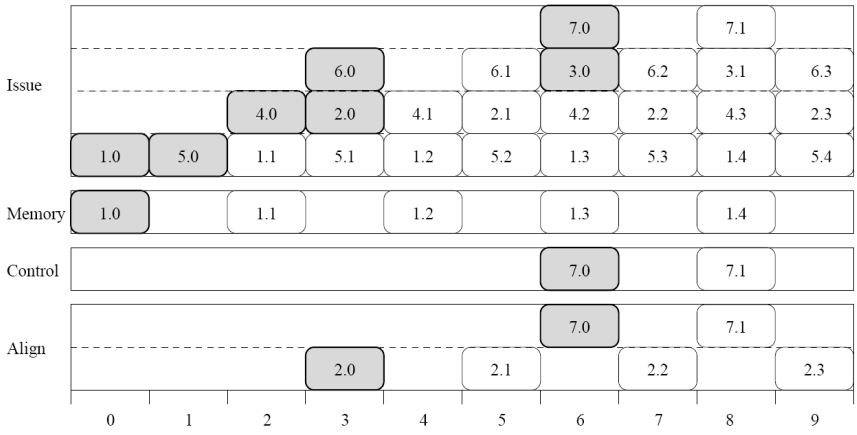

Figure 3: 1-periodic schedule with period $\lambda = 2$.

immediate operand. LDH, MULL, ADD, CMPNE and BRF belong respectively to classes MEM, MUL, ALU and CTL.

2.3 Lower bounds on the optimal period

Without resource constraints, the modulo scheduling problem is polynomially solvable. However, finding a modulo schedule of minimal period λ is known to be $\mathcal{NP}$-hard when resources are limited [HM94].

If we assume infinite availability of resources, the optimal period of a schedule is defined by

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

where $L(C)$ and $H(C)$ are respectively the sum of latencies and the sum of distances of arcs along the circuit C. It is well known (see. [HM94]) that $\lambda^{opt} \geq \lambda^{\infty}$. This bound is due to precedence constraints only and can be computed in polynomial time using a critical circuit algorithm.

On the other hand, a bound due to resource constraints only can be defined by

$$\lambda^{res} = \max_{1 \leq s \leq k} \frac{\sum_{i=1}^{n} b_i^s}{m_s}.$$

That is the minimum such that the renewable resources are not over-subscribed and it can be easily proven that $\lambda^{opt} \geq \lambda^{res}$.

In the rest of the paper we shall denote by $\lambda_{min} = \max(\lambda^{res}, \lambda^{\infty})$, the largest of the two above-defined lower bounds.

2.4 Architecture and data

In this paper, we made our experiments on a real benchmark of graphs issued from the ST200 compiler with real ressource requirements, where the smallest instance *"gsm-st231.10.rcms"* has 10 operations and 42 dependence edges, and the biggest one *"gsm-st231.18.rcms"* has 214 operations and 1063 edges.

Firstly, we made our experiments on a real VLIW architecture (ST200 of STMicroelectronics) with 6 functional units (resources) whose availabilities and operation demands are described in Table 1 Then, we considered the instances presented in [AA10] obtained from the above-defined instances by setting to 10 the availability of each resource and, for each operation, by randomly generating a resource demand vector whose components are chosen in the interval $\{1, \cdots, 10\}$.

Table 2 presents, for each original instance from the ST200 compiler, the number of operations #operations, the number of dependence constraints #prec, the precedence-based lower bound λ^{prec}, the resource-based lower

bound for the original instances λ^{res} (ind) and the resource-based lower bound for the modified instances λ^{res} (mod). One observes the resource constraints are much tighter for the modified instances.

The tests were performed on an INTEL(R) Core(TM) 2 DUO 1.99GHz RAM.

In our experimental framework,

- two ILP formulations, presented in section 3, are tested: Eichenberger et al. Decomposed formulation [ED97] and Dupont-de-Dinechin et al. Direct Formulation (FDI) [dD04]. We applied CPLEX 11 to solve the different ILP formulations.

- different DSP algorithms are introduced in section 4 and tested considering several retiming policies (Gasperoni and Schwiegelshohn's approach [GS94], longest path in the pattern minimization [CDR98b], zero weighted edges minimization and combined longest path and zero weighted edges minimizations [DH00a]) and different acyclic scheduling strategies (As early as possible scheduling, a list with critical paths as priorities, a list with Zinder-Roper [ZR98] priorities).

- a new algorithm, that combines the two previous approachs, is introduced in section 5.

We notice that this is the first study that compares and combines ILP formulations and DSP approximation algorithms for RCMSP.

Table 2: Characteristics of the industrial and modified instances

Instance name	#operations	#prec	λ_{prec}	λ^{res} (ind)	λ^{res} (mod)
adpcm-st231.1	86	405	11	21	52
adpcm-st231.2	142	722	38	35	82
gsm-st231.1	30	190	24	12	16
gsm-st231.2	101	462	12	26	59
gsm-st231.5	44	192	4	11	26
gsm-st231.6	30	130	3	7	17
gsm-st231.7	44	192	4	11	28
gsm-st231.8	14	66	1	8	9
gsm-st231.9	34	154	28	8	21
gsm-st231.10	10	42	1	4	6
gsm-st231.11	26	137	20	6	16
gsm-st231.12	15	70	1	8	10
gsm-st231.13	46	210	19	11	27
gsm-st231.14	39	176	5	10	20
gsm-st231.15	15	70	1	8	9
gsm-st231.16	65	323	4	16	38
gsm-st231.17	38	173	8	9	23
gsm-st231.18	214	1063	8	53	120
gsm-st231.19	19	86	2	8	12
gsm-st231.20	23	102	3	6	13
gsm-st231.21	33	154	18	8	20
gsm-st231.22	31	146	18	8	18
gsm-st231.25	60	273	10	16	37
gsm-st231.29	44	192	4	11	28
gsm-st231.30	30	130	3	7	16
gsm-st231.31	44	192	4	11	26
gsm-st231.32	32	138	15	8	21
gsm-st231.33	59	266	4	15	33
gsm-st231.34	10	42	2	4	7
gsm-st231.35	18	80	2	6	12
gsm-st231.36	31	143	2	10	18
gsm-st231.39	26	118	3	8	15
gsm-st231.40	21	103	2	10	12
gsm-st231.41	60	315	2	18	33
gsm-st231.42	23	102	3	6	14
gsm-st231.43	26	115	8	6	15

3 ILP formulations for the RCMSP

For the non periodic resource-constrained project scheduling (RCPSP), Pritsker *et al.* [PWW69] proposed an ILP formulation based on time-indexed binary variables z_i^t such that $z_i^t = 1$ if and only if the start time of operation i is equal to t. According to the periodic nature of the RCMSP, there are two ways for extending this model, yielding the two ILP models described in this section. To obtain easily linear constraints, both formulations suppose that λ is fixed. Hence the minimum λ for which the corresponding ILP is feasible is the desired optimum period. In our experiments, we simply perform a linear search by iteratively solving the ILP starting with $\lambda = \lambda_{min}$. Note that this solving scheme allows to easily integrate a secondary objective. Although, in this paper, only period minimization is considered, the ILP are in fact solved with an objective function set to the weighted sum of the operation start times. If w_i denote the weight of an operation i, the objective is $min \sum_{i=1}^{n} w_i \sigma_i$. As a preamble, Ayala and Artigues [AA10] have shown that, with this objective, the two formulations presented below are equivalent in the sense they give the same LP relaxation lower bound.

3.1 Direct formulation [dD04, Dup07]

Dupont-de-Dinechin [dD04, Dup07] proposed a time-indexed formulation based on a direct discretization of start times σ_i. This formulation is based on binary variables x_i^t such that $x_i^t = 1$ if and only if $\sigma_i = t$ and we have $\sigma_i = \sum_{t=0}^{T-1} t x_i^t$, where T is any upper bound of the makespan allowing to achieve the optimum λ. For ease of notation, we suppose there is an integer Q such that $T = Q\lambda$ (we can always increase T as needed to obtain this property). This formulation (direct) is expressed as follows:

$$min \sum_{i=1}^{n} w_i (\sum_{t=0}^{T-1} t x_i^t) \tag{4}$$

$$\sum_{t=0}^{T-1} x_i^t = 1 \quad \forall i \in \{1, \ldots, n\} \tag{5}$$

$$\sum_{t=0}^{T-1} t x_i^t + \theta_i^j - \lambda \omega_i^j \leq \sum_{t=0}^{T-1} t x_j^t, \quad \forall (i,j) \in E \tag{6}$$

$$\sum_{i=1}^{n} \sum_{q=0}^{T/\lambda - 1} x_i^{\tau + q\lambda} b_i^s \leq m_s, \forall \tau \in \{0, \ldots, \lambda - 1\}, s \in \{1, \ldots, k\} \tag{7}$$

$$x_i^t \in \{0,1\} \quad \forall i \in \{1,\ldots,N\}, \forall t \in \{0,\ldots,T-1\} \tag{8}$$

As already mentionned, objective (4) is minimization of the weighted sum of the operation start times. Constraints (5) state that each generic operation has to be started exactly once in $\{0,\ldots,T-1\}$. Constraints (7) ensure that the usage of a resource never exceeds its availability. Here, set $B(\tau)$ is the set of operations such that $x_i^t = 1$ for any t such that $t = \tau + q\lambda$ with $q \in \{0,\ldots,\frac{T}{\lambda} - 1 = Q-1\}$.

Inspired by the results of Christophides *et al* for the RCPSP [CAVT87], Dupont-de-Dinechin [dD04, Dup07] introduced the following so-called "disaggregated" precedence constraints:

$$\sum_{h=t}^{T-1} x_i^h + \sum_{h=0}^{t+\theta_i^j-\lambda\omega_i^j-1} x_j^h \leq 1, \forall t \in \{0,\ldots,T-1\}, \forall (i,j) \in E \tag{9}$$

As in the preceding case, replacing constraints (6) by constraints (9) yields a tighter formulation (direct+). The proof can be extended from the results obtained for the RCPSP (see e.g. [SBJ99]).

3.2 Decomposed formulation [ED97]

The start time of the generic operation i can be decomposed according to the division by λ. We have $\sigma_i = \tau_i + \alpha_i\lambda$ with $\tau_i \in \{0,\ldots,\lambda-1\}$ and $\alpha_i \in \mathbb{N}$.

Following this decomposition, Eichenberger and Davidson [ED97] introduce integer variables α_i and binary variables y_i^τ such that, $y_i^\tau = 1$ if only if $\tau_i = \tau$ which yields also $\tau_i = \sum_{\tau=0}^{\lambda-1} \tau y_i^\tau$. The formulation (decomp) is expressed as follows:

$$\min \sum_{i=1}^{n} w_i (\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + \alpha_i\lambda) \tag{10}$$

$$\sum_{\tau=0}^{\lambda-1} y_i^\tau = 1, \quad \forall i \in \{1,\ldots,n\} \tag{11}$$

$$\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + \alpha_i\lambda + \theta_i^j - \lambda\omega_i^j \leq \sum_{\tau=0}^{\lambda-1} \tau y_j^\tau + \alpha_j\lambda, \quad \forall (i,j) \in E \tag{12}$$

$$\sum_{i=1}^{n} y_i^\tau b_i^s \leq m_s, \quad \forall s \in \{1,\ldots,k\}, \forall \tau \in \{0,\ldots,\lambda-1\} \tag{13}$$

$$y_i^\tau \in \{0,1\} \quad \forall i \in \{1,\ldots,n\}, \forall \tau \in \{0,\ldots,\lambda-1\} \tag{14}$$

$$\alpha_i \in \{0, \ldots, Q-1\} \quad \forall i \in \{1, \ldots, n\} \tag{15}$$

As for the direct formulation, by replacing σ_i by $\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + \alpha_i \lambda$, objective (10) represent the weighted sum of operation start times. Constraints (12) are the precedence constraints (2). Constraints (11) state that each generic operation has to be started exactly once in the period or, equivalently, that the remainder of the division of σ_i by λ lies in $\{0, \ldots, \lambda-1\}$. With this decomposition, set $B(\tau)$ is precisely the set of operations such that $y_i^\tau = 1$, which directly gives resource constraints (13) from original modulo resource constraints (3).

Based on results obtained by Chaudhuri *et al.* [CWM94], Eichenberger and Davidson [ED97] propose a new precedence constraint, they call "structured" precedence constraint.

$$\sum_{x=\tau}^{\lambda-1} y_i^x + \sum_{x=0}^{(\tau+\theta_i^j-1) \mod \lambda} y_j^x + \alpha_i - \alpha_j \leq \omega_i^j - \lfloor \frac{\tau + \theta_i^j - 1}{\lambda} \rfloor + 1,$$
$$\forall \tau \in \{0, \ldots, \lambda-1\}, \forall (i,j) \in E \tag{16}$$

Replacing constraints (12) with constraints (16) yields a tighter formulation (decomp+) (see [ED97] for the proof).

3.3 Experimental study

We now evaluate the performance of the standard (direct+) and (decomp+) formulations on the industrial and modified instances for obtaining optimal solutions. First, remark that a lower bound of the optimal period can be obtained by finding the minimal λ for wich the LP relaxation is feasible. Theoretical results from [AA10] established that the (direct) and (decomposed) ILP yield the same lower bound, as well as (direct+) and (decomposed+). Unfortunately, the computational experiments carried out in [AA10] also showed that for the same industrial and modified instances as the one considered in the present paper, the lower bound obtained this way never exceeds λ_{min}, the trivial lower bound. A better lower bound can be obtained by the new formulation proposes in [AA10] but this formulaiton has an exponential number of variables and cannot be used for direct ILP solving.

For integer solving, we use here the (direct+) and (decomp+) formulations. Starting with the trivial lower bound on the period λ_{min}, the branch-and-bound of the ILP solver is used, incrementing the period until a feasible

solution is found which yields the optimal period. Tables 13 and 15 shows the optimal of best known integer solutions obtained by (direct+) and (decomp+).

Tables 3, 4, 5 and 6 show a classification of instances according to the time of execution and the size of the instance.

Table 3: Formulation decomp+ industrial instances

♯operat	♯instances	ms	sec	min	$> 1h$	No[1]
0 - 30	16	14	2	-	-	-
31 - 60	15	7	3	4	1	-
61 - 100	2	-	-	-	2	-
> 100	3	-	-	-	2	1
Total	36	21	5	4	5	1

Table 4: Formulation direct+ industrial instances

♯operat	♯instances	ms	sec	min	$> 1h$	No[1]
0 - 30	16	14	2	-	-	-
31 - 60	15	7	2	5	1	-
61 - 100	2	-	-	-	2	-
> 100	3	-	-	-	2	1
Total	36	21	4	5	5	1

Table 5: Formulation decomp+ modified instances

♯operat	♯instances	ms	sec	min	$> 1h$	No[1]
0 - 30	16	3	9	4	-	-
31 - 60	15	1	2	9	2^3	1
61 - 100	2	-	-	-	-	2
> 100	3	-	-	-	-	3
Total	36	4	11	13	2	6

Table 6: Formulation direct+ modified instances

♯operat	♯instances	ms	sec	min	$> 1h$	No[1]
0 - 30	16	3	8	5	-	-
31 - 60	15	-	3	9	-	3
61 - 100	2	-	-	-	-	2
> 100	3	-	-	-	-	3
Total	36	3	11	14	0	8

Table 13 shows that all industrial ST200 instances except one (*gsm-st231.18*) can be solved to optimality. The (decomp+) formulation is faster than the (direct+) formulation. However, if some instances are solved very quickly, more than 161 hours are necessary to solve instance *adpcm-st231.2*. For the most part solved industrial instances, the optimal period is equal to the trivial lower bound λ_{min}. In just one case (instance "adpcm-st231.2") the optimal period is not equal to the trivial lower bound.

Table 15 shows the optimal of best known integer solutions obtained by (direct+) and (decomp+) on the modified instances. It appears that the modified instances are harder to solve as less optimal solutions are found. There are 6 instances whose optimal solution is not found after three weeks of time run. We have two instances for which, after three weeks of time run, a feasible but not optimal solution is found (with gaps of 1.75% and 8%). The hardess of the modified instances, compared to the industrial instances, is also reflected by the fact that, for the optimally solved modified instances, the minimal period is now larger than the trivial lower bound.

4 Approximation algorithms

Among the software pipelining algorithms, a guaranteed approach, called Decomposed Software Pipelining (DSP), has been proposed by Gasperoni and Schwiegelshohn [GS94], followed by the retiming method by Calland, Darte and Robert [CDR98b] to solve the problem for parallel processors and ordinary precedence. The main idea of DSP is to decouple the problem into dependence constraints and resource constraints so as to decompose the problem into two subproblems: a cyclic scheduling problem ignoring resource constraints: **loop shifting**, and a standard acyclic graph for which efficient techniques are known: **loop compaction**. Initialy defined on parallel iden-

[1]Not solved instances.

tical processors, this approach has been extended to pipelined processors with latencies in [DH00a]. The worst case analysis of the extended algorithm is given in [BH09b] which provides a performance guarantee when arbitrary list schedule is used for loop compaction in the acyclic stage.

4.1 Loop shifting

The idea behind DSP is to define a shift for each operation to extract an acyclic subgraph of the initial uniform precedence graph and to schedule it using any non-cyclic scheduling algorithm. The acyclic schedule provided by the algorithm is then moved, using the initial shift, to a periodic schedule which period is the makespan of the acyclic schedule. [CDR98b] propose to use retiming as a shift and they define a **legal retiming** by:

$$\mathcal{R} : V \to \mathbb{Z}, \quad \forall (O_i, O_j) \in E,\ \mathcal{R}_j + \omega_i^j - \mathcal{R}_i \geq 0$$

The intuition behind retiming is that (O_i, q), which corresponds to the $(q+1)^{th}$ execution of the first instance $(O_i, 0)$, can also be interpreted as the $(q + 1 + \mathcal{R}_i)^{th}$ execution of a new first operation $O'_i = (O_i, -\mathcal{R}_i)$. Changing the definition of first occurences of operations allows to interleave operations of different iterations into the new first iteration. Precedence relations also move: if (O_i, q) precedes $(O_j, q+\omega_i^j)$ then (O'_i, q) precedes $(O'_j, q+\mathcal{R}_j+\omega_i^j-\mathcal{R}_i)$. So the value $\mathcal{R}_j + \omega_i^j - \mathcal{R}_i$ is the distance of a new cyclic precedence relation.

Moreover, if $\mathcal{R}_j + \omega_i^j - \mathcal{R}_i = 0$ then (O'_i, q) precedes (O'_j, q) for enough large integer q. Otherwise, (O'_i, q) precedes an occurrence (O'_j, q') with $q' > q$. Hence for these new generic operations $(O'_i)_{1 \leq i \leq n}$, the first iteration fulfils the precedence relations given by a graph called $G^{\mathcal{R}}$ computed from G by keeping only the arcs for which $\mathcal{R}_j + \omega_i^j - \mathcal{R}_i = 0$. Notice that $G^{\mathcal{R}}$ is acyclic since G has no zero height circuit.

Several ideas have been investigated to find a legal retiming: Gasperoni and Schwiegelsohn [GS94] approach can be reformulated as finding a legal retiming associated to a periodic schedule assuming unlimited resources. In [CDR98b], where using retiming for loop shifting is formalized, the authors consider two optimizations:

- the length of the longest path in $G^{\mathcal{R}}$ minimization
- the number of edges in $G^{\mathcal{R}}$ minimization, so as to reduce the number of precedence constraints for loop compaction

For the first objective, such retiming can be computed in polynomial time using the retiming algorithm due to Leiserson and Saxe [LS88] for clock-period minimization. Huard and Darte [DH00a] extended this algorithm to precedence latency problems.

For the second optimization, in [CDR98b] an ILP formulation was formulated to solve this problem. Then, [DH00a] defines a polynomial purely graph algorithm inspired by a minimal cost flow algorithm proposed by Fulkerson, known as out-of-kilter method [GM85]. An algorithm that computes retiming which combines longest path and zero weighted edges minimizations was also proposed to solve this problem.

4.2 Loop compaction

The idea behind DSP approach is to choose a particular retiming $\mathcal{R}$, use a guaranteed algorithm to get a schedule π of $G^{\mathcal{R}}$, and then to extend the guarantee to the induced periodic schedule.

List scheduling algorithms are the most used heuristics for scheduling with precedence and resource constraints. A simple list scheduling algorithm consist on scheduling operations as early as possible in order to meet resource constraints. To each operation, we can give a label to reflect the scheduling priority of that operation. Then, a list algorithm with priorities builds a solution by scheduling at each time the heighest priority operation among a set of concurrent operations ready to be issued. An intuitive priority would be the longest path in the acyclic graph. Zinder and Roper [ZR98] proposes to compute priorities taking into acount also the resouce constraints. The mean idea is to compute the priority of each operation using the schedule, that meets the precedence and resource constraints, of its successors. This algorithm was initially introduced for parallel processors systems and extended to pipelined processors in [BH10].

4.3 DSP algorithm

To formalize DSP approach, let $\mathcal{R}$ be a legal retiming of G and π be any (non cyclic) schedule of $G^{\mathcal{R}}$ that fulfills the resource constraints as well as the precedences induced by $G^{\mathcal{R}}$. We note π_i the start time of operation O_i in this schedule. In fact, a slightly lower value of $\lambda^{\mathcal{R}}$ can be computed as follows: If $C^{\mathcal{R}}_{max} = \max_{O_i \in V} \pi_i + 1$ and

$$I^{\mathcal{R}}_{max} = \max_{(O_i,O_j)\in G\setminus G^{\mathcal{R}}} \frac{\pi_i - \pi_j + \theta^j_i}{\mathcal{R}_j - \mathcal{R}_i + \omega^j_i},$$

we denote

$$\lambda^{\mathcal{R}} = \max(C^{\mathcal{R}}_{max}, I^{\mathcal{R}}_{max}).$$

Then, for any integer $q \in \mathbb{N}$ setting

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

we have the following result:

Lemma 1 *For any feasible retiming $\mathcal{R}$, $\sigma(\mathcal{R})$ is a feasible periodic schedule of G with period $\lambda^{\mathcal{R}}$.*

Proof. First, we prove that, at any time slot t, the operations scheduled at t in $\sigma(\mathcal{R})$ meet the resource constraints. We note $F_t = \{O_i | q \in \mathbb{N}, \sigma^q_i(\mathcal{R}) = t\}$ the set of operations whose occurrences are scheduled at t. Let O_i and O_j be two operations in F_t. We note q and q' their corresponding occurrences issued in time slot t. Hence,

$$\begin{array}{rcl} t = \sigma^q_i(\mathcal{R}) & = & \sigma^{q'}_j(\mathcal{R}) \\ \pi_i + (q + \mathcal{R}_i)\,\lambda^{\mathcal{R}} & = & \pi_j + (q' + \mathcal{R}_j)\,\lambda^{\mathcal{R}} \end{array}$$

$$\pi_i - \pi_j - \left(\mathcal{R}_j - \mathcal{R}_i + q' - q\right)\lambda^{\mathcal{R}} = 0.$$

Since $\pi_i < C^{\mathcal{R}}_{max} \leq \lambda^{\mathcal{R}}$ and similarly $\pi_j < \lambda^{\mathcal{R}}$, $-\lambda^{\mathcal{R}} < \pi_i - \pi_j < \lambda^{\mathcal{R}}$ and then,

$$\pi_i = \pi_j \text{ and } \mathcal{R}_i + q = \mathcal{R}_j + q'.$$

Hence, O_i and O_j are performed on the same time slot π_i in the acyclic schedule π. Thus, $O_j \in F_{\pi_i}$ and then $F_t \subseteq F_{\pi_i}$. Since π fulfills the resource constraints induced by $G^{\mathcal{R}}$, F_{π_i} (and then F_t) meets the resource constraints.

For precedence constraints, we need to prove that, for any arc $(O_i, O_j) \in E$ and any integer $q \in \mathbb{N}$:

$$\begin{array}{rcl} \sigma^q_i(\mathcal{R}) + \theta^j_i & \leq & \sigma^{q+\omega^j_i}_j(\mathcal{R}) \\ \Leftrightarrow \pi_i + (q + \mathcal{R}_i)\,\lambda^{\mathcal{R}} + \theta^j_i & \leq & \pi_j + \left(q + \mathcal{R}_j + \omega^j_i\right)\lambda^{\mathcal{R}} \end{array}$$

Hence, we have to verify that the following inequality is satisfied for each arc (O_i, O_j)

$$\pi_i - \pi_j + \theta^j_i \leq \left(\mathcal{R}_j - \mathcal{R}_i + \omega^j_i\right)\lambda^{\mathcal{R}}.$$

We have two cases: either (O_i, O_j) is kept in $G^{\mathcal{R}}$ or not.

- In the first case, $\mathcal{R}_j - \mathcal{R}_i + \omega_i^j = 0$. Since π fulfills the precedence constraints induced by $G^{\mathcal{R}}$, we have $\pi_i + \theta_i^j \leq \pi_j$. Then the inequality is satisfied.

- In the second case, $\mathcal{R}_j - \mathcal{R}_i + \omega_i^j > 0$ and by definition, $\lambda^{\mathcal{R}} \geq I_{max}^{\mathcal{R}} = \max_{(O_i,O_j)\in G\setminus G^{\mathcal{R}}} \frac{\pi_i - \pi_j + \theta_i^j}{\mathcal{R}_j - \mathcal{R}_i + \omega_i^j}$. Thus the inequality is available in this case too.

which achieves the proof. ■

Now to illustrate the DSP approach, we consider the following generic algorithm by using a list algorithm to produce π.

Algorithm 1: Extended DSP

1. Find a legal retiming $\mathcal{R}$ for G;

2. **for** $(O_i, O_j) \in E$ **do**
   if $\mathcal{R}_j - \mathcal{R}_i + \omega_i^j = 0$ **then**
   keep (O_i, O_j) in $G^{\mathcal{R}}$;

3. Perform a list scheduling on $G^{\mathcal{R}}$ coping with both precedence and resource constraints. Compute π_i the start time of operation O_i in this schedule and $\lambda^{\mathcal{R}}$;

4. Define the cyclic schedule $\sigma(\mathcal{R})$ by:
   for $1 \leq q \leq N$ **do**
   for $O_i \in V$ **do**
   $\sigma_i^q(\mathcal{R}) = \pi_i + \lambda^{\mathcal{R}}(q + \mathcal{R}_i)$;

A legal retiming for G of Figure 2 is given in Table 7. The only arcs of G remaining in $G^{\mathcal{R}}$ are (O_6, O_4) and (O_3, O_7).

Operations	O_1	O_2	O_3	O_4	O_5	O_6	O_7
$\mathcal{R}_i$	0	1	2	0	1	1	2

Table 7: A retiming $\mathcal{R}$ of G.

A schedule built by Algorithm 1 with period $\lambda^{\mathcal{R}} = 3$ is depicted in Figure 4. The two arcs of $G^{\mathcal{R}}$ are shown.

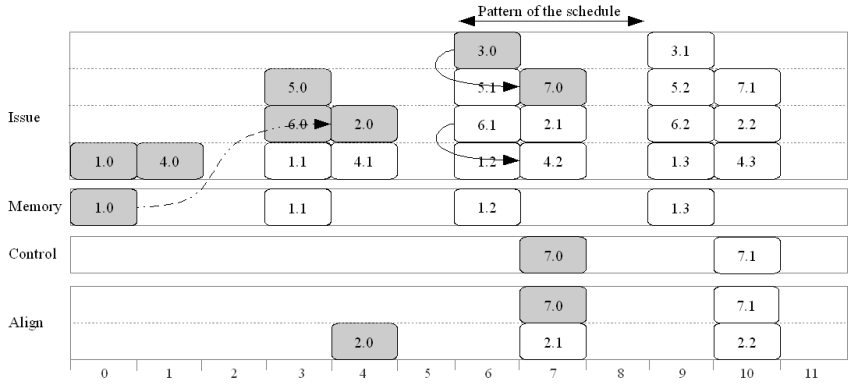

Figure 4: A periodic schedule generated by Algorithm 1 with $\lambda^{\mathcal{R}} = 3$.

We remark that, in this cyclic schedule, there are some idle cycles. In [BH09b] the following worst case bound is established when one of the retimings described in section 4.1 is applied for loop shifting and a list algorithm is considered for loop compaction:

Theorem 1

$$\lambda^{DSP} \leq \left(k + 1 - \frac{1}{B_{max} \cdot \theta_{max}}\right) \lambda^{opt} + \left(1 - \frac{1}{B_{max} \cdot \theta_{max}}\right) (\theta_{max} - 1)$$

where B_{max} and θ_{max} denote respectively the maximum capacity of each resource and the maximum precedence latency.

Notice that [DH00a] studied the worst case behavior only on identical parallel processors and this is the first worst case performance bound produced for pipelined processors.

This theoretical worst case ratio is quite large, and could discourage people to use such approach in industrial applications. Thus, in the next section, we propose to study the efficency of different DSP algorithms, extending those given in the litterature, from an experimental point of view.

4.4 Experimental study

The first experimental observations for shifting algorithms are depicted in the following table:

Model / Shifting	ST200 ϵ_{moy}	ST200 $\epsilon = 0$	RCCSP ϵ_{moy}	RCCSP $\epsilon = 0$
GS	0.42	69%	0.44	69%
Clock	0.58	75%	0.33	78%
MinEdge	1.44	44%	0.17	89%
ClockEdge	0.78	64%	0.25	83%

Table 8: Average distances to the best period given by DSP algorithms.

For ST200 architecture, the first and the second optimise the critical part of the acyclic graph and give the best period for more than 80% of instances. However, the third and the fourth retimings are more efficient for random resource demands. This observation confirms the conclusion of [DH00b], that zero weighted edges minimization is efficient mainly when resources are very limited, while Gasperoni and Schwiegelsohn heuristic give good results in the opposite case.

For the acyclic scheduling level, using priorities like longest path or Zinder-Roper priorities gives significantly better results than a simple as early as possible algorithm. We compared compaction using this two algorithms without any shifting and their results were almost similar (*see.* Table 9), with a slight advantage of longest path priorities whose running time is smaller.

Model / Compaction	ST200 ϵ_{moy}	ST200 $\epsilon = 0$	RCCSP ϵ_{moy}	RCCSP $\epsilon = 0$
$liste_0$	0.72	75%	0.78	64%
$liste_1$	0.14	95%	0.22	78%
$liste_2$	0.06	95%	0.14	89%

Table 9: Comparaison of list algorithms for loop compaction.

As these algorithms are relatively fast, one can properly design a platform implementing them all and choosing the minimal period among the computed values.

Now we consider the best period given by approximation algorithms considering only the instances solved by ILP formulations in order to compare the two approaches. An extract of the corresponding results is given in Table

10.

Model	ST200	RCCSP
Tests reaching λ^{opt}	30/36[1]	20/28[2]
Average ratio to λ^{opt}	1.03	1.04
Maximal ratio to λ^{opt}	1.33	1.21
Average distance to λ^{opt}	0.25	0.82
Maximal distance to λ^{opt}	3	5

Table 10: Comparaison of DSP solutions to optimal periods computed by ILP.

We first evaluated the ratio between the period given by DSP algorithms and the optimal period given by ILP formulation. With a value not greater than 1.3, it appears that the practical worst case ratio is much lower than the theoritical one. Also for the random ressource allocations, the average ratio is below 1.05. Moreover, we can see from these statistics that the optimal period is reached for more than 82% of instances. For the other instances, except some critical ones, the distance to the optimal period is equal to one. This proves that decomposed software pipelining is still an interesting technique even with complex resource and precedence constraints.

But like any heuristic, DSP algorithms can fall into traps and, for some of our instances, they might give a period with five time slots more than the optimal period.

In conclusion, we notice that ILP formulations may induce very long execution times and could not solve some instances of the problem in a reasonable time. On the other hand, DSP approach is very efficient in general, but can have a relatively large deviation from the optimal for critical instances. However there are opportunities for improvement, and in the next section, we present a new algorithm to overcome the weaknesses of this two approachs.

[1]For (*gsm-st231.18*), which is not solved by ILP, the period given by DSP is equal to λ_{min}. Therefore, it is optimally solved and we counted it in the statistics.

[2]The number 28 is the number of optimally solved instances by ILP formulations for RCCSP madel.

5 The hybrid approach

In this section, we propose to combine exact method and approximation algorithms to exploit the good properties by applying them to problems they can efficiently solve and to avoid as much as possible their respective defects.

5.1 Hybrid algorithm

The mean idea is to use an optimal algorithm for loop compaction instead of an approximation algorithm. So we propose to use the results of the first level of DSP approach and then the Eichenberger *et al.* formulation described in section 3.2 is applied for the second level. We get the following algorithm:

Algorithm 2: The hybrid approach

1. Find a legal retiming $\mathcal{R}$ for G;
2. Compute $\lambda^{\mathcal{R}}$ by solving the following system

$$min \sum_{i=1}^{n} w_i (\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + \mathcal{R}_i \lambda)$$

$$\sum_{\tau=0}^{\lambda-1} y_i^\tau = 1, \forall i \in [1, n]$$

$$\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + \mathcal{R}_i \lambda + \theta_i^j - \lambda \omega_i^j \leq \sum_{\tau=0}^{\lambda-1} \tau y_j^\tau + \mathcal{R}_j \lambda, \forall (O_i, O_j) \in E$$

$$\sum_{i=1}^{n} y_i^\tau b_i^s \leq m_s, \forall s \in \{1, \ldots, k\}, \tau \in [0, \lambda)$$

$$y_i^\tau \in \{0, 1\}$$

3. Define the cyclic schedule $\sigma(\mathcal{R})$ by:

for $1 \leq q \leq N$ **do**
  for $O_i \in V$ **do**
    $\sigma_i^q(\mathcal{R}) = \pi_i + \lambda^{\mathcal{R}}(q + \mathcal{R}_i)$;

Algorithm 2 replaces the third step of Algorithm 1 by the computation

of an exact solution by ILP instead of a list algorithm. Thus it computes for each operation O_i, the corresponding retiming $\mathcal{R}_i$ by one of DSP algorithms. Then, these values are injected in Eichenberger *et al* formulation to replace the variables α_i (period in which the first occurrence of operation O_i is placed in the schedule). Notice that unlike the ILP approach described in section 3, once a legal retiming is given, the minimization of the period λ can be solved directly by integer linear programming, without a second criteria and binary search for the minimal λ.

5.2 Experimental study

This algorithm is computed for the architectures described above and the corresponding statistics are given in Table 11.

	Model	ST200					RCCSP				
	CPU	*ms*	*sec*	*min*	$> 1h$	No[1]	*ms*	*sec*	*min*	$> 1h$	No[2]
♯Oper	♯Instances										
10 - 30	16	16	-	-	-	-	16	-	-	-	-
31 - 60	15	15	-	-	-	-	13	-	2	-	-
61 - 100	2	2	-	-	-	-	-	-	1	1	-
> 100	3	2	1	-	-	-	-	-	-	1	2
Total	36	35	1	0	0	0	29	0	3	2	2

Table 11: Distribution of instances depending on theirs sizes and computing times by the hybrid approach.

Compared to ILP formulation results given in Tables 13 and 15, we notice that there is a significant decreasing in computation time and more instances are solved.

[1]Not solved instances.

Model	ST200	RCCSP
Unsolved tests	0/36	2/36
Tests with $\lambda = \lambda^{opt}$	36/36	20/28
Mean ratio λ/λ^{opt}	1	1.03
Maximal ratio λ/λ^{opt}	1	1.21
Mean distance $\lambda - \lambda^{opt}$	0	0.62
Maximal distance $\lambda - \lambda^{opt}$	0	5

Table 12: Experimental efficiency of the Hybrid approach.

For ST200 architecture, all tests reach the optimal solution, even for the critical instance that is not solved by ILP formulation, as the period given by the hybrid approach is equal to the resource bound.

For the random resource allocation architecture (modified instances), the improvement is seen mainly in the computation time and in the number of instances solved by this approach compared to ILP formulations. However, there is almost no difference compared to list algorithms results given in Table 10. The hybrid approach improves the DSP period by one time slot for 4 instances, and for the remaining ones, the period is equal to that given by list algorithms. This proves that list algorithms give very efficient results for the acyclic level of DSP. On the other hand, the limited efficiency of this approach in this case may be due to the fact that the retiming does not take into account the criticality of resources which is more important for this architecture. This points a little weakness of DSP approach which is based on the separation into two levels: retime then compact. So, even if we rely on an optimal algorithm in the second level, the choice of the retiming may not be the best one for loop compaction.

6 Concluding remarks

This paper reports an experimental study of integer linear programming (ILP) formulations and Decomposed Software Pipelining (DSP) algorithms to solve the resource-constrained modulo scheduling problem. The experiments were ran on a set of real instances as well as randomly modified instances to reach more variability on resource usage. Although experiments had been made separately on both techniques, it is the first time that the results of DSP algorithms are compared to the real optimum. The experiments indicate that ILP formulation cannot solve all the problem in-

stances in reasonable execution time, and that DSP may fail in some cases to find a really close to optimal solution although the performance of this approach is very good in terms of execution time and mean deviation to optimal. To overcome the drawbacks of the two algorithms, we proposed a new hybrid algorithm based on a combination of the retiming approach and of ILP formulation. The methods solves fastly the set of industrial instances to optimality and obtain good solutions on the set of harder instances with random resource demands. The approach shows however some limits that suggest that an integration of resource constraints during the retiming phase as well as the design of a dedicated branch and bound replacing ILP for the second phase are promising research directions.

Acknowledgement

The authors wish to warmly thank Benoit Dupont de Dinechin for making the ST200 instances available and also for this helpful advices on modulo scheduling.

References

[AA10] M. Ayala and C. Artigues. On integer linear programming formulations for the resource-constrained modulo scheduling problem. Technical Report 10393, LAAS-CNRS, Toulouse, France, 2010.

[AJLA95] Vicki H. Allan, Reese B. Jones, Randall M. Lee, and Stephen J. Allan. Software pipelining. *ACM Comput. Surv.*, 27(3):367–432, 1995.

[BdDH06] F. Blachot, B. Dupont de Dinechin, and G. Huard. A heuristic for near-optimal software pipelining. In *Euro-Par 2006 Parallel Processing. Lecture Notes in Computer Science*, 2006.

[BDM+99] Peter Brucker, Andreas Drexl, Rolf Mohring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, January 1999.

[BH09a] A. Benabid and C. Hanen. Decomposed software pipelining for cyclic unitary rcpsp with precedence delays. In *MISTA 09: Pro-*

ceedings of the 4th Multidisciplinary International Scheduling Conferance, 2009.

[BH09b] Abir Benabid and Claire Hanen. Modulo scheduling for unitary resource constrained cyclic problems. In *ROADEF*, 2009.

[BH10] A. Benabid and C. Hanen. Minimizing lateness for precedence graphs with constant delays on dedicated pipelined processors. In *ISCO: International Symposium on Combinatorial Optimization*, 2010.

[CAVT87] N. Christofides, R. Alvarez-Valdés, and J. Tamarit. Project scheduling with resource constraints: a branch and bound approach. *European Journal of Operational Research*, 29:262–273, 1987.

[CDR98a] P.-Y. Calland, A. Darte, and Y. Robert. Circuit retiming applied to decomposed software pipelining. *IEEE Transactions on Parallel and Distributed Systems*, 9(1):24–35, 1998.

[CDR98b] Pierre-Yves Calland, Alain Darte, and Yves Robert. Circuit retiming applied to decomposed software pipelining. *IEEE Trans. Parallel Distrib. Syst.*, 9(1):24–35, 1998.

[CWM94] S. Chaudhuri, R.A. Walker, and J.E. Mitchell. Analyzing and exploiting the structure of the constraints in the ilp approach to the scheduling problem. *IEEE Transactions on Very Large Scale Integration Systems*, 2:456–471, 1994.

[DAA08] B. Dupont De Dinechin, C. Artigues, and S. Azem. Resource-constrained moulo scheduling. In C. Artigues, Sophie Demassey, and E. Néron, editors, *Resource Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, Control Systems, Robotics and Manufacturing Series, pages 267–277. ISTE-WILEY, 2008. ISBN 978-1-84821-034-9.

[dD04] B. Dupont de Dinechin. From machine scheduling to vliw instruction scheduling. *ST Journal of Research*, 1(2), 2004.

[DH00a] Alain Darte and Guillaume Huard. Loop shifting for loop compaction. *Lecture Notes in Computer Science*, 1863/2000:415–431, 2000.

[DH00b] Alain Darte and Guillaume Huard. Loop shifting for loop compaction. *Int. J. Parallel Program.*, 28(5):499–534, 2000.

[Dup07] B. Dupont de Dinechin. Time-indexed formulations and a large neighborhood search for the resource-constrained modulo scheduling problem. In P. Baptiste, G. Kendall, A. Munier-Kordon, and F. Sourd, editors, *3rd Multidisciplinary International Scheduling conference: Theory and Applications*. 2007.

[ED97] A. Eichenberger and E.S. Davidson. Efficient formulation for optimal modulo schedulers. *SIGPLAN - PLDI'97*, 1997.

[EDA94] A. Eichenberger, E. S. Davidson, and S. G. Abraham. Minimum register requirements for a modulo schedule. In *27th Annual International Symposium on Microarchitecture*, 1994.

[GM85] M. Gondran and M. Minoux. *Graphes et algorithmes*. Eyrolles, 1985.

[GS94] Franco Gasperoni and Uwe Schwiegelshohn. Generating close to optimum loop schedules on parallel processors. *Parallel Processing Letters*, 4:391–403, 1994.

[HM94] Claire Hanen and Alix Munier. Cyclic scheduling on parallel processors: An overview. In Philippe Chrétienne, Edward G. Coffman, Jan Karel Lenstra, and Zhen Liu, editors, *Scheduling theory and its applications*. J. Wiley and sons, 1994.

[LS88] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *NASA STI/Recon Technical Report N*, 89:17797–+, 1988.

[PWW69] A. Pritsker, L. Watters, and P. Wolfe. Multi-project scheduling with limited resources: a zero-one programming approach. *Management Science*, 16:93–108, 1969.

[Rau93] B. Ramakrishna Rau. Dynamically scheduled vliw processors. In *MICRO 26: Proceedings of the 26th annual international symposium on Microarchitecture*, pages 80–92, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[Rau94] B. Ramakrishna Rau. Iterative modulo scheduling: an algorithm for software pipelining loops. In *MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture*, pages 63–74, New York, NY, USA, 1994. ACM.

[RS02] S. Rajagopalan and Malik. Sharad. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. CRC Press, 2002.

[SBJ99] J. Sankaran, D. Bricker, and S. Juang. A strong fractional cutting-plane algorithm for resource-constrained project scheduling. *International Journal of Industrial Engineering*, pages 99–111, 1999.

[SMD04] M. Smelyanskiy, S. Mahlke, and E.S. Davidson. Probabilistic predicate -aware modulo scheduling. In *International symposium on Code generation and optimization: feedback-directed and runtime optimization*, 2004.

[ZR98] Y. Zinder and D. Roper. An iterative algorithm for scheduling unit-time operations with precedence constraints to minimise the maximum lateness. *Annals of Operations Research*, 81:321–340, 1998.

Table 13: Optimal integer solutions of the industrial instances

Instances	PLNE(decomp+)		PLNE(direct+)	
	λ	CPU_s	λ	CPU_s
adpcm-st231.1	21	14400	21	16235
adpcm-st231.2	40	582362	40	601000
gsm-st231.1	24	0.05	24	0.05
gsm-st231.2	26	79362	26	83991
gsm-st231.5	11	0.05	11	0.05
gsm-st231.6	7	17	7	20
gsm-st231.7	11	0.05	11	0.05
gsm-st231.8	8	0.05	8	0.05
gsm-st231.9	28	0.05	28	0.05
gsm-st231.10	4	0.05	4	0.05
gsm-st231.11	20	0.05	20	0.05
gsm-st231.12	8	0.05	8	0.05
gsm-st231.13	19	1856	19	2023
gsm-st231.14	10	301.25	10	478
gsm-st231.15	8	0.05	8	0.05
gsm-st231.16	16	7520	16	8156
gsm-st231.17	9	0.05	9	0.05
gsm-st231.18	-	-	-	-
gsm-st231.19	8	0.05	8	0.05
gsm-st231.20	6	0.05	6	0.05
gsm-st231.21	18	0.05	18	0.05
gsm-st231.22	18	0.05	18	0.05
gsm-st231.25	16	3652	16	4001
gsm-st231.29	11	12.6	11	15
gsm-st231.30	7	12	7	15
gsm-st231.31	11	47	11	73
gsm-st231.32	15	0.05	15	0.05
gsm-st231.33	15	2365	15	2503
gsm-st231.34	4	0.05	4	0.05
gsm-st231.35	6	0.05	6	0.05
gsm-st231.36	10	27	10	42
gsm-st231.39	8	0.05	8	0.05
gsm-st231.40	10	0.05	10	0.05
gsm-st231.41	18	2356	18	2562
gsm-st231.42	6	0.05	6	0.05
gsm-st231.43	8	0.05	8	0.05

Table 14: Optimal of feasible integer solutions of the modified instances (random resource demands)

Instances	PLNE(decomp+)		PLNE(direct+)	
	λ	CPU_s	λ	CPU_s
adpcm-st231.1	-	-	-	-
adpcm-st231.2	-	-	-	-
gsm-st231.1	25	250	25	375
gsm-st231.2	-	-	-	-
gsm-st231.5	36	280	36	299.03
gsm-st231.6	27	152	27	265
gsm-st231.7	41	92	41	115
gsm-st231.8	12	0.27	12	0.31
gsm-st231.9	32	0.56	32	60
gsm-st231.10	8	0.10	8	0.11
gsm-st231.11	24	0.37	24	0.39
gsm-st231.12	13	12.65	13	19
gsm-st231.13	43	985.03	43	1236
gsm-st231.14	33	220	33	252
gsm-st231.15	12	12.36	12	13
gsm-st231.16	-	-	-	-
gsm-st231.17	33	90	33	105
gsm-st231.18	-	-	-	-
gsm-st231.19	15	38.23	15	43
gsm-st231.20	20	123	20	137
gsm-st231.21	30	42.03	30	59
gsm-st231.22	29	80.36	29	112
gsm-st231.25	56 (Gap=1.75%)	604800	-	-
gsm-st231.29	42	210	42	513
gsm-st231.30	25	58	25	67
gsm-st231.31	39	142	39	169
gsm-st231.32	30	0.25	30	1.01
gsm-st231.33	52(Gap=8%)	604800	-	-
gsm-st231.34	7	5.05	7	8
gsm-st231.35	14	52	14	53
gsm-st231.36	24	230	24	403
gsm-st231.39	21	95	21	168
gsm-st231.40	17	15	17	29
gsm-st231.41	-	-	-	-
gsm-st231.42	18	12	18	17
gsm-st231.43	20	15	20	23

7 Appendix I - Experimental result tables

Table 15: Periods given by DSP algorithms

Instances	Industrial instances			Modified instances		
	λ^{DSP}	$\lambda^{DSP}/\lambda^{opt}$	$\lambda^{DSP} - \lambda^{opt}$	λ^{DSP}	$\lambda^{DSP}/\lambda^{opt}$	$\lambda^{DSP} - \lambda^{opt}$
adpcm-st231.1	21	1	0	80	-	-
adpcm-st231.2	41	1,03	1	139	-	-
gsm-st231.1	24	1	0	30	1,2	5
gsm-st231.2	26	1	0	93	-	-
gsm-st231.5	11	1	0	36	1	0
gsm-st231.6	8	1,14	1	27	1	0
gsm-st231.7	11	1	0	41	1	0
gsm-st231.8	8	1	0	12	1	0
gsm-st231.9	28	1	0	32	1	0
gsm-st231.10	4	1	0	8	1	0
gsm-st231.11	20	1	0	24	1	0
gsm-st231.12	8	1	0	13	1	0
gsm-st231.13	19	1	0	43	1	0
gsm-st231.14	10	1	0	34	1,03	1
gsm-st231.15	8	1	0	12	1	0
gsm-st231.16	16	1	0	59	-	-
gsm-st231.17	12	1,33	3	33	1	0
gsm-st231.18	53	1	0	194	-	-
gsm-st231.19	8	1	0	15	1	0
gsm-st231.20	6	1	0	20	1	0
gsm-st231.21	18	1	0	30	1	0
gsm-st231.22	18	1	0	29	1	0
gsm-st231.25	16	1	0	55	-	-
gsm-st231.29	11	1	0	42	1	0
gsm-st231.30	8	1,14	1	25	1	0
gsm-st231.31	11	1	0	39	1	0
gsm-st231.32	15	1	0	30	1	0
gsm-st231.33	15	1	0	52	-	-
gsm-st231.34	4	1	0	8	1,14	1
gsm-st231.35	6	1	0	16	1,14	2
gsm-st231.36	10	1	0	29	1,21	5
gsm-st231.39	8	1	0	23	1,1	2
gsm-st231.40	10	1	0	17	1	0
gsm-st231.41	19	1,06	1	50	-	-
gsm-st231.42	6	1	0	19	1,06	1
gsm-st231.43	10	1,11	1	23	1,15	3

Table 16: Periods given by the hybrid approach for industrial instances

Instances	HD retiming		GS retiming			
	λ^{hyb}	CPU_s	λ^{hyb}	CPU_s	$\lambda^{hyb}/\lambda^{opt}$	$\lambda^{hyb} - \lambda^{opt}$
adpcm-st231.1	21	0.0007	21	0.0020	1	0
adpcm-st231.2	40	0.0012	41	15.0000	1	0
gsm-st231.1	24	0.0001	24	0.0020	1	0
gsm-st231.2	26	0.0001	26	0.0020	1	0
gsm-st231.5	11	0.0001	11	0.0020	1	0
gsm-st231.6	7	0.0000	7	0.0020	1	0
gsm-st231.7	11	0.0001	11	0.0020	1	0
gsm-st231.8	8	0.0000	8	0.0020	1	0
gsm-st231.9	28	0.0001	28	0.0020	1	0
gsm-st231.10	4	0.0000	4	0.0020	1	0
gsm-st231.11	20	0.0001	20	0.0020	1	0
gsm-st231.12	8	0.0001	8	0.0020	1	0
gsm-st231.13	19	0.0001	19	0.0020	1	0
gsm-st231.14	10	0.0001	10	0.0020	1	0
gsm-st231.15	8	0.0000	8	0.0020	1	0
gsm-st231.16	16	0.0001	16	0.0020	1	0
gsm-st231.17	9	0.0001	12	0.0020	1	0
gsm-st231.18	53	45.0000	53	900.0000	1	0
gsm-st231.19	8	0.0000	8	0.0020	1	0
gsm-st231.20	6	0.0000	6	0.0020	1	0
gsm-st231.21	18	0.0001	18	0.0020	1	0
gsm-st231.22	18	0.0001	18	0.0020	1	0
gsm-st231.25	16	0.0002	16	0.0020	1	0
gsm-st231.29	11	0.0001	11	0.0020	1	0
gsm-st231.30	7	0.0000	7	0.0020	1	0
gsm-st231.31	11	0.0000	11	0.0020	1	0
gsm-st231.32	15	0.0000	15	0.0020	1	0
gsm-st231.33	15	0.0000	15	0.0020	1	0
gsm-st231.34	4	0.0000	5	0.0020	1	0
gsm-st231.35	6	0.0000	6	0.0020	1	0
gsm-st231.36	10	0.0000	10	0.0020	1	0
gsm-st231.39	9	0.0000	8	0.0020	1	0
gsm-st231.40	10	0.0000	10	0.0020	1	0
gsm-st231.41	18	0.0000	18	0.0020	1	0
gsm-st231.42	6	0.0000	6	0.0020	1	0
gsm-st231.43	9	0.0000	11	0.0020	1	0

Table 17: Periods given by the hybrid approach for modified instances

Instances	HD retiming		GS retiming			
	λ^{hyb}	CPU_s	λ^{hyb}	CPU_s	$\lambda^{hyb}/\lambda^{opt}$	$\lambda^{hyb} - \lambda^{opt}$
adpcm-st231.1	79	1 day	-	-	-	-
adpcm-st231.2	-	-	-	-	-	-
gsm-st231.1	29	0.0030	28	0.0050	1.12	3
gsm-st231.2	93	1 day	-	-	-	-
gsm-st231.5	?	?	?	?	?	?
gsm-st231.6	27	0.0020	27	0.0050	1	0
gsm-st231.7	41	0.0020	41	0.0050	1	0
gsm-st231.8	12	0.0020	12	0.0050	1	0
gsm-st231.9	32	0.0020	34	0.0050	1	0
gsm-st231.10	8	0.0020	8	0.0050	1	0
gsm-st231.11	24	0.0020	24	0.0050	1	0
gsm-st231.12	13	0.0020	13	0.0050	1	0
gsm-st231.13	43	0.0050	43	5 min	1	0
gsm-st231.14	34	0.0020	34	0.0050	1.03	1
gsm-st231.15	12	0.0020	12	0.0050	1	0
gsm-st231.16	59	600.0000	59	20 min	-	-
gsm-st231.17	33	0.0020	33	0.0050	1	0
gsm-st231.18	-	-	-	-	-	-
gsm-st231.19	15	0.0020	15	0.0050	1	0
gsm-st231.20	20	0.0020	20	0.0050	1	0
gsm-st231.21	30	0.0020	30	0.0050	1	0
gsm-st231.22	29	0.0020	29	0.0050	1	0
gsm-st231.25	56	1 h	56	32 min	-	-
gsm-st231.29	42	0.0020	42	0.0050	1	0
gsm-st231.30	25	0.0020	25	0.0050	1	0
gsm-st231.31	39	0.0020	39	0.0050	1	0
gsm-st231.32	30	0.0020	30	0.0050	1	0
gsm-st231.33	52	15min	52	25 min	-	-
gsm-st231.34	8	0.0020	8	0.0050	1.14	1
gsm-st231.35	16	0.0020	16	0.0050	1.14	2
gsm-st231.36	29	0.0020	29	0.0050	1.21	5
gsm-st231.39	23	0.0020	23	0.0050	1.1	2
gsm-st231.40	17	0.0020	17	0.0050	1	0
gsm-st231.41	50	0.0020	50	2 min	-	-
gsm-st231.42	19	0.0020	19	0.0050	1.06	1
gsm-st231.43	23	0.0020	23	0.0050	1.15	3