Cyclic DataFlows in computers and embedded systems

Claire C. Hanen, Alix Munier-Kordon

Chapter 1
Cyclic DataFlows in computers and embedded systems

Claire Hanen and Alix Munier-Kordon

1.1 Introduction

This chapter addresses cyclic scheduling problems issued from the control of data flows in computers, embedded systems or sensor networks. Although in various context, parts and data may induce the same theoretical scheduling problems, we focus here on specific models and constraints. We point out analogies with production scheduling as well as differences and show the main basic results of the field, following the introduction on cyclic scheduling given in [39], chapters 5, 6 and 8.

Dealing with data flows instead of manufacturing process means that tasks/jobs represent computation and/or data transmission. Precedence constraints are here induced by data dependencies: a job can be processed only when its input data, produced or carried by another task, is available. Notice that in a manufacturing process, a part is usually transformed, assembled, but remains in the system, although a computation task may create or delete data. Precedence constraints may also be defined when a limited memory constraint is considered. Indeed, a job J_i that has to write a data in a full memory or buffer has to wait that another one, say J_i, frees place, inducing then a precedence relation from executions of J_i to J_j. These constraints are frequently considered in embedded systems for which the overall available memory is limited.

Computations are done by physical components that, from a scheduling point of view are similar to usual processors or machines in production process though parallel processors or more complex resources from RCPSP problems are usually used [3]. However, energy saving may induce unusual constraints on the scheduling

Claire Hanen
Sorbonne Université, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, Paris, France,
e-mail: Claire.Hanen@lip6.fr

Alix Munier-Kordon
Sorbonne Université, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, Paris, France,
e-mail: Alix.Munier@lip6.fr

process, in particular grouping of tasks processed by the same component, in order to avoid too many on/off.

We consider in this chapter a finite set of jobs $\mathscr{J} = \{J_i, \quad 1 \leq i \leq n\}$ which communicate data following a Synchronous DataFlow Graph formalism. Synchronous DataFlow Graph (SDF in short) is a simple model of computation introduced by Lee and Messerschmitt[29] for the description of Digital Signal Processing Applications. In this context, SDF or extensions were considered to model H263 Encoder [9], an MP3 playback [36] or a Reed-Solomon decoder [5]. The SDF obtained do not exceed here more that height actors. SDF associated to an application may also be generated automatically using a DataFlow language [40, 22]. The number of actors for real-life applications ranges up to 600. The size of the instances encountered in this new generation of embedded systems is significantly larger than before as they express increasingly higher levels of concurrency.

Each jobs of a fixed SDF has to be executed repeated infinitely. Thus, checking the feasibility of a SDF or evaluating its maximum throughput can be seen has a cyclic scheduling problem. One of the aim of this chapter is to investigate the relationship between cyclic scheduling problems and Dataflow problems. The main questions that these two communities have explored are the following:

- *Schedulability*: does a feasible infinite schedule exist?
- *Evaluation of the maximum throughput*: what is the structure and the performance of a schedule that maximizes the throughput?
- *Performance of a periodic schedule*: what is the optimal cycle time of a schedule with a specific periodic structure?
- *Memory optimization*: what is the minimum amount of memory to reach feasibility? or a given cycle time?

We propose in next sections, a panorama of theoretical results developed for dataflow models in connection with cyclic scheduling problems. Section 1.2 is dedicated to the presentation of the SDF model. Basic results about the precedence constraints and the normalization are recalled, leading to the definition of a feasible schedule and its normalized average cycle time. Two small examples, namely a loop parallelization problem and the modelling of periodic data transfer for a real-time system are presented in Section 1.3. Section 1.4 presents some basic results and optimization problems for the special case of uniform precedence graph, which is particular important class of SDFs. Section 1.5 is dedicated to the presentation of basic mathematical properties on SDF and two important optimization problems. Section 1.6 is our conclusion.

1.2 Synchronous dataFlow graphs

This section presents some basic definitions and results on Synchronous Dataflow Graphs. Subsection 1.2 introduces the general model and the repetition vector. Next subsection recalls that a SDF models an infinite set of precedence relations between

the successive executions of the jobs. Subsection 1.2.3 presents the normalization of a SDF. This transformation will be useful to study the schedulability and the determination of a periodic schedule in section 1.5. We lastly presents some common criteria and scheduling policies.

1.2.1 General Model

Let us consider a set of n jobs $\mathscr{J} = \{J_1, \ldots, J_n\}$ with processing times $\{p_1, \ldots, p_n\}$ to be repeated many times. For $J_i \in \mathscr{J}$, $<J_i,k>$ denotes the k^{th} occurrence of J_i. Jobs are usually supposed to be totally or partially non-reentrant, *i.e.* two successive executions of a job may not overlap or for all $n > 0$, $<J_i,n+1>$ starts at least one time unit after $<J_i,n>$ starts.

Jobs can exchange data using FIFO (First-In First Out) queues. Each FIFO has exactly one input job J_i and one output job J_j and is thus modelled by an arc $a = (J_i,J_j)$. Arcs are usually bi-valued by two strictly positive integers $u(a)$ and $v(a)$ with the assumptions that:

1. $u(a)$ data (or tokens) are stored in the FIFO at the completion of each execution of J_i;
2. $v(a)$ data are removed from the FIFO before each execution of J_j. If there is not enough data, the job cannot be executed and must wait for them.

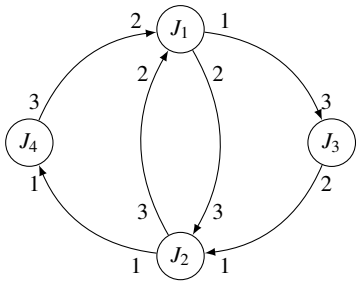
Let A be the set or arcs. $M_0(a)$ for each arc $a \in A$ is a non negative integer corresponding to the initial number of data in the associated buffer. These values can be fixed at the beginning, or may be variable for some optimization problems. A Synchronous DataFlow Graph (in short SDF) is then a tri-valued multi-graph $G = (\mathscr{J},A,u,v,M_0)$. Let $\mathscr{C}(G)$ denotes the set of circuit of G.

A schedule is a function $s : \mathscr{J} \times \mathbb{N}^\star \to \mathbb{R}^+$ such that $s(J_i,k)$ is the starting time of $<J_i,k>$. A schedule is feasible if at any instant the number of tokens in any FIFO is non negative.

Consider for example the SDF of $n = 4$ jobs $\mathscr{J} = \{J_1,J_2,J_3,J_4\}$ depicted by Figure 1.1. We also suppose that $p_1 = 1$, $p_2 = 2$, $p_3 = 1$ and $p_4 = 2$. Figure 1.2 presents the first executions of the earliest schedule of the SDF of Figure 1.1.

Let us define the weight of any circuit c of G by $W(c) = \Pi_{a \in c} \frac{u(a)}{v(a)}$. Note that, if $W(c) > 1$, the number of data items stored in the FIFO will increase as far as the jobs are executed. In the contrary, if $W(c) < 1$, this numbers tends to 0, leading to a deadlock. These situations correspond to design flaws and such graphs can be dismissed. Thus, all studies are restrained to **unitary graphs** for which the weight of every circuit c is 1, *i. e*, $\forall c \in \mathscr{C}(G), W(c) = 1$.

Suppose that, at time instant t, J_i was executed n_i times, with $n_i > 0$ and that J_j was executed n_j times with $n_j > 0$. Then, the total number tokens at t stored in the buffer associated to arc a equals $M_0(a) + u(a)n_i - v(a)n_j$. Thus, we observe that if $n_i = k \times v(a)$ and $n_j = k \times u(a)$, the number of tokens in the queue equals $M_0(a)$. More formally, the following theorem is proved in [29]:



$a \in A$	$u(a)$	$v(a)$	$M_0(a)$
$a_1 = (J_1, J_3)$	1	3	0
$a_2 = (J_3, J_2)$	2	1	1
$a_3 = (J_1, J_2)$	2	3	1
$a_4 = (J_2, J_1)$	3	2	3
$a_5 = (J_4, J_1)$	3	2	5
$a_6 = (J_2, J_4)$	1	1	0

Fig. 1.1: A Synchronous Data Flow Graph G

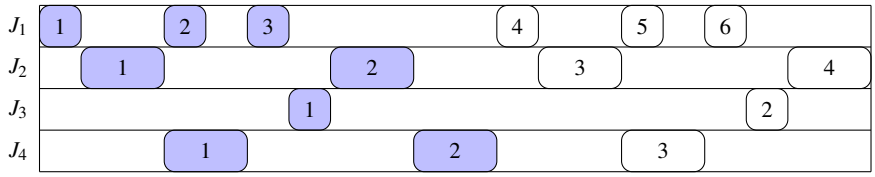


Fig. 1.2: First executions of the earliest schedule of the SDF pictured by Figure 1.1

Theorem 1.1 (Repetition vector). *Suppose that G is a unitary SDF. Then, there exists an integer vector $N \geq 1^n$ such that or for every arc $a = (J_i, J_j) \in A$, the equality $u(a) \times N_i = v(a) \times N_j$ holds. Then, the graph is feasible iff each job $J_i \in \mathscr{J}$ can be executed at least N_i times. Moreover, once each job is executed exactly N_i times (if it is possible), the systems returns in its initial state, ı.e the current marking of the buffers equals its initial value.*

We can check that our example pictured by Figure 1.1 is unitary. The equations verified by the repetition vectors are $N_1 = 3N_3$, $2N_3 = N_2$, $2N_1 = 3N_2$, $N_2 = N_4$ and $3N_4 = 2N_1$. The smallest integer solution is then $N_1 = 3, N_2 = 2, N_3 = 1$ and $N_4 = 2$.

1.2.2 Precedence constraints associated to a SDF and useful tokens

A precedence constraint between executions $<J_i, n_i>$ and $<J_j, n_j>$ with $(n_i, n_j) \in \mathbb{N}^{\star 2}$ expresses that $<J_j, n_j>$ cannot be executed before the completion of $<J_i, n_i>$. Munier [34] proved that each arc $a = (J_i, J_j)$ is equivalent to an infinite set of precedence relations between the successive executions of J_i and J_j defined the following theorem. A proof can also be found in [32].

Theorem 1.2 (Precedence constraints associated with a FIFO queue). *Let J_i and J_j be two re-entrant jobs. A FIFO queue $a = (J_i, J_j) \in A$ with initially $M_0(a)$ tokens models a precedence relation between the n_ith execution of J_i and the n_jth execution of J_j iff*

$$u(a) > M_0(a) + u(a) \cdot n_i - v(a) \cdot n_j \geq \max\{u(a) - v(a), 0\}.$$

For example, let us consider the arc $a = (J_1, J_2)$ with $u(a) = 2$, $v(a) = 3$ and $M_0(a) = 1$. The inequality of Theorem 1.2 becomes

$$2 > 1 + 2 \cdot n_i - 3 \cdot n_j \geq 0.$$

The couples of indexes (n_1, n_2) such that there exists a precedence relation due to a are then $\{(1+3k, 1+2k), k \in \mathbb{N}\}$ and $\{(3+3k, 2+2k), k \in \mathbb{N}\}$.

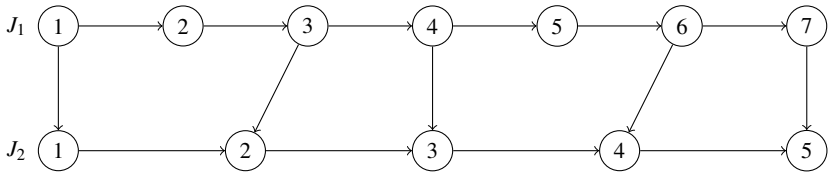


Fig. 1.3: Precedence relations between first executions of J_1 and J_2 and the arc $a = (J_1, J_2)$ with $u(a) = 2$, $v(a) = 3$ and $M_0(a) = 1$. Jobs J_1 and J_2 are supposed to be re-entrant.

A **useful initial marking** is such that, for any arc $a = (J_i, J_j)$, $M_0(a)$ is a multiple of $\gcd(u(a), v(a))$. A corollary of Theorem 1.2 is that useful initial markings are dominant [33, 32]. Moreover the initial marking $M_0(a)$ of a may be replaced by $\left\lfloor \frac{M_0(a)}{\gcd(u(a),v(a))} \right\rfloor \times \gcd(u(a), v(a))$ without any influence on the precedence constraints associated to a. Thus, we only consider useful initial markings.

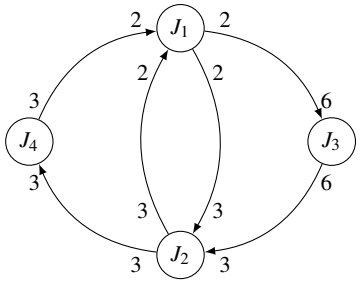
1.2.3 Normalization

Let us assume that a SDF $G = (\mathscr{J}, A, u, v, M_0)$ is a unitary graph. A SDF is said to be **normalized** if there exists a positive integer vector $Z = (Z_1, \cdots, Z_n)$ such that, for any arc $a = (J_i, J_j) \in A$, $u(a) = Z_i$ and $v(a) = Z_i$. Marchetti and Munier [33, 32] proved the following theorem:

Theorem 1.3 (Normalization). *If G is a unitary SDF then, there exists an integer vector $Z \geq 1^{\mathrm{n}}$ such that, for any arc $a = (J_i, J_j)$, $Z_i \times v(a) = Z_j \times u(a)$. It follows that the normalized SDF G' built from G by setting, for any arc $a = (J_i, J_j)$, $u'(a) = Z_i$, $v'(a) = Z_j$ and $M'_0(a) = M_0(a) \times \frac{Z_i}{u(a)}$ generates the same set of precedence constraints as G.*

Theorem 1.3 can be seen as a corollary of Theorem 1.1. Indeed, if the repetition vector N is given, we can get the normalization vector by setting $M = \mathrm{lcm}(N_1, \cdots, N_n)$ and for any job J_i, $Z_i = \dfrac{M}{N_i}$. In the following we only consider normalized SDF.

For example, Figure 1.4 presents the normalized SDF G' associated with the SDF G shows by Figure 1.1 and its initial marking. We get $M = \text{lcm}(2,3) = 6$ and thus $Z_1 = \frac{M}{3} = 2$, $Z_2 = \frac{M}{2} = 3$, $Z_3 = M = 6$ and $Z_4 = \frac{M}{2} = 3$.



$a \in A$	$M_0(a)$
$a_1 = (J_1, J_3)$	0
$a_2 = (J_3, J_2)$	3
$a_3 = (J_1, J_2)$	1
$a_4 = (J_2, J_1)$	3
$a_5 = (J_4, J_1)$	5
$a_6 = (J_2, J_4)$	0

Fig. 1.4: Normalized graph G' associated with G

1.2.4 Uniform precedence graphs

A SDF G is said to be **uniform** if for any arc $a = (J_i, J_j)$, $u(a) = v(a) = 1$. The corresponding inequality of Theorem 1.2 becomes $1 > M_0(p) + n_i - n_j \geq 0$, and thus $n_j - n_i = M_0(a)$. In this case, the corresponding set of precedence constraints between executions of J_i and J_j verifies :

$$\forall n > 0 \quad \big(s(J_i, n) + p_i \leq s(J_j, n + M_0(a))\big).$$

Observe that in this case, $p_i > 0$ and $M_0(a) \geq 0$.

However, uniform precedence graphs can be defined more generally as in [35]. Indeed, in the more general case, the two integer values associated to any arc $a = (J_i, J_j)$ may be negative. A uniform precedence graph is then defined as a bi-valued oriented graph $G = (\mathscr{J}, A, \ell, h)$. The length and the height of an arc are respectively function defined as $\ell : A \to \mathbb{Z}$ and $h : A \to \mathbb{Z}$. The precedence relations associated to any arc $a = (J_i, J_j)$ are then defined by:

$$\forall n \geq \max(1, 1 - h(a)), s(J_i, n) + \ell(a) \leq s(J_j, n + h(a)).$$

Several authors [33, 32] have observed that the precedence relations induced by any unitary SDF can be expressed using a uniform precedence graph for which each job J_i is duplicated N_i times. This transformation, called the **expansion** of the graph, allows to consider all the algorithmic tools developed for uniform precedence graphs to SDF, and thus was extensively used.

Its main drawback is that the size of the expanded graph is not polynomial and may be huge for real-life applications. Indeed, its total number of vertices equals

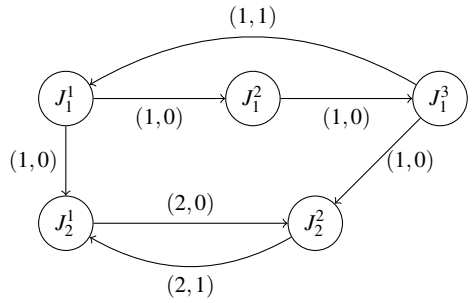


Fig. 1.5: Expansion of the graph composed by two non re-reentrant jobs J_1 and J_2 and the arc $a = (J_1, J_2)$ with $u(a) = 2$, $v(a) = 3$ and $M_0(a) = 1$

$\sum_{i=1}^{n} N_i$ and its number of arcs is around $\sum_{a=(J_i,J_j)\in A} \min(N_i, N_j)$. The consequence is that the methods developed for uniform precedence graphs are not efficient for these instances. However, as we will see in subsection 1.5.1, partial expansions may be considered to develop efficient exact algorithms for the throughput evaluation.

1.2.5 Criteria

Several criteria may be considered to evaluate a feasible schedule s. The most common one is the average cycle time of s, which is the inverse of the throughput. More formally, the **average cycle time** of job J_i for a schedule s is the mean time interval between two executions of J_i:

$$\lambda_i^s = \lim_{k \to +\infty} \frac{s(J_i, k)}{k}.$$

The **normalized average cycle time** of s can be defined then as

$$\lambda^s = \max_{J_i \in \mathscr{J}} \frac{\lambda_i^s}{Z_i}.$$

Another common criteria of a schedule is the **latency** $\mathscr{L}^s$. Roughly speaking, the latency is the maximum delay between a stimulation and the answer of the system. The SDF G must be without circuits. The latency of the entire system is the maximum time gap from a data input of a system to a connected outcome. This criteria is particularly important for real-time systems to measure the worst-case reaction time of a system.

1.2.6 Scheduling policies

A schedule s is said to be **K-periodic** if there exists for any job J_i a period w_i and an integer K_i such that, for n sufficiently large, $s(J_i, n+K_i) = s(J_i, n) + w_i$. K_i is the of J_i, while w_i is its **period**. Note that

$$\lambda_i^s = \frac{w_i}{K_i}.$$

Moreover, if G is strongly connected, the normalized average cycle time is

$$\lambda^s = \frac{w_i}{K_i Z_i}.$$

The most common scheduling policy consists on executing the actors as as soon as possible (asap in short) which maximizes the throughput. An asap schedule always consists of two stages [34]. The first one is an initialization phase which is necessarily finite and possibly null. A K-periodic steady state phase follows. The periodicity factor of job J_i verifies $K_i = \alpha \times N_i$ with $\alpha \in \mathbb{N}^\star$.

The earliest schedule depicted by Figure 1.2 is K-periodic. Values w_i, K_i and λ_i^s are depicted by Table 1.1. The normalized average cycle time equals $\lambda^s = \frac{11}{6}$.

Table 1.1: Parameters of the earliest schedule of Figure1.2

J_i	w_i	K_i	λ_i^s	Z_i
J_1	11	3	$\frac{11}{3}$	2
J_2	11	2	$\frac{11}{2}$	3
J_3	11	1	11	6
J_4	11	2	$\frac{11}{2}$	3

The main drawback of the asap schedule is that its description is not of polynomial size. Indeed, values of the repetition vector are not polynomial and may be huge. Many authors (see as example [31, 8]) restrict their study to periodic schedules in order to get efficient algorithms. The structure of periodic schedules of a SDF is presented in Section 1.5.2

1.3 Modelling Examples

Two usual applications for which SDF and uniform graphs are particularly suitable are presented in this section. The first one concerns a loop parallelization. It has been studied since the early 90's [21, 38, 13, 20] and the introduction of parallel computers. Most of computation time is indeed spent in loops, so that the good use of parallelism allowed by the architecture is crucial. Our second example shows that communications between real-time periodic jobs following Liu and Layland model [30] can be expressed using a particular normalized SDF. This modelling can be used to evaluate the whole latency of the system.

1.3.1 Loop parallelization

Let us describe on an example how a task system associated to the execution of a loop on a specific architecture can be modeled by a uniform task system, provided that enough resources are available.

Assume that arrays a, b are stored in the memory of a computer, and consider the C loop depicted in figure 1.6. We describe the jobs associated with assembly instructions. We assume that all instructions are processed by pipelined units, that can start a new instruction at each time unit, while the execution time till the end of an operation is 2 for additions, 6 for multiplication, and 4 for memory operations (load and store).

```
for (i = 3; i < n; i++){
    b[i] =   a[i-3] + c;
    a[i] += b[i] · d;
}
```

J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
address a[i]	address b[i]	load a[i-3]	add c	store b[i]	mult d	load a[i]	add a[i]	store a[i]

Fig. 1.6: A C loop and its associated jobs

Figure 1.7 shows the uniform constraints induced by the loop semantic as well as the architecture (assuming here unlimited number of functional units). The partial reentrance is modelled by the loops around each job with label $(1,1)$. Although interleaving the iterations is allowed, the storage of $a[i]$ in the memory at iteration i, i.e job $<J_9, i>$, must precede the load of $a[i]$ at iteration $i+3$ (Job $<J_3, i+3>$). Thus the arc (J_9, J_3) has $\ell = 4$ and $h = 3$. Uniform constraints can also model the use of a limited number of buffers. For example, we can assume here that the successive address of $a[i]$ are stored in a buffer of size 3, so that at most three executions of J_3 can start without starting J_1 and J_1 can start only if a register is free, i.e. if J_3 started. This is modeled by the arcs (J_1, J_3) and (J_3, J_1) with values $(2,3)$ and $(0,0)$. When dealing with loop execution on parallel architectures, it is necessary to build a compact schedule, that can be easily described by a finite set of instructions. Hence in this field most authors considered strictly periodic schedules, where all jobs have

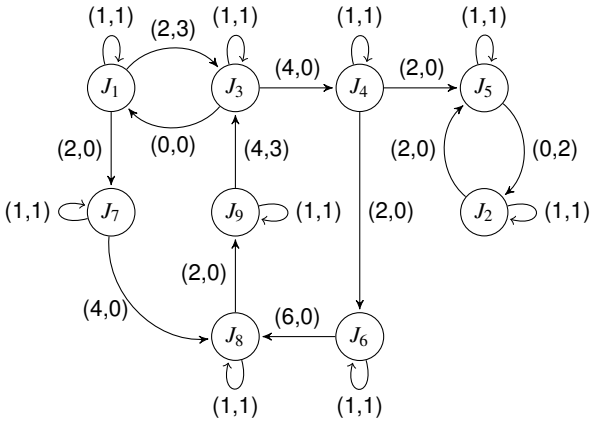


Fig. 1.7: A uniform graph modeling a loop. Arcs are labelled with (ℓ, h)

the same period λ. Figure1.8 shows an optimal periodic schedule for the graph, computed with the techniques described in section 1.4.1.

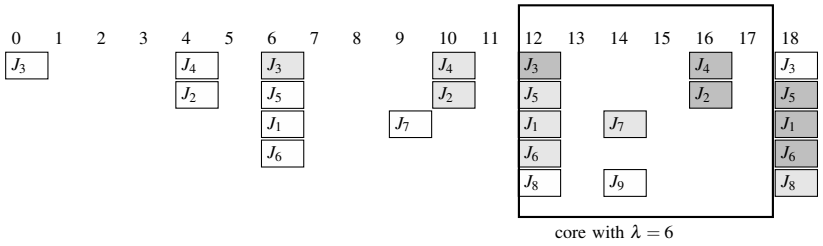


Fig. 1.8: An optimal periodic schedule

1.3.2 Periodic data transfers for a real-time system

Let us consider a set of jobs based on the model of Liu and Layland [30]. Each job J_i is characterized by a period T_i, a processing time C_i, a deadline D_i, and a release date r_i. The nth occurrence of J_i can be processed if and only if its execution start date $s(t_i, n)$ is superior or equal to its release date

$$r_i + (n-1)T_i \leq s(J_i, n).$$

and its execution end date cannot exceed its deadline

$$s(J_i,n)+C_i \leq r_i+(n-1)T_i+D_i.$$

Suppose for example that job J_i needs data from job J_i. We consider that the nth execution of J_i writes a unique data at time $r_i+(n-1)T_i+D_i$ and that the nth execution of J_j reads a unique data at time $r_j+(n-1)T_j$. The data are not stored in a FIFO queue, but in a unique memory. Thus, task J_j may read several time the same data if its period $T_j < T_i$.

For example, consider 3 jobs J_1, J_2 and J_3 which parameters are shown in Table 1.2. We assume that J_1 sends data to J_2 and that J_2 sends data to J_3. Figure 1.9 presents the relations between jobs. For example, $<J_2,2>$ is reading a data from $<J_1,1>$, while $<J_2,4>$ is reading a data from $<J_1,2>$. The data considered by $<J_2,3>$ comes from $<J_1,1>$, the arcs is omitted by transitivity, since $<J_2,2>$ precedes $<J_2,3>$.

Table 1.2: Parameters of jobs J_1, J_2 and J_3

J_i	r_i	T_i	C_i	D_i
J_1	0	30	10	20
J_2	0	20	10	10
J_3	0	40	5	20

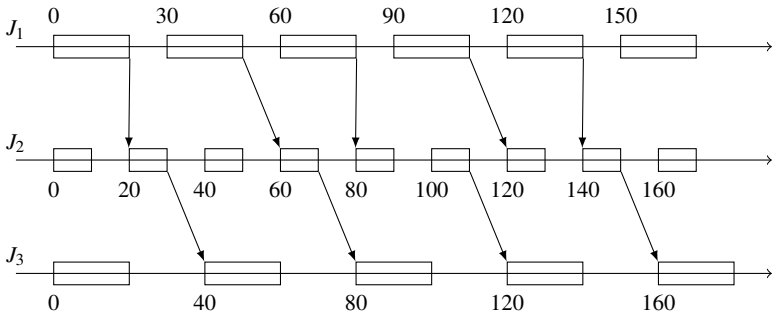


Fig. 1.9: Communications between successive executions of jobs J_1, J_2 and J_3

The question is to compute efficiently the latency of the system. The first problem is then to characterized couples of integers $(n_i,n_j) \in \mathbb{N}^{\star 2}$ such that $<J_j,n_j>$ reads a data from $<J_i,n_i>$.

By studying the lifetime of the data, Khatib et al. [28] observed that the relations between the executions of communicating jobs corresponds to precedence relations of a unitary SDG built following Theorem 1.4:

Theorem 1.4. *Let J_i and J_j be two periodic jobs such that J_i communicates with J_j. The set of communicating instances of jobs J_i and J_i corresponds to prece-*

dence relations of an arc $a = (J_i, J_j)$ of a normalized SDF with $Z_i = T_i$, $Z_j = T_j$ and $M_0(a) = T_j + \alpha - T_a^\star$ with $T_a^\star = \gcd(T_i, T_j)$ and $\alpha = \left\lceil \frac{r_i - r_j + C_i}{T_a^\star} \right\rceil \times T_a^\star$.

Thus, the corresponding SDF is composed by two arcs $a_1 = (J_1, J_2)$ and $a_2 = (J_2, J_3)$ with $Z_1 = 30$, $Z_2 = 20$, $Z_3 = 40$ and the initial markings $M_0(a_1) = 10$ and $M_0(a_2) = 40$. Note that the latency of the graph equals 60. It corresponds to the path $<J_1, 3>$, $<J_2, 5>$, $<J_2, 6>$ and $<J_3, 4>$. Section 1.5.4 is dedicated to the evaluation of the latency of the SDF extracted from a real-time system.

1.4 Uniform precedence graphs

Some fundamental basic results on uniform precedence graphs are firstly recalled. We then introduce a generic technique, called decomposed software pipelining, that was used by several authors to solve periodic scheduling problems with resource constraints and to get approximation results. We finally present constraints recently introduced to handle energy saving in sensor networks and we mention some complexity results.

1.4.1 Basic results

Let consider that $G = (\mathscr{J}, A, \ell, h)$ is a uniform precedence graph G. If no additional resource constraint is considered, the schedulability, the evaluation of the maximum throughput and the performance of a periodic schedule are polynomially solvable.

These questions were initially considered for non-negative uniform case [14], *i.e*, for any arc a, $\ell(a) > 0$ and $h(a) \geq 0$. These results were extended in [35] for any integer values. For the sake of simplicity, we mention here the main results for the case where G is strongly connected. General case can be found in [35, 15].

Let $\mathscr{C}^+(G)$ (*Resp.* $\mathscr{C}^-(G)$) be the set of circuits c of G with $h(c) > 0$ (*Resp.* $h(c) < 0$). For any circuit $\mu \in \mathscr{C}(G)$, let $L(\mu) = \Sigma_{a \in c} \ell(a)$ and $H(\mu) = \Sigma_{a \in c} h(a)$. Let also define the two ratios:

$$\lambda^+(G) = \max_{\mu \in \mathscr{C}^+(G)} \frac{L(\mu)}{H(\mu)}$$

$$\lambda^-(G) = \begin{cases} \min\limits_{\mu \in \mathscr{C}^-(G)} \dfrac{L(\mu)}{H(\mu)} & \text{if } \mathscr{C}^-(G) \neq \emptyset \\ +\infty & \text{otherwise} \end{cases}$$

A circuit $\mu \in \mathscr{C}^+(G)$ is **critical** if $\frac{L(\mu)}{H(\mu)} = \lambda^+(G)$. The critical circuit of the graph depicted in Figure 1.7 is the circuit $\mu = (J_3, J_4, J_6, J_8, J_9, J_3)$ and its value equals $\frac{L(\mu)}{H(\mu)} = \lambda^+(G) = \frac{18}{3} = 6$.

A schedule s is said to be **strictly periodic** if there is a constant λ such that $\forall J_i \in \mathscr{J}\ \forall k > 0 \quad \big(s(J_i,k) = s_i + (k-1)\lambda\big)$. λ is the **average cycle time** of s, also called its **period**. First point of Theorem 1.5 deals with the schedulability. Second point concerns the the evaluation of the maximum throughput while the third point is about the performance of a periodic schedule:

Theorem 1.5 ([39, 35]). *Let G be a uniform strongly connected task system.*

1. *G is feasible if and only $\lambda^+(G) \leq \lambda^-(G)$ and there is no circuit μ in $\mathscr{C}(G)$ with $H(\mu) = 0$ and $L(\mu) > 0$.*
2. *If G is feasible, its minimum average cycle time is $\lambda^+(G)$ and the asap schedule is $K-$periodic.*
3. *If G is feasible, there exists an optimal strictly periodic schedule s with $\lambda^s = \lambda^+(G)$*
4. *Checking feasibility, computing the optimal cycle time and the optimal strictly periodic schedule can be done in polynomial time according to graph algorithms.*

Dasdan et al. [18] have experimentally tested several algorithms to compute the maximum cost to time ratio, which is exactly our problem here. Notice that a fixed value $\lambda \in [\lambda^+(G), \lambda^-(G)]$ iff there is no valued positive cycles in the graph G valued by $V_\lambda(a) = \ell(a) - \lambda.h(a)$ for any arc a. Checking for a positive cycle in a graph can be done in polynomial time using Bellman-Ford algorithm [16]. Howard's algorithm, which is supposed to be the most efficient for the problem, although pseudo-polynomial in the worst case, increases a lower bound b of $\lambda^+(G)$ until the critical circuit is reached or an infeasibility is detected. Another efficient and polynomial approach is a parametric path algorithm with complexity $O(n^4)$ [2, 27].

Dot?

Figure 1.10 shows the graph of Figure 1.7 valued by V_λ for $\lambda = 6$. First execution times $s_i \in \mathscr{J}$ of a feasible strictly periodic schedule of period 6 are also reported.

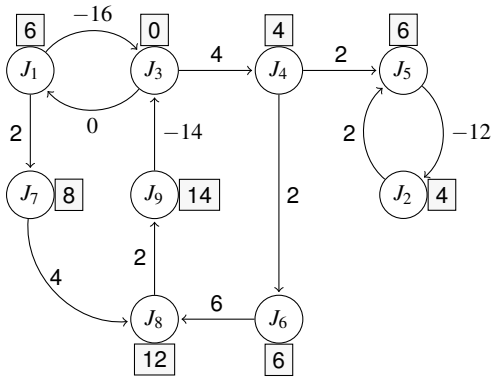


Fig. 1.10: Graph of Figure 1.7 valued by V_λ for $\lambda = 6$. First execution times $s_i \in \mathscr{J}$ of a feasible periodic schedule of period 6 are reported in the squares.

1.4.2 Decomposed software pipelining and resource constrained problems

As seen in section 1.3.1, loop parallelization induces a uniform task system. The architecture on which the loop is executed induces additional resource constraints. From the simple case of parallel processors [24, 25] to the more complex case of RCMSP (resource-constrained modulo scheduling problem), two main approaches have been investigated, in order to find an optimal strictly periodic schedule. Although it can be easily proved that periodic schedules are not dominating schedules, their simple formulation make them very easy to implement and thus often used in loop parallelization context.

Firstly the ILP formulations, for example in [3, 20, 19] combining classical ILP formulations of resource constraints (either time-indexed or not), and linear expression of uniform constraints. In [3], several models are described and experimentally compared. The second approach, known as **decomposed software pipelining (DSP)** is based on the decomposition of the cyclic scheduling problem into two phases, **retiming** and **compaction**, the first one is related to the uniform task system, and the second to non cyclic resource constrained scheduling. In particular, several approximation algorithms have been proposed, based on this ideas [21, 10, 13, 6]. Finally, in [3], a hybrid approach combining shifting and ILP has been investigated.

In this section we describe the decomposed software pipelining technique and summarize the approximation results.

DSP relies on the notion of retiming. The main interest of this technique is to transform a set of uniform constraints into a set of usual precedence constraints, so that the remaining problem is an acyclic scheduling problem with resource constraints.

The intuition behind retiming is that while dealing with periodic schedules, the real iteration number of a job occurrence is not so important. Consider an occurrence $<J_i,k>$, which corresponds to the $(k)^{th}$ execution of the first instance of J_i. It can also be interpreted as the $(k+r_i)^{th}$ execution of $J_i(r_i)$ whose first occurrence is $<J_i,-r_i>$. The height of precedence relations are then changed: if there is a uniform constraint $a=(J_i,J_j)$ labelled by $(\ell(a),h(a))$, then $s(J_i(r_i),k)+\ell(a) \leq s(J_j(r_j),k+r_j+h(a)-r_i)$. So the value $r_j+h(a)-r_i$ is the height of a new uniform precedence relation between $J_i(r_i)$ and $J_j(r_j)$.

Definition 1.1. *A* **legal retiming** *associates to each job J_i an integer value r_i so that:*

$$r : \mathscr{J} \rightarrow \mathbb{Z}, \quad \forall a=(J_i,J_j) \in A \quad \big(r_j+h(a)-r_i \geq 0\big)$$

Now considering a legal retiming, if $r_j+h(a)-r_i=0$ then $<J_i(r_i),k>$ precedes $<J_j(r_j),k>$ for enough large integer k. So that the precedence relations induced by the uniform constraint is now within an iteration of the shifted jobs. Otherwise, $<J_i(r_i),k>$ precedes an occurrence $<J_j(r_j),k'>$ with $k'>k$ which belongs to a next iteration. Hence for these new generic operations $(J_i(r_i))_{1\leq i\leq n}$, the first iteration fulfills the non cyclic precedence relations given by a graph called G^r computed from G by keeping only the arcs $a=(J_i,J_j)$ for which $r_j+h(a)-r_i=0$.

Several ideas have been investigated to find a legal retiming for nonnegative uniform task systems. Notice first that a retiming can always be found from any strict periodic schedule s fulfilling the uniform constraints.

Let s be a strict periodic schedule with period λ. For any job J_i, s_i can be uniquely decomposed with respect to the period: $s_i = t_i + \lambda.q_i$, with $0 \leq t_i < \lambda$ and q_i is an integer. $(t_i)_{\{J_i \in \mathscr{J}\}}$ is called the **core of the periodic schedule**, and $(q_i)_{\{J_i \in \mathscr{J}\}}$ is the **shift of the periodic schedule**.

The shift $(q_i)_{\{J_i \in \mathscr{J}\}}$ is a feasible retiming. This property was used by Gasperoni and Schwiegelsohn [21] by finding the shift of an optimal periodic schedule assuming unlimited resources. Figure 1.11 shows the graph G^r considering the retiming associated with the shift of the optimal schedule depicted in figure 1.8.

In [13], where using retiming for loop shifting is formalized, the authors consider two optimizations, with polynomial graph algorithms:

- the length of the longest path in G^r minimization
- the number of arcs in G^r minimization, so as to reduce the number of precedence constraints for loop compaction.

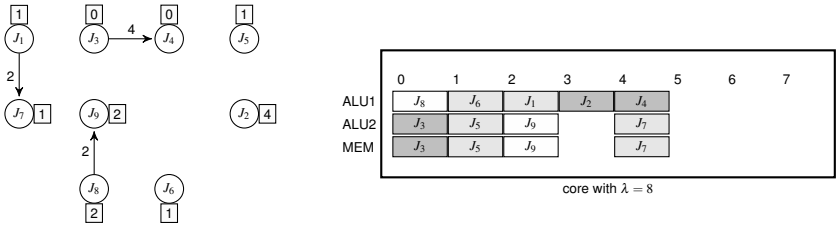


Fig. 1.11: Retiming graph G^r, with r shown above the nodes and periodic schedule with resource constraints

The idea behind DSP approach is to choose a particular retiming r, and then use an algorithm to get a schedule $(t_i)_{\{J_i \in \mathscr{J}\}}$ of G^r, fulfilling the resource constraints to get a periodic schedule of the original problem.

This relies on the following result:

Theorem 1.6. *If r is a feasible retiming, and $(t_i)_{\{J_i \in \mathscr{J}\}}$ is a schedule fulfilling the non cyclic precedence constraints of G^r and the resource constraints, then there exists a periodic schedule s whose core is $(t_i)_{\{J_i \in \mathscr{J}\}}$ and whose shift is $(r_i)_{\{J_i \in \mathscr{J}\}}$.*

Figure 1.11 shows a construction of a core for our example, assuming that arithmetic operations are performed on one of the two available ALU's, while memory jobs (load and store) use one ALU and one memory controller at the same time. The makespan of the schedule $(t_i)_{\{J_i \in \mathscr{J}\}}$, combined with the observation of precedence constraints crossing the core lead to the computation of a period λ in polynomial time [3]. For our example $\lambda = 8$.

From this an interesting special case can be noted: if G has no circuit (except the ones due to the non-reentrance hypothesis for jobs), then it is always possible

to get a feasible retiming r so that G^r has no arcs. Thus at the compaction step, only independent jobs have to be considered. Hence if the underlying non cyclic scheduling problem is easily solvable for independent tasks then the DSP approach provides an optimal periodic schedule. This occurs for example in cyclic shop-like problems (open shop, job-shop) if, unlike in [37], no limitation on the completion time of an iteration, or on the interleaving between iterations is given.

List scheduling algorithms are the most used heuristics for scheduling with precedence and resource constraints. Efficiency of these algorithms in practice is well known. Moreover usually a worst case performance guarantee can be determined in most resource context, from the parallel processors to RCPSP settings where a job may require several units of different resources during its execution.

Using such algorithms at the compaction step leads to a worst case ratio on the periodic schedule. This has been considered for parallel processors [21, 17] and extended to RCPSP in[6].

1.4.3 Energy saving or other resource dependent constraints

In this section we consider problems issued from sensor networks, and in particular the scheduling problems induced by the IEEE 802.15.4/ZigBee network. Here the jobs represents data communications. Now, in real networks, while dealing with periodic schedules, the period is quite long with respect to the processing times of jobs. Moreover, the resources involved in the communication must be awaken to perform the jobs during each period. To avoid energy loss due to many in and out of the resources, it can be interesting to group jobs using the same resource as much as possible, so that the resource is awaken during a short interval once per period. In the context of periodic schedules, this will induce constraints on the core of the schedule, regardless the occurrence number of the concerned jobs.

Let us now present a model of data-flows inspired by the ZigBee norm, introducing grouping constraints. This work is issued from [23, 1]. We consider a tree T, whose nodes represent the clusters and whose edges represent the logical links between them. We then consider a collection of flows. Each flow f is defined by a copy of a subtree of T, oriented as an in-tree, and represents the communication of data along the communication links of T from source nodes of the flow to the unique sink. For flow f, if node i belongs to the sub-tree of f, then we denote, by J_f^i, the communication task associated to node i in flow f. Figure 1.12 shows an example of a tree consisting of seven clusters and two flows.

An iteration of each flow will start at each period. Moreover, the energy constraints of the ZigBee standard consider that each cluster should be active once in each period. So tasks J_f^i for all flows f passing through node i belong to group i, which should be grouped in the period.

Of course, if we do not limit the time of delivery for each flow, then the periodic scheduling problem can be handled in polynomial time by considering a retiming that lead to independent jobs. However, if we wish to achieve a good response time,

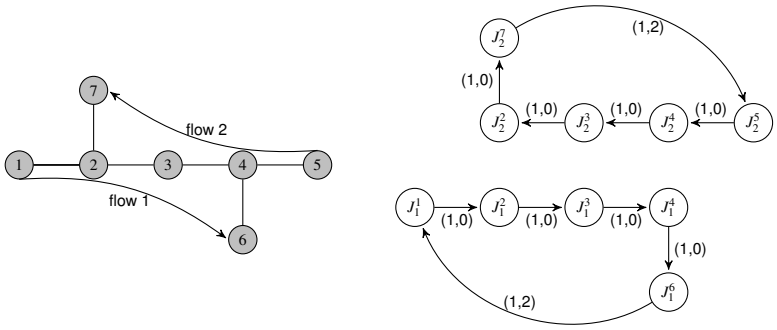


Fig. 1.12: An example of the tree T and two flows, and the associated uniform graph

we need to fix some time limits. We assume here that for each flow f the number of periods crossed by f from a source until its delivery should be less than a given integer p_f. The experiments with a scheduling tool [1], which enables system designers to configure all the required parameters of the IEEE 802.15.4/ZigBee cluster-tree WSNs, illustrate the efficiency of the model.

Moreover, this model induces for each flow a representation of the constraints induced by the data flow by a uniform graph:

1. The nodes of G are for each flow the tasks J_f^i.
2. If there is a communication link $J_f^i \to J_f^j$ in the underlying sub-tree, then (J_f^i, J_f^j) is an arc of G with h value 0.
3. If J_f^i is the sink of the flow f and J_f^j is a source, there is an arc from J_f^i to J_f^j with $h(J_f^i, J_f^j) = p_f + 1$.

Figure 1.12 depicts the graph associated to the two flows, considering $p_1 = p_2 = 1$.

Consider now a uniform task system G, and assume that each job has a group label $k_i \in \{1, \ldots, K\}$. A periodic schedule is said to be grouped if the tasks of the same group are executed close to each other in the core. This notion can be expressed by different means, but we can choose the simplest way here, where each group is to be scheduled as a single super-task in the core schedule. In the context we consider here, the period is usually large with respect to the processing times so we can consider that the complexity induced by the schedule of the jobs inside a super-task is not worth. As we consider here feasibility questions, we assume in the following that the super-task has a unit processing time, but the same results can be obtained by considering sum or max of the processing times of the grouped jobs.

Though the ZibBee feasibility question turns out in the following question: Given a uniform precedence graph G and group labels of the tasks, does a grouped periodic schedule of G exist? We call the UGF (Uniform Grouped Feasibility), this decision problem.

One can easily see that for some instances no grouped periodic schedule exists. If we consider our example assuming $p_1 = p_2 = 0$ this means that the first execution of all jobs have to be scheduled during the first period. As $<J_1^2, 1>$ precedes $<J_1^3, 1>$ for the execution of the first flow, and $<J_2^3, 1>$ precedes $<J_2^2, 1>$ for the second flow, and as J_1^2, J_2^2 (resp. J_1^3, J_2^3) belong to the group of node 2 of the tree (resp. node 3), we get a contradiction.Figure 1.13 shows a core of a grouped schedule for the example of figure 1.12. We prove in [23] that the general UGF problem is

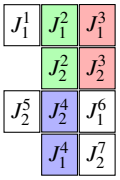


Fig. 1.13: A core of a grouped periodic schedule - grouped jobs are shown by colors

NP-Complete, but the specificity of the tree underlying communication path for the ZigBee problem lead to a polynomial algorithm, based on the use of decomposed software pipelining, which proves its efficiency in practice in [1].

In [26] the authors explore a weaker way of considering grouping in a uniform task system by introducing precedence constraints with arbitrary latencies on the core schedule, called **core constraints**. Unfortunately, they prove that even if no additional resource constraints is assumed, and if unit processing times are considered, deciding the existence of a periodic schedule is also NP-complete.

1.5 Synchronous data flows

This section aims to present several important theoretical results on normalized SDF. The feasibility and the evaluation of the minimum normalized average cycle time are two challenging problems for which the complexity is unknown. Subsection 1.5.1 presents some algorithms to answer these two questions. Subsection 1.5.2 is dedicated to characterization of a periodic feasible schedule of minimum period, leading to a polynomial time algorithm to compute it. This characterization is considered to optimize the total buffers capacity under a minimum period constraint in subsection 1.5.3. Lastly, subsection 1.5.4 is dedicated to the computation of the latency for a real-time application which communications between tasks are modelled using a SDF.

1.5.1 Feasibility and evaluation of the minimum normalized average cycle time

Let us suppose that $G = (\mathscr{J}, A, u, v, M_0)$ is a normalized SDF. G is feasible (or live) if there exists an infinite feasible schedule. Following Theorem 1.1, the simplest way to test the feasibility is to execute the jobs as soon as possible until each job J_i is executed at least N_i times. If it is possible, G is live.

The main drawback of this method is that values N_i are not polynomial and may be quite huge for real-life systems. From a theoretical point of view, the complexity of checking the feasibility of a SDF remains unknown. However, a simple sufficient condition of feasibility was proved by Marchetti ans Munier [33, 32].

Theorem 1.7 (Sufficient condition of feasibilty of a SDF). *Let G be a normalized SDF. If, for any circuit $c \in \mathscr{C}(G)$, the inequality*

$$\sum_{a \in c} M_0(a) > \sum_{a=(J_i,J_j) \in c} \big(Z_j - \gcd(Z_i, Z_j)\big)$$

is true, then G is feasible.

Checking this condition requires to label each arc of $a = (J_i, J_j)$ of the SDF by $V(a) = Z_j - \gcd(Z_i, Z_j) - M_0(a)$ and testing that the sum of the labels of each circuit remains strictly negative. As example, Figure 1.14 pictures the SDF from Figure 1.1 with arcs $a = (J_i, J_j)$ valued by $V(a)$. This graph has nos positive or null valued circuits, thus G is feasible.

Checking the existence of positive circuits can be done using a two steps polynomial time algorithm: the first step consists on checking the non existence of positive or null valued circuits using Bellmann-Ford algorithm [16]. A depth-first search algorithm applied only to critical arcs allows to check the non existence of null-valued circuits.

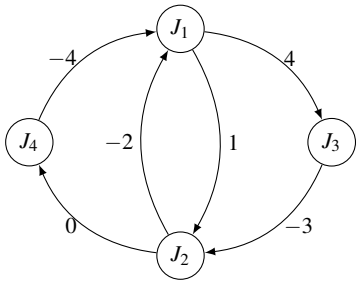


Fig. 1.14: SDF G from Figure 1.1 with arcs $a = (J_i, J_j)$ valued by $V(a) = Z_j - \gcd(Z_i, Z_j) - M_0(a)$

Munier [34] proved that the earliest schedule of a SDF is K-periodic. Thus, the simplest way to evaluate the minimum normalized average cycle time is to compute the earliest schedule until the convergence of the normalized average cycle time. Another way is to compute the expansion of the graph, and determine its average cycle time. The main drawback of these two methods is that they are not polynomial, and thus not efficient whenever $\Sigma_{i=1}^{n} N_i$ is important.

Bodin et al. [11] have proved that, for any integer vector of n components $X \geq 1^n$ an expansion G_X (which is a uniform precedence graph) of G can be defined. For any arc $a = (J_i, J_j)$, arcs of G_X between the duplicates of J_i and J_j models a superset of precedence constraints between J_i and J_j. They also show that dominant values for the computation of the minimum normalized average cycle can be achieved for the vector set $\{X \in \mathbb{N}^n : \forall i \in \{1, \cdots, n\} \quad (X_i \text{ divides } \mathrm{N_i})\}$. These expansions can be used to get upper-bounds of the minimum normalized average cycle.

Algorithm 1 was also developed by Bodin et al. [12] to compute the minimum normalized average cycle time by expanding only jobs of the successive critical circuits. Although non polynomial, this algorithm allows to evaluate quickly this value for industrial instances of large size.

Algorithm 1: Computation of the minimum normalized average cycle time

Require: A normalized SDF $G = (\mathscr{J}, A, u, v, M_0)$.
Ensure : Normalized minimum average cycle time $\lambda(G)$ of G.

Set $M = (Z_1, \cdots, Z_n)$, $\forall i \in \{1, \cdots, n\} \quad (N_i = \frac{M}{Z_i})$;
Set $X = 1^n$, G_X the corresponding expanded graph and c a critical circuit of G_X;
while *every job J_i of c is not expanded N_i times* **do**
    Set $X_i = N_i$ for every job J_i of c;
    Update G_X and a critical circuit c of G_X;
Let $\lambda(c)$ be the average cycle time of G_X; $\lambda(G) = \frac{\lambda(c)}{X_1}$;

1.5.2 Existence and Computation of a periodic schedule of minimum average cycle time

We show in this section that the determination of a feasible periodic schedule of minimum period is a polynomial problem for a normalized SDF. A schedule s is **periodic** if for any job J_i, there exists $w_i \in \mathbb{Q}^{\star +}$ with $\forall n > 1 \quad \big(s(J_i, n) = s(J_1, 1) + (n-1)w_i\big)$. Benabid et al. [7] have proved Theorem 1.8 that characterizes periodic schedules.

Theorem 1.8. *Let G be a normalized strongly connected SDF. For any periodic schedule s, there exists a rational $\lambda^s \in \mathbb{Q}^{\star +}$ such that for any job J_i, $\frac{w_i}{Z_i} = \lambda^s$. Moreover, the precedence relations associated with any place $a = (J_i, J_j)$ are fulfilled by*

s iff

$$s(J_j,1) - s(J_i,1) \geq p_i + \lambda^s (Z_j - M_0(a) - \gcd(Z_i,Z_j)).$$

λ^s is then the average cycle time of s.

Since length $p_i > 0$ for any job J_i, there exists a periodic schedule for G iff for any circuit $c \in \mathscr{C}(G)$, the inequality $\sum_{a=(J_i,J_j)\in c}(Z_j - M_0(a) - \gcd(Z_i,Z_j)) < 0$ holds, which is exactly the condition of feasibility of Theorem1.7. If this condition is true, the minimum average cycle time λ^s can then be computed by finding critical circuits of the graph G_1 with the same structure of G and for which each arc $a = (J_i,J_j)$ is bi-valued by $(p_i, M_0(a) + \gcd(Z_i,Z_j) - Z_j)$.

Consider for example the bi-valued graph G_1 pictured by Figure 1.15 and associated with the graph G of Figure 1.1. The critical circuit of G_1 is $c = (J_1,J_3,J_2,J_1)$ with ratio $\lambda(c) = \frac{1+2+2}{-4+3+2} = 5$. Thus the minimum normalized average period of a periodic schedule is $\lambda^s = 5$.

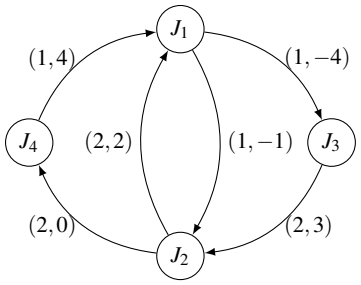


Fig. 1.15: SDF G from Figure 1.1 with arcs $a = (J_i,J_j)$ bi-valued by $(p_i, M_0(a) + \gcd(Z_i,Z_j) - Z_j)$

1.5.3 Optimization of the total buffers capacity under a minimum period constraint

SDF can be considered to model data exchanges [29] for streaming applications. Jobs correspond here to programs that are repeatedly executed. Arcs are associated to buffers. The total amount of memory needed to execute an application is an important criteria for the designers due to the cost of the memories. Thus, the minimization of the total buffers capacity under a minimum period constraint is an important bi-criteria optimization problem.

The capacity $F(a)$ of an arc a is the maximum number of tokens that can be stored simultaneously in the buffer corresponding to a. First at all, Marchetti and Munier proved in [31] that the capacity of a buffer may be modelled using a back-

ward arc as follows by studying the precedence relations induced by this capacity constraints.

Theorem 1.9. *Any arc $a = (J_i, J_j)$ initially marked by $M_0(a)$ with a capacity limited by $F(a) \geq M_0(a)$ may be replaced by a couple of arcs (with no limited capacity) $a_1 = (J_i, J_j)$ and $a_2 = (J_j, J_i)$ with $M_0(a_1) = M_0(a)$ and $M_0(a_2) = F(a) - M_0(a)$ (see Figure 1.16).*

$a = (J_i, J_i)$ with bounded capacity $F(a)$
$M_0(a) \leq F(a)$

$a_1 = (J_i, J_i)$ and $a_2 = (J_j, J_i)$
$M_0(a_1) = M_0(a)$ and $M_0(a_2) = F(a) - M_0(a)$

Fig. 1.16: Transformation of an arc e with a capacity bounded by $F(a)$ into a coupe of arcs with no capacity constraint

Let $G_s = (\mathscr{J}, A_s, u, v, M_0)$ be the SDF associated to G for which each arc a with a limited capacity is replaced by a couple of arcs (a_1, a_2). For any arc a, we denote by $\theta(a) > 0$ the size needed to store one unique data in a. The size of a is then $F(a) \times \theta(a)$ and the total size is thus

$$\mathscr{F} = \sum_{a \in A} \theta(a) F(a) = \sum_{a \in A_s} \theta(a) M_0(a)$$

The optimization problem addressed here is to find an initial marking $M_0(a), a \in A_s$ such that the total size of the memories $\sum_{a \in A_s} \theta(a) M_0(a)$ is minimum, while there exists a schedule with a normalized average cycle time at most equal to K.

Since there is no polynomial algorithm to compute the feasibility and the minimum average cycle time of a SDF, we do not know if this problem belong to $\mathscr{NP}$. Several authors limit their study to periodic schedules in order to get around this problem. In this case, the optimization problem can be expressed easily using the following Integer Linear Program $\Pi(K)$:

minimize
$\left(\sum_{a \in A_s} \theta(a) M_0(a)\right)$
subject to

$$\begin{cases} \forall a = (J_i, J_j) \in A_s & \left(s(J_j, 1) - s(J_i, 1) \geq p_i - K(M_0(a) + \gcd(Z_i, Z_j) - Z_j)\right) \\ \forall a = (J_i, J_j) \in A_s & \left(M_0(a) = k_{i,j} \cdot \gcd(Z_i, Z_j)\right) \\ \forall a = (t_i, t_j) \in A_s & \left(k_{i,j} \in \mathbb{N}\right) \\ \forall J_i \in \mathscr{J} & \left(s(J_i, 1) \geq 0\right) \end{cases}$$

If the initial marking is not fixed, Marchetti and Munier proved in [31] that this problem is $\mathscr{NP}$-complete even if G is a uniform precedence graph with $F(a) = 1$ for every arc. Benazouz et al. [8] developed a 2-approximation ratio algorithm for the general case. The idea of this algorithm is first to solve the associated relaxed linear program. An approximated solution is then built using a classical rounding technique.

1.5.4 Evaluation of the latency for real-time systems

Consider a normalized SDF G without circuits issued from a set of real-time Jobs which are communicating as described in section 1.3.2. The problem is to evaluate efficiently the latency of the system. Latency is a measure of the response time of the system, it is thus fundamental for real-time systems.

Let us define the maximum (*Resp.* minimum) latency $\mathscr{L}_{max}$ (*Resp.* $\mathscr{L}_{min}$) between two connected jobs J_i and J_j as the maximum (*Resp.* minimum) duration between the end of an execution of $<J_i,n_i>$ and the start of an execution of $<J_j,n_J>$ such that there is a precedence relation from $<J_i,n_i>$ to $<J_j,n_J>$. Theorem 1.10 proved by Khatib et al. [28], expresses the minimum and the maximum latency between two periodic communicating jobs:

Theorem 1.10. *The maximum and the minimum latencies between a couple of periodic jobs (J_i,J_j) such that J_i communicates data to J_j are*

$$\mathscr{L}_{min}(J_i,J_j) = r_j - r_i + \alpha - C_i$$

and

$$\mathscr{L}_{max}(J_i,J_j) = r_j - r_i - \max\{0, T_i - T_j\} + \alpha - T_a^\star + T_i - C_i$$

with $T_a^\star = \gcd(T_i,T_j)$ *and* $\alpha = \left\lceil \frac{r_i - r_j + C_i}{T_a^\star} \right\rceil \times T_a^\star$.

Consider for example the two communicating jobs J_1 and J_2 from Figure 1.9. We get $\alpha = 10$, $T_a^\star = 10$, $\mathscr{L}_{min}(J_1,J_2) = 0$ and $\mathscr{L}_{max}(J_1,J_2) = 10$. The delay between the end of $<J_1,1>$ and the beginning of $<J_2,1>$ equals 0 and corresponds to $\mathscr{L}_{min}(J_1,J_2)$. The delay between the end of $<J_1,2>$ and the beginning of $<J_2,1>$ equals 10 and corresponds to $\mathscr{L}_{max}(J_1,J_2)$.

The latency of the entire system is a time gap from a data input of a system to a connected outcome. The worst-case latency is then the longest time gap between the input Job executions and output Job executions of a system.

The exact evaluation of the latency can be obtained by computing an expanded valued graph [28]. Each vertex J_i is duplicated N_i times. Arcs correspond to relations between executions and are valued by the exact latency between the corresponding executions following Theorem 1.10. This method is clearly not polynomial and is not efficient for large repetition vectors.

Now, let $G_{max} = (\mathscr{J} \cup \{s,d\}, A_m, w)$ be the graph built from the SDF G as follows.

- Let $a = (J_i, J_j)$ be an arc of G with $T_a^\star = \gcd(T_i, T_j)$. An arc $e = (J_i, J_j)$ is built for G_{max} with

$$w(e) = \begin{cases} \left\lceil \frac{C_i}{T_a^\star} \right\rceil T_a^\star + T_i - T_a^\star & \text{if } T_i \leq T_j \\ \left\lceil \frac{C_i}{T_a^\star} \right\rceil T_a^\star + \left\lceil \frac{T_i}{T_j} \right\rceil T_j - T_a^\star & \text{otherwise} \end{cases}$$

- For any job J_i without predecessor, add the arc $e = (s, J_i)$ with $w(e) = 0$; for any job J_i without successor, add the arc $e = (J_i, p)$ with $w(e) = D_i$.

Khatib et al. [28] proved that an upper bound of the maximum latency can be computed by evaluating the longest paths of G_{max}. On the same way, a lower bound of the maximum latency can be computed by evaluating the longest paths of $G_{min} = (\mathscr{J} \cup \{s, d\}, A_m, w')$ defined as follows :

- For any arc $e \in A_m$ corresponding to an arc $a = (J_i, J_j) \in A$, $w'(e) = \left\lceil \frac{C_i}{T_a^\star} \right\rceil T_a^\star$.
- For any job J_i without predecessor, add the arc $e = (s, J_i)$ with $w'(e) = 0$; For any job J_i without successor, add the arc $e = (J_i, p)$ with $w'(e) = D_i$

They also experimentally show that the gap between the exact value and the upper (resp. lower) bound varies between 10 and 15% (resp. 20 and 30%).

Let us consider the real-time system of section 1.3.2 adding a communication from J_1 to J_3. Corresponding graphs G_{min} and G_{max} are pictured by Figure 1.17. The value of longest paths of these two graphs are respectively equal to 90 and 60.

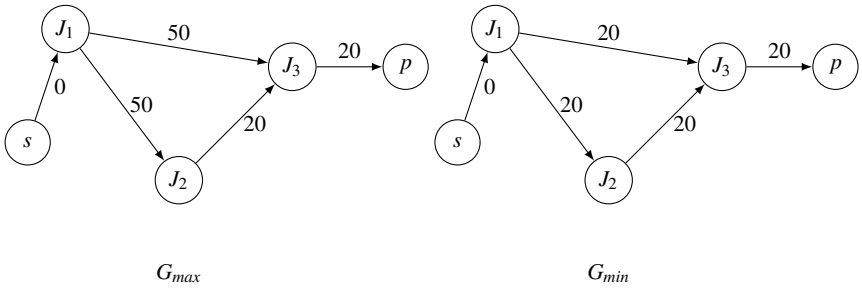


Fig. 1.17: G_{min} and G_{max} for the real-time system of section 1.3.2

1.6 Conclusion

Cyclic scheduling problems arise in many crucial applications of computing and real time systems. Data transfers and data dependences induce specific precedence constraints, that have been analyzed and sometimes combined with resource constraints. In this chapter we proposed an insight on the main theoretical tools and

algorithms of the field, and gave the flavor of the most recent questions and results. Several questions remain open, from complexity to algorithmic issues. We hope that the readers will further contribute to their solutions.

References

1. Ahmad, A., Hanzálek, Z.: An energy efficient schedule for IEEE 802.15.4/zigbee cluster tree WSN with multiple collision domains and period crossing constraint. IEEE Transactions on Industrial Informatics **14**(1), 12–23 (2018)
2. Alacaide, D., Chu, C., Kats, V., Levner, E., Sierksma, G.: Cyclic multiple robot scheduling with time-window constraints using a critical path approach. European Journal of Operational Resarch **177**, 147–162 (2007)
3. Ayala, M., Benabid, A., Artigues, C., Hanen, C.: The resource-constrained modulo scheduling problem: an experimental study. Computational Optimization and Applications **54**(3), 645–673 (2013)
4. Barroso, L.: The price of performance. ACM Queue **3**(7), 48–53 (2005)
5. Bekooij, M.J., Jansen, P.G., Smit, G.J., Wiggers, M.H.: Efficient computation of buffer capacities for cyclo-static real-time systems with back-pressure. In: 2007 44th ACM/IEEE Design Automation Conference, San Diego, CA, 4–8 June 2007, pp. 281–292. IEEE (2007)
6. Benabid, A., Hanen, C.: Worst case analysis of decomposed software pipelining for cyclic unitary RCPSP with precedence delays. Journal of Scheduling **14**(5), 511–522 (2011)
7. Benabid-Najjar, A., Hanen, C., Marchetti, O., Kordon, A.M.: Periodic schedules for bounded timed weighted event graphs. IEEE Transactions on Automatic Control **57**(5), 1222–1232 (2012)
8. Benazouz, M., Marchetti, O., Kordon, A.M., Urard, P.: A new approach for minimizing buffer capacities with throughput constraint for embedded system design. In: The 8th ACS/IEEE International Conference on Computer Systems and Applications, AICCSA 2010, Hammamet, Tunisia, 16–19 May 2010, pp. 1–8. IEEE (2010)
9. Benazouz, M., Marchetti, O., Munier-Kordon, A., Michel, T.: A new method for minimizing buffer sizes for cyclo-static dataflow graphs. In: 2010 8th IEEE Workshop on Embedded Systems for Real-Time Multimedia, Scottsdale, AZ, 2010, pp. 11–20. IEEE (2010)
10. Blachot, F., de Dinechin, B.D., Huard, G.: SCAN: A Heuristic for Near-Optimal Software Pipelining. In: Nagel W.E., Walter W.V., Lehner W. (eds) Euro-Par 2006 Parallel Processing. Euro-Par 2006. Lecture Notes in Computer Science, vol 4128. Springer, Berlin, Heidelberg (2006)
11. Bodin, B., Kordon, A.M., de Dinechin, B.D.: K-periodic schedules for evaluating the maximum throughput of a synchronous dataflow graph. In: 2012 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS XII, Samos, Greece, 16–19 July 2012, pp. 152–159. IEEE (2012)
12. Bodin, B., Kordon, A.M., de Dinechin, B.D.: Optimal and fast throughput evaluation of CSDF. In: DAC '16 Proceedings of the 53rd Annual Design Automation Conference, Austin, Texas, 5–9 June 2016, pp. 160:1–160:6. ACM New York, NY, USA (2016)
13. Calland, P.Y., Darte, A., Robert, Y.: Circuit retiming applied to decomposed software pipelining. IEEE Transactions on Parallel and Distributed Systems **9**(1), 24–35 (1998)
14. Carlier, J., Chrétienne, P.: Problèmes d'ordonnancement : modèlisation, complexité, algorithmes. Masson, Paris (1988)
15. Claire, H.: Cyclic scheduling. In: Robert, Y., Vivien, F. (eds.) Introduction to Scheduling. Springer (2009)
16. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. MIT Press (1990)
17. Darte, A., Huard, G.: Loop shifting for loop compaction. International Journal of Parallel Programming **28**(499), 415–431 (2000).

18. Dasdan, A., Irani, S., Gupta, R.K.: Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. In: DAC '99 Proceedings of the 36th annual ACM/IEEE Design Automation Conference, New Orleans, Louisiana, USA, 21–25 June 1999, pp. 37–42. ACM New York, NY, USA (1999)
19. Dupont De Dinechin, B., Artigues, C., Azem, S.: Resource-constrained moulo scheduling. In: Artigues, C., Demassey, S., Néron, E. (eds.) Resource Constrained Project Scheduling: Models, Algorithms, Extensions and Applications, Control Systems, Robotics and Manufacturing Series, pp. 267–277. ISTE-WILEY (2008)
20. Eichenberger, A., Davidson, E.: Efficient formulation for optimal modulo schedulers. In: ACM SIGPLAN conference on programming language design and implementation, pp. 194–205. Las Vegas, Nevada (1997)
21. Gasperoni, F., Schwiegelshohn, U.: Generating close to optimum loop schedules on parallel processors. Parallel Processing Letters **4**, 391–403 (1994)
22. Goubier, T., Sirdey, R., Louise, S., David, V.: Σc: A programming model and language for embedded manycores. In: Xiang, Y., Cuzzocrea, A., Hobbs, M., Zhou, W. (eds) Algorithms and Architectures for Parallel Processing — 11th International Conference, ICA3PP, Melbourne, Australia, 24–26 October 2011, pp. 385–394. Springer (2011)
23. Hanen, C., Hanzalek, Z.: Grouping tasks to save energy in a cyclic scheduling problem: a complexity study. HAL CCSD (2012)
24. Hanen, C., Munier, A.: Cyclic scheduling on parallel processors: An overview. In: Chrétienne, P., Coffman, E.G., Lenstra, J.K., Liu, Z. (eds.) Scheduling theory and its applications. J. Wiley and sons (1994)
25. Hanen, C., Munier, A.: A study of the cyclic scheduling problem on parallel processors. Discrete Applied Mathematics **57**(2-3), 167–192 (1995)
26. Hanzálek, Z., Hanen, C.: The impact of core precedences in a cyclic RCPSP with precedence delays. Journal of Scheduling **18**(3), 275–284 (2015)
27. Kats, V., Levner, E.: Cyclic routing algorithms in graphs: Performance analysis and applications to robot scheduling. Computers & Industrial Engineering **61**(2), 279–288 (2011)
28. Khatib, J., Kordon, A.M., Klikpo, E.C., Trabelsi-Colibet, K.: Computing latency of a real-time system modeled by synchronous dataflow graph. In: Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, 19-21 October 2016, pp. 87–96. ACM New York, NY, USA (2016)
29. Lee, E.A., Messerschmitt, D.G.: Synchronous data flow. Proceedings of the IEEE **75**(9), 1235–1245 (1987)
30. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM **20**(1), 46–61 (1973)
31. Marchetti, O., Kordon, A.M.: Complexity results for weighted timed event graphs. Discrete Optimization **7**(3), 166–180 (2010)
32. Marchetti, O., Munier-Kordon, A.: Cyclic scheduling for the synthesis of embedded systems. In: Robert, Y., Vivien, F. (eds.) Introduction to Scheduling. Springer (2009)
33. Marchetti, O., Munier-Kordon, A.: A sufficient condition for the liveness of weighted event graphs. European Journal of Operational Research **197**(2), 532–540 (2009).
34. Munier, A.: Régime asymptotique optimal d'un graphe d'événements temporisé généralisé: application à un problème d'assemblage. RAIRO-Automatique Productique Informatique Industrielle **27**(5), 487–513 (1993)
35. Munier-Kordon, A.: A graph-based analysis of the cyclic scheduling problem with time constraints: schedulability and periodicity of the earliest schedule. Journal of Scheduling **14**(1), 103–117 (2011).
36. Oh, H., Ha, S.: Efficient code synthesis from extended dataflow graphs for multimedia applications. In: Proceedings of the 39th Design Automation Conference, DAC 2002, New Orleans, LA, USA, 10–14 June 2002, pp. 275–280. ACM New York, NY, USA (2002)
37. Pempera, J., Smutnicki, C.: Open shop cyclic scheduling. European Journal of Operational Research **269**(2), 773–781 (2018)

38. Rau, B.R.: Iterative modulo scheduling: an algorithm for software pipelining loops. In: MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture, 30 November–2 December 1994, pp. 63–74. ACM, New York, NY, USA (1994)
39. Robert, Y., Vivien, F.: Introduction to Scheduling. Chapman and Hall/CRC Press (2009)
40. Thies, W., Karczmarek, M., Gordon, M., Maze, D.Z., Wong, J., Hoffman, H., Brown, M., Amarasinghe, S.: StreamIt: A compiler for streaming applications. Tech. rep., Massachussetts Institute of Technology (2001)