Resource-constrained Scheduling of Partitioned Algorithms on Processor Array.

Michèle Dion, Tanguy Risset, Yves Robert

▶ **To cite this version:**

Michèle Dion, Tanguy Risset, Yves Robert. Resource-constrained Scheduling of Partitioned Algorithms on Processor Array.. [Research Report] LIP RR-1994-19, Laboratoire de l'informatique du parallélisme. 1994, 2+29p. hal-02101776

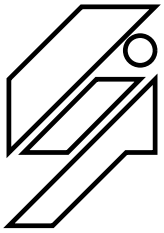
HAL Id: hal-02101776

https://hal-lara.archives-ouvertes.fr/hal-02101776

Submitted on 17 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Laboratoire de l'Informatique du Parallélisme

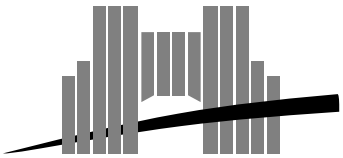
Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

Resource-constrained Scheduling of Partitioned Algorithms on Processor Arrays

Michèle Dion
Tanguy Risset
Yves Robert

June 1994

Research Report N^{O} 94-19



Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) 72.72.80.00 Télécopieur : (+33) 72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

Resource-constrained Scheduling of Partitioned Algorithms on Processor Arrays

Michèle Dion
Tanguy Risset
Yves Robert

June 1994

Abstract

We deal with the problem of partitioning and mapping uniform loop nests onto physical processor arrays. Resource constraints are taken into account: not only we assume a limited number of available processors, but we also assume that the communication capabilities of the physical processors are restricted (in particular, the number of communication links in each direction is bounded). This paper is motivated by the recent work of Chou and Kung and of Thiele. Our main contributions are a new formulation of the complex optimization problem to be solved in terms of a single integer linear programming problem, as well as optimal scheduling algorithms and complexity results in the case of linear processor arrays.

Keywords: parallelism, SPMD, processor arrays, scheduling, tiles, resource constraints

Résumé

Nous étudions dans ce rapport le partitionnement et le placement de nids de boucles uniformes sur des tableaux de processeurs. Les contraintes liées aux ressources sont prises en compte: non seulement nous supposons disposer d'un nombre fini de processeurs, mais également que les capacités de communication des processeurs sont limitées (en particulier, le nombre de liens de communication dans chaque direction est borné). Les récents travaux de Chou et Kung et de Thiele sont à l'origine de ce travail. Nos principales contributions résident dans la reformulation d'un problème complexe d'optimisation en un problème de programmation linéaire, ainsi que dans des algorithmes d'ordonnancement et des résultats de complexité dans le cas de tableaux linéaires de processeurs.

Mots-clés: parallélisme, SPMD, tableaux de processeurs, ordonnancement, tuiles, ressources bornées

Resource-constrained Scheduling of Partitioned Algorithms on Processor Array

Michèle Dion, Tanguy Risset and Yves Robert

Laboratoire LIP-IMAG, CNRS URA 1398
Ecole Normale Supérieure de Lyon, 69364 LYON Cedex 07
e-mail: [mdion,risset,yrobert]@lip.ens-lyon.fr

June 28, 1994

Abstract

We deal with the problem of partitioning and mapping uniform loop nests onto physical processor arrays. Resource constraints are taken into account: not only we assume a limited number of available processors, but we also assume that the communication capabilities of the physical processors are restricted (in particular, the number of communication links in each direction is bounded). This paper is motivated by the recent work of Chou and Kung and of Thiele. Our main contributions are a new formulation of the complex optimization problem to be solved in terms of a single integer linear programming problem, as well as optimal scheduling algorithms and complexity results in the case of linear processor arrays.

1 Introduction

In this paper, we deal with the problem of partitioning and mapping uniform loop nests onto physical processor arrays. Resource constraints are taken into account: not only we assume a limited number of available processors, but we also assume that the communication capabilities of the physical processors are restricted (in particular, the number of communication links in each direction is bounded).

Partitioning (or tiling) uniform loop nest onto processor arrays has motivated a great amount of research in the last ten years [1, 6, 10, 11, 12, 15, 20, 18, 21, 19, 22, 3] (this small list is far from bieng exhaustive). *Tiling* is a widely used technique to increase the granularity of computations and the locality of data references. When targeting a SPMD style of programming [10, 18, 20, 21, 3], tiles are usually considered to be *atomic*: inter-processor communications only take place at the end of the processing of each tile. While well-suited to a data-parallel approach, this hypothesis is unnecessarily restrictive when targeting VLSI processor arrays. Also, most current distributed-memory machines are capable of performing communications in parallel with computations. Therefore, it is of great practical importance to study tiling techniques while assuming that communications and computations can overlap within a tile.

This paper is motivated by the recent work of Chou and Kung [4] and of Thiele [23]. Chou and Kung [4] have made a major contribution in formulating the problem of tiling uniform dependence graphs assuming limited resources and computation/communication overlap. Thiele [23] was the first to introduce Integer Linear Programming (ILP) techniques to optimize the scheduling of partitioned algorithms onto VLSI processor arrays with limited computational resources.

We start from the formulation of the problem by Chou and Kung. We summarize their approach and we elaborate upon it along two important directions:

- We capture the complex optimization problem to be solved in terms of a single Integer Linear Programming (ILP) problem. We point out that Chou and Kung were only able to solve a very restricted instance of the problem that they have formulated.
- We give optimal scheduling algorithms and complexity results in the case of linear processor arrays.

The paper is organized as follows: in Section 2 we state the scheduling problem to be solved. We follow the approach of Chou and Kung and we review the heuristics that they propose. In Section 3 we derive a new formulation of the problem in terms of a single ILP problem. We give several examples and briefly discuss some practical issues to compute the solution. Then in Section 4 we concentrate upon linear processor arrays. In this particular case, we are able to give an analytical expression for the best scheduling and we derive optimal algorithms. Finally in Section 5 we summarize our results and give several perspectives for future work.

2 Problem formulation

The original formulation of the problem is due to Chou and Kung [4]. We introduce it by working out a small example.

2.1 Scheduling constraints

Consider the following perfect uniform loop nest of depth 2:

for $i_1 = 1$ to N_1 do
  for $i_2 = 1$ to N_2 do
    $a(i_1, i_2) = f(a(i_1 - 1, i_2), a(i_1, i_2 - 1), a(i_1 - 1, i_2 + 1))$
  endfor
endfor

where f is an arbitrary function. The iteration space is the rectangle $Iter = \{(i_1, i_2), 1 \leq i_1 \leq N_1, 1 \leq i_2 \leq N_2\}$. The dependences vectors are captured in the dependence matrix $D = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \end{pmatrix}$.

Assume now that this loop nest is to be executed on a 2D-grid of processors of size $M_1 \times M_2$. In most practical cases the size of the iteration space (i.e. $N_1 \times N_2$, the number of computation points) is much larger than the size of the processor grid. It is then needed to partition the iteration space into rectangular tiles and to allocate a

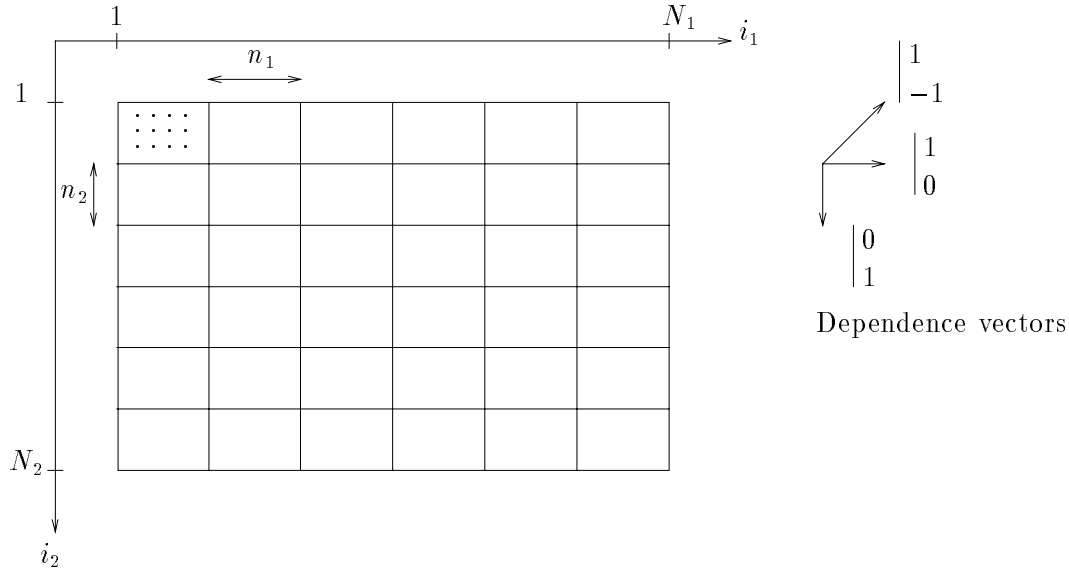


Figure 1: Iteration space for the example ($n_1 = 4$, $n_2 = 3$)

whole tile to a single processor. For the sake of simplicity, Chou and Kung assume the tiles to be rectangular and along the direction indices i_1 and i_2 of the iteration space *Iter* (see Figure 1). Let $n_1 \times n_2$ be the size of a tile, where $N_1 = n_1 \times M_1$ and $N_2 = n_2 \times M_2$. In Figure 1 we have tiles of size 4 by 3.

The problem is thus resource-constrained as one single processor is responsible for processing a whole tile of $n_1 \times n_2$ computation points. But there are many other possible limitations to take into account, in particular the number of communication channels and the fixed topology of the communication network.

Indeed, as stated by Chou and Kung, the physical characteristics of VLSI layout impress a stringent limitation on the design of array processors. Typically, a 2D-grid of processors will be interconnected with horizontal and vertical physical channels (and possibly also diagonal and antidiagonal channels), each of these in limited number. This means that long distance communications will have to be "routed", i.e. decomposed into a sequence of physically possible communications. This also means that several inter-processor communications which are simultaneously ready to be performed will compete for the communication resources.

In our example, assume that the interconnection network is that given in Figure 2. There are four communication channels (or links) going out every processor (or cell): they are labeled $east_1$, $east_2$, *north* and *south* as indicated in the figure.

Each processor is responsible for a whole tile and must execute

- $n_1 \times n_2$ computations
- as many communications as imposed by the dependence vectors.

We see in Figure 3 that there are 13 dependence vectors that require an inter-cell communication (all other dependences are internalized in the cell). These 13 inter-cell communications will lead to 14 physical communications, as the dependence vector

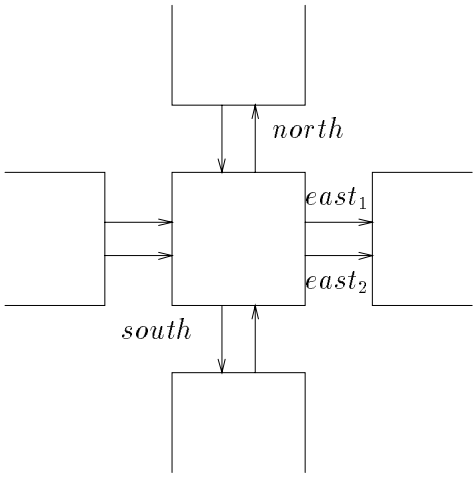


Figure 2: Communication network for the example

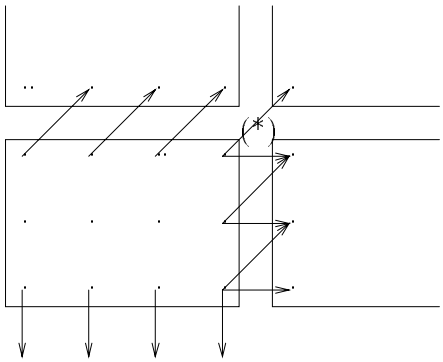


Figure 3: Inter-cell communications

labeled $(*)$ on the Figure cannot be realized by using a single link but instead requires a two-step implementation: a first communication using link *north* then a second one using either link *east* (or the other way round).

In the general case the interconnection network is represented by a tuple (L, Q) where L is a set of column vectors and Q is a row vector. $L = \{l_1, \ldots, l_d\}$, where l_i represents a unique direction of communication links. $Q = (q_1, \ldots, q_d)$ represents the number of communication channels at each direction. In our example we have $L = \{\text{east,south,north}\} = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ -1 \end{pmatrix} \right\}$ and $Q = (2, 1, 1)$. Each inter-cell communication induced by a dependence vector is then decomposed as an ordered sequence $(s_1, \ldots, s_i, \ldots)$ where s_i belongs to L. Note that

- the same dependence vector can lead to different inter-cell communications. This is illustrated in Figure 3 with dependence vector $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ which leads either to the sequence (l_3) (*north*) or to the sequence (l_3, l_1) (*north* then *east*), depending upon the location of the vector source within the tile
- the order in which the links are used to implement an inter-cell communication is assumed to be *a-priori* fixed. Of course this is an arbitrary restriction but it is required to reduce the number of parameters (in practice sequences will be of length 2 at most, we can still try several orderings).

How will we schedule the tiles ? Let τ_{calc} be the time needed for one computation (an instance of the statement $a(i_1, i_2) = f(a(i_1-1, i_2), a(i_1, i_2-1), a(i_1-1, i_2+1))$). Let τ_{comm} be the time for one communication over a physical link. There are several realistic hypotheses that make Chou and Kung's approach very interesting when targeting a VLSI processor grid:

Communication/computation overlap. Tiles are not supposed to be atomic (as opposed to the work presented in [10, 19, 20, 3]). Rather, as soon as a point in the tile has been computed, we can start to communicate its value to neighboring tiles.

Cyclic scheduling. We look for a scheduling in which each tile will execute the same program, up to a translation in time. More precisely, let us assign tile indices (x, y) to each tile, where $0 \leq x < M_1$ and $0 \leq x < M_2$. If tile $(0, 0)$ starts its program P at time-step 0, tile (x, y) will start the same program at time-step $T_1 x + T_2 y$, where T_1 and T_2 are the relative time offsets between processors in each dimension.

Searching for a cyclic scheduling is the key to a regular design. Of course, the goal is to minimize the global execution time, and therefore to choose T_1 and T_2 as small as possible. We formulate the objective function to be minimized more precisely in Section 3. Beforehand, we express all the constraints to be satisfied in an intuitive manner.

Let us consider a fixed tile $\mathcal{T}$ of tile index (x, y). Let (j_1, j_2) be the indices of the computation points inside the tile, where $1 \leq j_1 \leq n_1 = 4$ and $1 \leq j_2 \leq n_2 = 3$. We have the following relationship between global and local indices of the computation points: $i_1 = x \times n_1 + j_1, i_2 = y \times n_2 + j_2$.

Scheduling computations There is a partial ordering between points of the tile: if both points p and q belong to $\mathcal{T}$ and if p depends upon q, i.e. if $p = q + d_i$ for some dependence vector $d_i \in D$, then q must be executed before p. More precisely, let t_p (resp: t_q) be the time-step where execution of point p (resp: q) begins. We have the constraint

$$t_q + \tau_{calc} \leq t_p \tag{1}$$

Scheduling communications Consider first the case of an inter-cell communication c whose ordered sequence is the singleton (l_i). Such a communication can be realized with the use of a single channel of direction l_i. Let $q \in \mathcal{T}$ be the source of the communication and $p \in \mathcal{T}'$ be the sink. Note that the index (x', y') of $\mathcal{T}'$ satisfies to $(x', y') = (x, y) + l_i^t$. In our example, if l_i corresponds to the direction *east*, then $l_i^t = (1, 0)$, $x' = x + 1$ and $y' = y$. This means that point p of $\mathcal{T}'$ is scheduled with a relative offset T_1 to the same point p of $\mathcal{T}$. Let t_c be the time-step at which the communication c begins. The scheduling constraints are

$$\begin{array}{c} t_q + \tau_{calc} \leq t_c \\ t_c + \tau_{comm} \leq t_p + T_1 \end{array} \tag{2}$$

The first constraint expresses the fact that the communication c cannot start before the end of the computation of q. The second constraint says that execution of point p in $\mathcal{T}'$ cannot start before the end of c.

Now consider the case of an inter-cell communication to be decomposed by using two, say, physical links. In the example, point q of local index $(1, 4)$ in $\mathcal{T}$ is the source of a communication to point p of local index $(3, 1)$ in $\mathcal{T}'$ of tile index (x', y'), with $x' = x + 1$ and $y' = y - 1$ (see Figure 3: communication marked with (*)). If the fixed routing order is *north* then *east* as before, we label the two physical communications as c_1 and c_2 (with starting times t_{c_1} and t_{c_2}) and we have the constraints:

$$\begin{array}{c} t_q + \tau_{calc} \leq t_{c_1} \\ t_{c_1} + \tau_{comm} \leq t_{c_2} - T_2 \\ t_{c_2} + \tau_{comm} \leq t_p + T_1 \end{array} \tag{3}$$

Of course, we need to compute the number of physical communications beforehand and to assign a distinct label (number) to each of them.

Resource constraints All resource constraints remains to be expressed. At most one point can be executed within $\mathcal{T}$ at a given time-step. Also, there is only a limited number of links available in each communication direction. In the simple case $\tau_{calc} = \tau_{comm} = 1$, we have to say that at most 1 computation and q_i communications along direction l_i can start at any time-step. In the general case, the condition is more difficult to state. How to express resource constraints is explained in Section 3.

Objective function The goal is to minimize the total execution time, which is equal to

$$(M_1 - 1) \times |T_1| + (M_2 - 1) \times |T_2| + t_{last} + \tau_{calc} \tag{4}$$

Here, we assume that in the first tile, the first point start execution at time 0 and the last one at time t_{last}. The expression $(M_1 - 1) \times |T_1| + (M_2 - 1) \times |T_2|$ represents the

time at which the last tile starts execution, and is a good approximation of the total execution time in practice.

2.2 Chou and Kung's heuristics

There are many variables and many constraints, therefore Chou and Kung propose several heuristics to order computations and communications.

Computation ordering If we assumed unlimited computation resources, we could use Lamport's hyperplane method to schedule the computations points of the dependence graph. We would search for a linear scheduling vector π such that $\pi^t \cdot d_i \geq 1$ for all dependence vector $d_i \in D$. We write this set of constraints as $\pi^t \cdot D \geq 1$. In our example, let $\pi^t = (\pi_1, \pi_2)$, we obtain the conditions

$$\pi_1 \geq 1,\ \pi_2 \geq 1,\ \text{and}\ \pi_1 - \pi_2 \geq 1.$$

A possible choice is $\pi^t = (2, 1)$. With Lamport's hyperplane method, a point $p \in Iter$ is executed at time-step $t_\pi(p) = \pi^t \cdot p$.

Given π, we have the partial ordering induced by t_π to schedule the points of a tile. Using local indices within the tile to identify the points, we have 9 values of t_π for the 12 points of the tile:

$t_\pi(p)$	3	4	5	6	7	8	9	10	11
points	(1,1)	(1,2)	(1,3) (2,1)	(2,2)	(2,3) (3,1)	(3,2)	(3,3) (4,1)	(4,2)	(4,3)

Chou and Kung propose to use t_π as a total ordering for computation events. They break ties arbitrarily (for example using a lexicographical criterion). This heuristic dramatically reduces the search space. Computation points are totally ordered as $p_1, p_2, \ldots, p_{n_1 \times n_2}$. Let t_{p_i} be the i-th computation event, i.e. the time at which the computation of the i-th point p_i begins. All computation constraints reduce to

$$t_{p_i} + \tau_{calc} \leq t_{p_{i+1}}, 1 \leq i < n_1 \times n_2.$$

Indeed, dependences are preserved owing to the choice of π with the condition $\pi^t \cdot D \geq 1$. And resource constraints follow from the total ordering of computations.

Communication ordering Inter-cell communications are decomposed into physical communications. All physical communications are assigned a distinct label. Communications along a given direction l_i are ordered totally, according to the value i of their sink p_i: the smaller i, the more urgent the communication. Ties are broken using source values. Finally, in case there are more than one channel in direction l_i (i.e. $q_i > 2$), links are attributed on in a round-robin fashion (ASAP policy).

Problem solution The above heuristics permit to have a total ordering of the computations, as well as a total ordering of communications along each direction. Let $nb_{calc} = n_1 \times n_2$ be the number of computations and nb_{comm} be the number of communications (after having decomposed inter-cell communications into sequences of physical communications). Therefore, there remains nb_{calc} variables t_{p_i} for the computation

events, and nb_{comm} variables t_{c_i} for the communication events. Theses variables are linked by the scheduling constraints (equations (1),(2) and (3)).

The objective function (equation (4)) has absolute value operators, which is not allowed by the standard format of ILP problems. But the sign of the T_i's can be determined with the help of the scheduling vector π, and the problem is now reduced to a standard ILP problem.

2.3 Discussion

As already said, the optimization problem to be solved has many variables and many constraints, hence heuristics are likely to be crucial for deriving a good solution.

We believe that Chou and Kung's approach can be greatly improved. Indeed, their approach relies upon a linear scheduling vector π that gives a partial ordering for computations. From this partial ordering a total ordering is induced (ties are broken arbitrarily). This solution is nice in that it reduces the search space, but it can lead to poor results as illustrated by the following two simple examples.

Ties breaking Consider a 2d-problem with dependence matrix $D = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and assume $\tau_{calc} = \tau_{comm} = 1$. A possible scheduling vector is $\pi^t = (1,1)$. Consider a 2×2 tiling with unlimited communication resources (2 links in each direction *east* and *south* are enough). The tile $\mathcal{T}$ of tile index $(0,0)$ is the first tile to start computing. We schedule point $(1,1)$ of tile $\mathcal{T}$ at time-step 1 and point $(2,2)$ of $\mathcal{T}$ at time-step 4, because of the partial ordering induced by π. For points $(2,1)$ and $(1,2)$ of $\mathcal{T}$, we have to break ties:

- if we schedule point $(1,2)$ of $\mathcal{T}$ at time-step 2 (and point $(2,1)$ at time-step 3), then $T_1 = 3$, $T_2 = 4$ and the global computation time is $3(M_1 - 1) + 4(M_2 - 1) + 4$
- if we do the other way round, we obtain as global computation time $4(M_1 - 1) + 3(M_2 - 1) + 4$.

If $M_1 \gg M_2$ the second solution is much better and this cannot be found by the method presented in [4].

Linear tiles Consider now a unidimensional problem with dependence matrix $D =$ (3). Tiles are segments of size n (see Figure 4). Assume $\tau_{calc} = \tau_{comm} = 1$ and one link

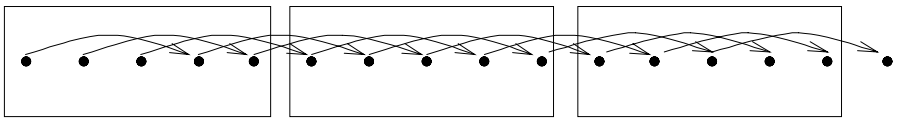


Figure 4: Example of linear tiling, $n = 5$, $l = 3$

in the direction *east* (unlimited communication resources) for the sake of simplicity. The only possible scheduling vectors are positive multiples of $\pi = (1)$, hence Chou and Kung's heuristic will always lead to execute points from the left to the right. The offset

for a tile of size n will then be $T_1 = n-2$, while we can achieve much better: we obtain $T_1 = \lfloor \frac{2n}{3} \rfloor + 2$ in Section 4. To get such a result, we have to use non-linear orderings for scheduling the points of the tile (see Section 4).

3 A solution based upon integer linear programming

We expose the problem formulation for a two dimensional graph, but the extension to any dimension is immediate. The problem tackled here is close to the well-known *single machine scheduling problem* [13, 2]. The differences lie in the cyclicity constraint and in the communication channel management. The major difficulty is to express collision constraints: a processor can execute at most a single computation at each time-step. The same constraint stands for each channel: a channel can perform at most one communication at a time. In the single machine scheduling problem, computation points are numbered from 1 to N^p_{max} (using the lexicographic order on their coordinates for example). An array $COMP[1..N^p_{max}, 1..N^p_{max}]$ of $0-1$ variables is introduced. Computation number i is executed in *position* j if and only if $COMP[i,j] = 1$. Deriving an integer linear problem to solve the single machine scheduling is easy [13]. However, in our problem, communications and computations neither take the same amount of time nor compete for the same resources. Thus we cannot limit ourselves to the search of the *order* of execution, we must compute the actual starting times of each computation and of each communication. We describe in this section a technique to express the problem as an integer linear problem. A similar idea has been introduced by Thiele [23] in a much restricted framework. We first precisely describe all constraints, then we explain how to express these constraints using an integer linear programming formulation.

3.1 Constraints of the cyclic two-dimensional problem

Consider a two-dimensional uniform graph of size $N_1 \times N_2$, each tile is composed of $N^p_{max} = n_1 \times n_2$ computation points (in figure 1 for example, $N^p_{max} = 12$). These N^p_{max} points are numbered from 1 to N^p_{max} (using the lexicographic order on local indices).

We schedule computations and communications for tile $(0,0)$, which starts execution at time 0, and we derive the offsets T_1 and T_2, which respectively correspond to the starting time of tile $(1,0)$ (east) and of tile $(0,1)$ (south). Then if a tile $\mathcal{T}$ has (x,y) for tile index , it starts execution at time $T_1 x + T_2 y$.

The interconnection network is known statically, It is composed of four kinds of links: links going *east*, *west*, *south* or *north*. Each communication induced by a dependence vector going out of tile $(0,0)$ is decomposed into an *ordered list* of inter-cell communications, each of them taking place in one of the four previous directions. The total number of inter-cell communications is N^c_{max} (in figure 3, $N^c_{max} = 14$). We must:

- schedule the computations so that:
  - each computation is executed once and only once
  - two computations taking place on the same processor do not overlap
  - dependences between computations are respected
- schedule (and map onto the links) the communications so that:

- each communication is executed once and only once
- two communications taking place on the same link do not overlap
- dependences between computations and communications are respected

- minimize the total latency, approximated by the expression

$$(M_1 - 1) \times |T_1| + (M_2 - 1) \times |T_2|$$

3.1.1 Linear integer problem formulation

We first make the following assumption: we know *a priori* a constant T_{max} to bound the execution time of a tile[1]. This assumption enables us to consider a "time" vector of size T_{max} whose i-th coordinate is equal to i:

$$TIME = (0, 1, 2, ..., T_{max} - 1)$$

We introduce two $0-1$ variable arrays, $COMP$ for the computations and $COMM$ for the communications.

Computations

The array $COMP$ is used to indicate the starting time of each computation:

$$COMP[1..N^p_{max}, 0..T_{max} - 1]$$

with the rule: $COMP[i, j] = 1$ if and only if the computation of the point i in tile $(0, 0)$ *begins at step* j, and $COMP[i, j] = 0$ otherwise. Let τ_{calc} be the time needed for one computation. The non-collision constraint between computations is expressed as:

$$\forall j,\ 0 \leq j \leq T_{max} - \tau_{calc}, \quad \sum_{i=1}^{N^p_{max}} \sum_{k=j}^{j+\tau_{calc}-1} COMP[i, k] \leq 1$$

which says in effect that the sub-matrix $COMP[1..N^p_{max}, j..j + \tau_{calc} - 1]$ must contain at most one "1". Thus, in any time interval of τ_{calc} steps, at most one computation starts.

The other constraints on the computations are expressed as follows:

- each computation starts once and only once:

$$\begin{array}{lll} \forall i, j & 0 \leq COMP[i, j] & \leq 1 \\ \forall i & \sum_{j=0}^{T_{max}-1} COMP[i, j] & = 1 \end{array}$$

[1] This assumption can be easily eliminated: guess a value for T_{max}; if it leads to an empty solution set for the linear problem, increase the value; iterate until a solution is found. More efficient methods to determine a good constant T_{max} are given in [23].

- If the point i_1 depends upon the point i_2 then:

$$T^p_{beg}[i_2] + \tau_{calc} \leq T^p_{beg}[i_1]$$

where $T^p_{beg}[i]$ is the starting time of point i, expressed by:

$$T^p_{beg}[i] = \sum_{j=0}^{T_{max}-1} COMP[i,j] * TIME[j] = COMP[i,\cdot]^t \cdot TIME$$

Communications

Expressing the communication constraints is more complicated. However, we use similar ideas. Because of the cyclicity of the scheduling, we only have to schedule inter-cell communications going out of a tile (in-going communications are going out of neighbor cells!). We introduce the 3-dimensional array of $0-1$ variables:

$$COMM[1..N^c_{max}, 1..N^l_{max}, 0..T_{max}-1]$$

with the rule: $COMM[i,j,k] = 1$ if and only if the communication number i begins at step k on the link j. The physical links are known statically, and we assume that they are ordered so that

- links numbered from 1 to N^l_{east} go *east*
- links numbered from $N^l_{east}+1$ to N^l_{west} go *west*
- links numbered from $N^l_{west}+1$ to N^l_{south} go *south*
- links numbered from $N^l_{south}+1$ to $N^l_{north} = N^l_{max}$ go *north*

The total number of links is N^l_{max}. There are N^c_{max} communications to process. As for the "time" vector, we consider a "link" vector $LINK$ of size N^l_{max}, which is used to enumerate all the links,

$$LINK = (1, 2, \ldots, N^l_{max})$$

The direction of each inter-cell communication is recorded in the array

$$DIR[1..N^c_{max}, 1..4]$$

by storing a 1 in the adequate column (and 0 elsewhere). We obtain the starting time of communication i by computing the following dot product

$$T^c_{beg}[i] = \sum_{j=1}^{N^l_{max}} \sum_{k=1}^{T_{max}} COMM[i,j,k] \times TIME[k] = \sum_{j=1}^{N^l_{max}} COMM[i,j,\cdot]^t \cdot TIME$$

and the number of the link used by the communication i is:

$$Link[i] = \sum_{j=1}^{N^l_{max}} \sum_{k=0}^{T_{max}-1} COMM[i,j,k] \times LINK[j] = \sum_{k=0}^{T_{max}-1} COMM[i,\cdot,k]^t \cdot LINK$$

We must check that each communication begins once and only once, and that two communications on the same link do not overlap. This gives the following constraints:

$$\begin{array}{ll} \forall i,j,k, & 0 \leq COMM[i,j,k] \leq 1 \\ \forall i, & \sum_{j=1}^{N^l_{max}} \sum_{k=0}^{T_{max}-1} COMM[i,j,k] = 1 \\ \forall j,k, & 0 \leq j \leq N^l_{max},\ 0 \leq k \leq T_{max} - \tau_{comm}, \\ & \sum_{i=1}^{N^c_{max}} \sum_{l=k}^{k+\tau_{comm}-1} COMM[i,j,l] \leq 1 \end{array}$$

To check that each communication is mapped on a link that goes in the right direction, we must impose, for each communication i, the following condition:

$$DIR[i,\cdot]^t \cdot \begin{pmatrix} 1 \\ N^l_{east}+1 \\ N^l_{west}+1 \\ N^l_{south}+1 \end{pmatrix} \leq LINK[i] \leq DIR[i,\cdot]^t \cdot \begin{pmatrix} N^l_{east} \\ N^l_{west} \\ N^l_{south} \\ N^l_{north} \end{pmatrix}$$

The last set of constraints must ensure that dependences between two communications and dependences between communications and computations are respected, which can be easily expressed as we have the expression of $T^p_{beg}[i]$ for any computation i and $T^c_{beg}[j]$ for any communication j.

For instance if communication i_1 depends upon communication i_2 (in fact, the copy of i_2 coming from the previous tile in the direction of i_2), we get the constraint:

$$T^c_{beg}[i_2] + DIR[i_2,\cdot]^t \cdot \begin{pmatrix} -T_1 \\ T_1 \\ -T_2 \\ T_2 \end{pmatrix} + \tau_{comm} \leq T^c_{beg}[i_1]$$

We have a similar set of constraints for dependences between computations and communications.

Objective function

As already stated, the objective function is approximated by $(M_1 - 1) \times |T_1| + (M_2 - 1) \times |T_2|$, which is not a linear expression. Fortunately, we have another way to express the optimization criteria which amounts to "minimize the total latency"

$$\min_{T_1,T_2} \max_{\substack{0 \leq x,x' \leq M_1 \\ 0 \leq y,y' \leq M_2}} (xT_1 + yT_2 - x'T_1 - y'T_2)$$

We can transform this problem into a single linear problem by using the duality theorem as explained in [7, 8]:

$$\left\{ \begin{array}{l} Ap \leq b \\ Aq \leq b \\ \max X(p-q) \end{array} \right. \iff \left\{ \begin{array}{l} X_1 A = X \\ X_2 A = -X \\ X_1 \geq 0 \\ X_2 \geq 0 \\ \min (X_1 + X_2) b \end{array} \right.$$

Clearly, our ILP problem formulation is much more powerful than the one of [4] and will lead to optimal solutions in the two examples given in Section 2.3.

However, a direct solution to our ILP problem is very expensive because many variables as forced to be integral. We have tried several implementations of integer linear problem solvers (lp-solve: Unix public domain routine, PIP [9], Omega [16]) and we have not been able to solve problems with more than 20 points in each tile (for example, the ILP problem expressed for the example of Section 2.1 using small two-dimensional tiles of size 2×2 is composed of approximately 1200 unknowns and 2500 equations). This complexity justifies the use of heuristics. It might be more efficient to try standard branch-and-bound heuristics (as in [23]) than to directly use those in [4]. Another possibility is to solve first the problem with an unlimited number of communication links, then to use the computation ordering deduced from the solution of this problem to apply Chou and Kung's heuristics.

Also, the complexity of the problem motivates the search for an analytical exact solution in some particular cases. That is what we propose for linear tiles in the following section.

4 Scheduling unidimensional graphs

We suppose in this section that we are given a unidimensional uniform dependence graph, with all dependence vectors going in the same direction *east* (otherwise, the graph would not be schedulable). Such a graph may come, for instance, from the single loop:

```
DO i=1,N
   A[i]=f(A[i-3],A[i-7])
ENDDO
```

It may also come from a multidimensional nested loop where the computation points are built from all the inner loops so as to increase granularity.

We want to map this graph onto a unidimensional processor array. Each processor will be assigned a linear tile of computation points. More precisely, each tile will include n consecutive computation points of the graph (see figure 4).

If we call T_1 the offset between the starting times of two consecutive tiles, tile x will begin its computation at time xT_1. Thus minimizing the total execution time is roughly equivalent to minimizing T_1 (if we assume that the number n_1 of tiles is not too small compared to n - this is the same approximation as in Section 3.1). This scheduling problem can (in theory) be solved by using the integer linear programming formulation shown in the previous section. However, this solution is not satisfactory because the number of variables grows very quickly as $O(n^2)$, and the problem becomes intractable, unless heuristics are used.

Our goal in this section is to analytically determine the best possible value for T_1. However, we restrict ourselves to the case where communication resources are unlimited. In the particular case of linear processor arrays, this hypothesis is not very restrictive, as $\lfloor \frac{\tau_{comm}}{\tau_{calc}} \rfloor$ links are sufficient to eliminate all delays potentially due to communications. When targeting VLSI processor arrays, we even would be satisfied with a single communication link in the likely many cases where the inequality $\tau_{comm} \leq$

τ_{calc} holds. This section is organized as follows: first, we introduce some definitions and preliminary results. Then we solve the problem in the simple case of a single dependence vector. Finally we generalize to arbitrary unidimensional dependence graphs.

4.1 Definition and preliminary results

Definition 1 *A **schedule** S is a set of starting times for the n points within a tile. A schedule is **valid** if:*

- *there is at most one computation during each time interval of length τ_{calc}*
- *dependences are respected*

*The efficiency of a schedule S is given by the **period** $T_1(S)$ which is the elapsed time between the starting time of a tile and the starting time of the following tile.*

First we prove Lemma 1 which allows us to "normalize" the computation time and the communication time.

Lemma 1 *The best schedule for $\tau_{calc} = 1$ and $\tau_{comm} = 0$ gives an order for executing the points of a tile which leads to the best schedule for any positive value of τ_{calc} and τ_{comm}.*

Proof: Let $S(\tau^0_{calc}, \tau^0_{comm})$ be the set of valid schedules for $\tau_{calc} = \tau^0_{calc}$ and $\tau_{comm} = \tau^0_{comm}$. Let $S_1 \in S(\tau^0_{calc}, \tau^0_{comm})$. As dependence vectors all go in the same direction, we can apply the cut theorem [17] and consider that the communication cost is 0. We obtain a schedule $S_2 \in S(\tau^0_{calc}, 0)$ where the points of a tile are executed in the same order, with $T_1(S_2) = T_1(S_1) - \tau^0_{comm}$. Then we can choose a clock step such that $\tau^0_{calc} = 1$. Thus we obtain a schedule $S_3 \in S(1,0)$ with $T_1(S_3) = \frac{T_1 - \tau^0_{comm}}{\tau^0_{calc}}$.

Reciprocally, given a schedule $S_4 \in S(1,0)$, it is obvious to derive a valid schedule $S_5 \in S(\tau^0_{calc}, \tau^0_{comm})$ with the same execution ordering and $T_1(S_5) = \tau^0_{calc} \times T_1(S_4) + \tau^0_{comm}$.

As the two transformations described are non-decreasing functions of T_1, the optimal schedule in $S(\tau^0_{calc}, \tau^0_{comm})$ and in $S(1,0)$ are obtained for the same order. ■

For the sake of clarity, we let $\tau_{calc} = 1$ and $\tau_{comm} = 0$ in the following. Thus we just have to check that any two execution steps are different and that dependences are respected. Moreover, we will consider only schedules where all points of a tile are executed consecutively:

Lemma 2 *Given a valid schedule S_1, there exist a valid schedule S_2 such that all points of a tile are executed consecutively and $T_1(S_2) \leq T_1(S_1)$*

Proof: If there is a "hole" in schedule S_1, i.e. no point is scheduled at a given step i whereas some points are scheduled later, then we simply delete the holes and construct S_2 so that points are executed consecutively, with the same order as in S_1. Clearly, S_2 is a valid schedule and its period is not greater than $T_1(S)$. ■

4.2 A small example

The problem to solve now has two kinds of parameters. First, the number and lengths of the dependence vectors, second, the size of each tile. It turns out that even the problem with a single dependence vector is not obvious to solve. Indeed, consider the problem with one dependence vector of length $l = 3$, the size of the tile being $n = 7$. We number the points of the tile from 1 to n and we let t_i be the execution time of point i.

The first obvious valid schedule is the naive schedule: $t_i = i$. This schedule is represented on Figure 5. We see that we cannot compute the first point of the second tile before step 6 thus, for this schedule, $T_1 \geq 5$. In fact we check that $T_1 = 5$ leads to a valid solution. In general, if the period T is chosen so that the l first points of the second tile are scheduled correctly , than T is a valid period for the schedule (one does not have to check dependences for all the other points of the second tile).

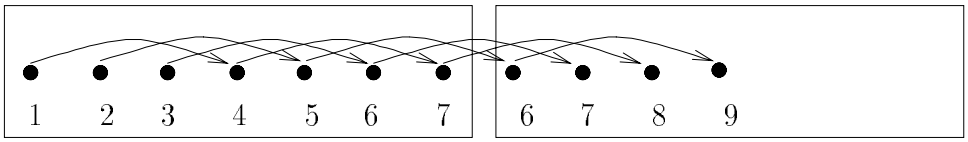


Figure 5: The naive scheduling

A greedy approach would be the following: if we want to minimize the period, we should try to start the second tile as soon as possible (assuming we choose to start the first tile with its first point). To achieve this goal we compute all points in the dependence path leading to the first point of the second tile as soon as possible. This is done in Figure 6. We take $t_2 = 2$ and $t_5 = 3$ so that the first point of the second tile can be executed at step 4. But we have to cope with the cyclicity constraint. Indeed, as we have scheduled the second point of the first tile just after the first point, this property must remain true in the second tile, thus the second point of the second tile must be executed at step 5. This is not possible as some dependences are necessarily violated. Thus $T_1 = 3$ cannot be reached with this strategy.

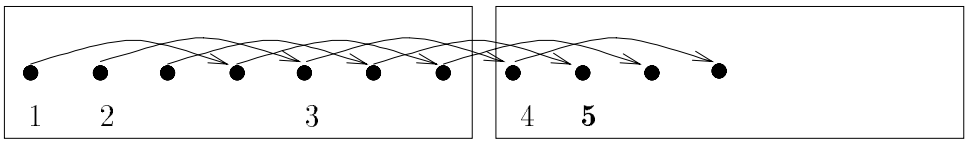


Figure 6: A wrong schedule

In fact, we have two problems: one is to find the minimum period that we can find for a given problem; the other is to find a schedule which achieves this period. In our example, we shall show that the minimum period is $T_1 = 4$, which is obtained by the schedule of figure 7.

The last thing to point out is that the respective values of n and l (if we have one dependence vector) are not independent. If $n \wedge l \neq 1$ [2], consider the reduced graph of

[2] We write $gcd(u, v) = u \wedge v$

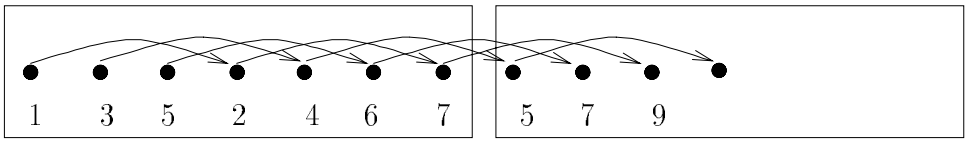


Figure 7: The optimal schedule

Figure 8 where a single tile is represented: dependences going out of the tile are re-sent as inputs to the same tile. We have $n \wedge l$ connected components in this reduced graph. Thus, the cyclicity condition has no impact on the schedule of two points belonging to distinct connected component. It means exactly that we can *independently* schedule each connected component (see Figure 9). The problem is then reduced to scheduling a smaller tile whose size is relatively prime to the dependence vector length.

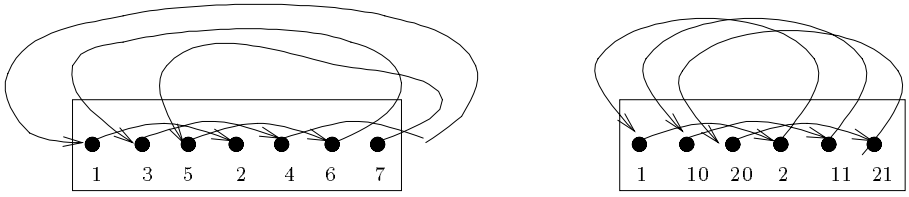


Figure 8: Connected components when $l \wedge n = 1$ and when $l \wedge n = 3$

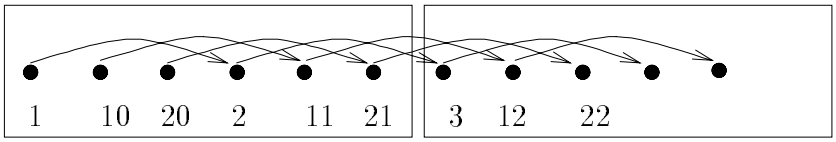


Figure 9: Independent schedules for the connected components ($l \wedge n = 3$)

4.3 Scheduling a linear tile with one dependence vector

Each tile contains n points, and the length of the dependence vector is l. Thanks to Lemmas 1 and 2 we suppose that $\tau_{calc} = 1$ and $\tau_{comm} = 0$. Also, following the previous discussion, we can restrict ourselves to the case $n \wedge l = 1$ without loss of generality. To simplify notations, we write T instead of T_1 to denote the period between the computation of two consecutive tiles.

4.3.1 Case $l = 2$

We begin with a simple case, *i.e* the length of the dependence vector is 2. The number n of points in a tile is odd ($n = 2k + 1$ to fulfill the hypothesis $n \wedge l = 1$).

Theorem 1 *For $n = 2k+1, k \geq 0$ and $l = 2$, the optimal scheduling has a period T_{opt} of:*

$$T_{opt} = \left\lceil \frac{3n-1}{4} \right\rceil$$

Remark Remember that in Theorem 1 we assume $\tau_{calc} = 1$ and $\tau_{comm} = 0$. Using Lemma 1, we obtain

$$T_{opt} = \left\lceil \frac{3n-1}{4} \right\rceil \tau_{calc} + \tau_{comm}$$

in the general case.

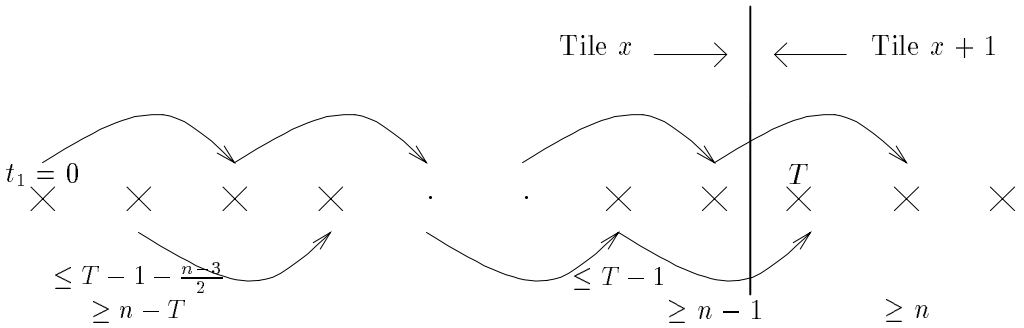


Figure 10: $n = 2k+1, l = 2$

Proof The first computed point in the tile is the first or the second, *i.e* $t_1 = 0$ or $t_2 = 0$. We first suppose that $t_1 = 0$. The last point to be computed is the ultimate or the pen-ultimate point, *i.e* $t_{n-1} = n-1$ or $t_n = n-1$ (we see that all the points can be executed consecutively). If $t_{n-1} = n-1$, the computation of the second tile begins at least at step n and $T \geq n$. It is not a good solution (not faster that if all the points were computed sequentially). We consider the second case, $t_n = n-1$ (see Figure 10.)

$t_n = n-1$ and there is a dependence between the last point of a tile and the second point of the next tile, so

$$t_2 + T \quad \geq \quad n \tag{5}$$

The first point of the second tile is computed at step T. There is a dependence between the pen-ultimate point of the first tile and the first point of the second tile so, $t_{n-1}+1 \leq T$ There are $\frac{n-3}{2}$ arcs in the dependence path to go from the second point of the tile to the pen-ultimate one, hence

$$t_2 + 1 + \frac{n-3}{2} \quad \leq \quad T \tag{6}$$

Inequalities 1 and 2 give us:

$$\begin{array}{rcl} n - T & \leq & T - 1 - \frac{n-3}{2} \\ 2T & \geq & n + 1 + \frac{n-3}{2} \\ T & \geq & \frac{3n-1}{4} \end{array}$$

If $t_2 = 0$, with a similar demonstration, we obtain:

$$T \geq \frac{3n+1}{4}$$

T is, at best, equal to $\lceil \frac{3n-1}{4} \rceil$. If we can find a solution with $T = \lceil \frac{3n-1}{4} \rceil$, it will be an optimal scheduling.

Example of a solution with $T = \lceil \frac{3n-1}{4} \rceil$

- for j even, $t_j = \lceil \frac{3n-1}{4} \rceil - \frac{n+1-j}{2}$
- for j odd,
  - if $j < 2(\lceil \frac{3n-1}{4} \rceil + 1) - n$, $t_j = \frac{j-1}{2}$
  - if $j \geq 2(\lceil \frac{3n-1}{4} \rceil + 1) - n$, $t_j = \frac{j+n-2}{2}$

The first point of the first tile is executed at step $t_1 = 0$, the first point of the second one is executed at step T. Starting from T (first point of the next tile), dates of execution are decremented backwards the path dependence, they are incremented from 0 to $n-1$ along the other path dependence in taking into account that some dates have already been used. For an example of such a solution, see Figure 11.

×	×	×	×	×	×	×	×	×	×	×
0	3	1	4	2	5	7	6	8	7	10

Figure 11: Example of optimal scheduling for $n = 9$ and $l = 2$

By construction, all dependences inside the tiles are respected.

$$t_n = n - 1$$
$$t_2 + T = 2\lceil \frac{3n-1}{4} \rceil - \frac{n-1}{2}$$
$$t_2 + T \geq 2\frac{3n-1}{4} - \frac{n-1}{2} = n$$
$$t_2 + T > t_n$$

Besides,

$$t_1 + T > t_{n-1}$$

All dependences are respected. Thus, the solution is feasible and it is an optimal solution.

■

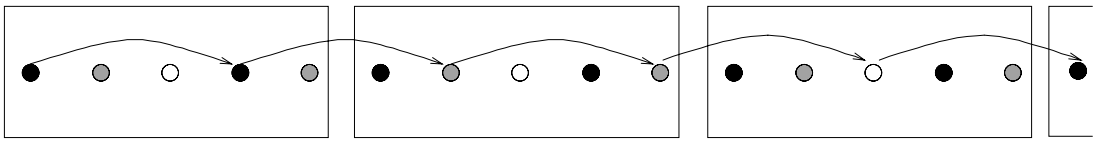
4.3.2 Case $l \geq 3$

We consider now tiles of n points and with one dependence vector of length l, $n \wedge l = 1$. We suppose $n = lp + k$, k, p integers and $0 < k < l0$. Assuming that $n \wedge l = 1$, there is only one connected component in the reduced graph but, inside each tile, the dependence paths define l components. As n and l are relatively prime, the components are not independent and during the transition from a tile to the next, there is a change of connected component. We call $X_{i_1}, X_{i_2}, ..., X_{i_l}$ the different components. The indices $i_1, i_2, ..., i_l$ are chosen in such a such a way that:

- X_{i_1} is the component whose first point is the first point executed in the tile
- during the transition from a tile to the next, there is a dependence from the last point of X_{i_1} to the first point of X_{i_l}, from the last point of X_{i_l} to the first point of $X_{i_{l-1}}$, ..., from the last point of X_{i_2} to the first point of X_{i_1}.

$$X_{i_1} \rightarrow X_{i_l} \rightarrow ... \rightarrow X_{i_2} \rightarrow X_{i_1}$$

See Figure 12 when $n = 5$ and $l = 3$.



● First component (X_{i_1})
● Second conponent (X_{i_3})
○ Third conponent (X_{i_2})

Figure 12: Components for $n = 5$ and $l = 3$

The length of a component can be $p - 1$ or p according to the value of k (there are k components of length p, the others are of length $p - 1$). In the case of Figure 12, there are two components of length p and one of length $p - 1$.

We define the l-tuple $(\lambda_1, \lambda_2, ..., \lambda_l)$ so that $\lambda_j = 1$ if the length of the component X_{i_j} is p, and 0 if its length is $p - 1$, i.e $\lambda_j = |X_{i_j}| - (p - 1)$.

We define the values (t_1, t'_1), (t_1, t'_1),...,(t_j, t'_j) ,...,(t_l, t'_l) so that the first point of the component X_{i_j} is executed at step t_j in the first tile and its last point at step t'_j.

Lemma 3 *We have the following inequalities between the beginning and the finishing dates of the connected components:*

$$\begin{array}{lll} \forall\, 1 \leq j \leq l-1, & & \\ (i) & t'_{j+1} & \leq t_j + T - 1 \\ (ii) & t'_j & \geq t_j + (p-1) + \lambda_j \end{array}$$

Proof (i) During the transition from a tile to the next there is a dependence between the last point of the component $X_{i_{j+1}}$ and the first point of the component X_{i_j} ($X_{i_{j+1}} \rightarrow X_{i_j}$). The last point of the component $X_{i_{j+1}}$ of tile x is executed at $t'_{j+1} + xT$ and the first point of the component X_{i_j} of tile $x+1$ at $t_j + (x+1)T$ so,

$$t'_{j+1} + xT \leq t_j + (x+1)T - 1$$

Thus (i) is demonstrated.

(ii) The length of the component X_{i_j} is $p - 1 + \lambda_j$. There are at least $p - 1 + \lambda_j$ tops between the execution of the first point of the component X_{i_j} and the execution of the last point. This gives (ii). ■

We want to find lower bounds for T. Let X_{i_f} be the component which contains the last point executed in a tile.

If $f \neq 1$, $X_{i_f} \rightarrow X_{i_{f-1}} \rightarrow ... \rightarrow X_{i_2} \rightarrow X_{i_1}$. $t_1 = 0$, $t'_f = n - 1$.

With Lemma 3, we have:

$$\begin{array}{rcl}
n - 1 \leq & t'_f & \leq t_{f-1} + T - 1 \\
t_{f-1} + (p-1) + \lambda_{f-1} \leq & t'_{f-1} & \leq t_{f-2} + T - 1 \\
 & \cdot & \\
 & \cdot & \\
t_3 + (p-1) + \lambda_3 \leq & t'_3 & \leq t_2 + T - 1 \\
t_2 + (p-1) + \lambda_2 \leq & t'_2 & \leq T - 1
\end{array}$$

Summing up all previous inequalities, we obtain:

$$\begin{array}{rcl}
(f-1)T & \geq & n - 1 + (f-2)p + \sum_{i=2}^{f-1} \lambda_i \\
T & \geq & \frac{l+f-2}{f-1} p + \frac{k-1+\sum_{i=2}^{f-1} \lambda_i}{f-1} \\
T & \geq & 2p + \frac{k-1+\sum_{i=2}^{f-1} \lambda_i}{f-1}
\end{array}$$

Hence, if $f \neq 1$, we obtain $T \geq 2p$.

In the same way, if $f = 1$, i.e the same component X_{i_1} contains the first and the last point executed in a tile, we obtain:

$$\begin{array}{rcl}
lT & \geq & n + (l-1)p + k + \sum_{i=2}^{l} \lambda_i \\
T & \geq & \frac{2l-1}{l} p + \frac{k-1}{l} + \frac{sum_{i=2}^{l} \lambda_i}{l}
\end{array}$$

We want to improve this bound and to show that even when $f = 1$ (the same component begins and finishes) we have a lower bound in $2p$ for T.

Lemma 4 *If the same component X_{i_1} contains the first and the last executed point in a tile, we have the following inequalities:*

$$\begin{array}{l}
\forall\, 2 \leq j \leq l, \\
n + (l-j)p - (l-j+1)T + \sum_{i=j+1}^{l} \lambda_i \leq t_j \leq (j-1)(T-p) - \sum_{i=2}^{j} \lambda_i \\
n - 1 + (l-j+1)(p-T) + \sum_{i=j}^{l} \lambda_i \leq t'_j \leq (j-1)(T-p) + p - 1 - \sum_{i=2}^{j-1} \lambda_i
\end{array}$$

Proof By induction on j.

$t'_2 \leq t_1 + T - 1 \leq T - 1$, $t_2 \leq t'_2 - (p-1) - \lambda_2 \leq T - p - \lambda_2$. For the upper bounds, the inequalities are true for j=2. Assuming that they are satisfied for j, because of Lemma 3:

$$\begin{array}{lcl} t'_{j+1} & \leq & (j-1)(T-p) - \sum_{i=2}^{j} \lambda_i + T - 1 \\ t'_{j+1} & \leq & j(T-p) + p - 1 - \sum_{i=2}^{j} \lambda_i \\ t_{j+1} & \leq & j(T-p) + p - 1 - \sum_{i=2}^{j} \lambda_i - (p-1) - \lambda_{j+1} \\ t_{j+1} & \leq & j(T-p) - \sum_{i=2}^{j+1} \lambda_i \end{array}$$

The upper inequalities are satisfied for j+1.

$t_l \geq t'_1 - T + 1 \geq n - T$, $t'_l \geq t_l + p - 1 + \lambda_l \geq n - T + p - 1 + \lambda_l$. For the lower bounds, the inequalities are true for j=l. Assuming that they are true for j+1, because of Lemma 3

$$\begin{array}{lcl} t_j & \geq & n - 1 + (l-j)(p-T) + \sum_{i=j+1}^{l} \lambda_i - T + 1 \\ t_j & \geq & n + (l-j)p - (l-j+1)T + \sum_{i=j+1}^{l} \lambda_i \\ t'_j & \geq & n + (l-j)p - (l-j+1)T + \sum_{i=j+1}^{l} \lambda_i + (p-1) + \lambda_j \\ t'_j & \geq & n - 1 + (l-j+1)(p-T) + \sum_{i=j}^{l} \lambda_i + (p-1) \end{array}$$

The lower inequalities are satisfied for j. ■

Lemma 5 *Assuming that the same component begins and finishes, if there exists j such that* $t'_j \geq t'_{j+1}$ *then* $T \geq 2p + \frac{k + \sum_{i=2}^{l} \lambda_i - \lambda_j}{l-1}$

Proof Assuming $\exists j / t'_j \geq t'_{j+1}$:

The inequalities of the previous lemma as satisfied and we have:

$$\begin{array}{lcl} t'_{j+1} & \geq & n - 1 + (l-j)(p-T) + \sum_{i=j+1}^{l} \lambda_i \\ t'_{j+1} & \geq & lp + k - 1 + (l-j)(p-T) + \sum_{i=j+1}^{l} \lambda_i \\ t'_{j+1} & \geq & 2lp + k - 1 - jp + jT - lT + \sum_{i=j+1}^{l} \lambda_i \end{array}$$

and

$$t'_j \leq (j-1)(T-p) + p - 1 - \sum_{i=2}^{j-1} \lambda_i$$

so,

$$\begin{array}{lcl} 2lp + k - 1 - jp + jT - lT + \sum_{i=j+1}^{l} \lambda_i & \leq & jT - jp - T + p + p - 1 - \sum_{i=2}^{j-1} \lambda_i \\ 2lp + k - lT + \sum_{i=j+1}^{l} \lambda_i & \leq & 2p - T - \sum_{i=2}^{j-1} \lambda_i \\ 2(l-1)p + k + \sum_{i=2}^{l} \lambda_i - \lambda_j & \leq & (l-1)T \\ T & \geq & 2p + \frac{k + \sum_{i=2}^{l} \lambda_i - \lambda_j}{l-1} \end{array}$$

■

Lemma 6 *Assuming that for any* j, $1 \leq j \leq l-1$ *we have* $t'_j \leq t'_{j+1}$ *then*

$$T \geq 2p - 1$$

Proof We assume that $\forall\ 1 \leq j \leq l-1, t'_j \leq t'_{j+1}$ and $T \leq 2p-1$.

We call $(u_1, u_2, ..., u_{l-1})$ the number of points of the components $(X_{i_1}, X_{i_2}, ..., X_{i_{l-1}})$ executed between t_l and t'_{l-1}.

$$\begin{array}{lcl} t'_l & \leq & t_{l-1} + T - 1 \\ t'_l & \geq & t_l + p - 1 + \lambda_l + \sum_{i=1}^{l-1} u_i \\ t_{l-1} & \geq & t_l + p - 1 + \lambda_l + \sum_{i=1}^{l-1} u_i - T + 1 \\ t_l - t_{l-1} & \leq & T - p - \lambda_l - \sum_{i=1}^{l-1} u_i \end{array}$$

If $T \leq 2p-1$ then,

$$\begin{array}{lcl} t_l - t_{l-1} & \leq & p - 1 - \lambda_l - \sum_{i=1}^{l-1} u_i \\ t_l - t_{l-1} & \geq & p - 1 + \lambda_{l-1} - u_{l-1} \end{array}$$

so,

$$\begin{array}{lcl} p - 1 - \lambda_l - \sum_{i=1}^{l-1} u_i & \geq & p - 1 + \lambda_{l-1} - u_{l-1} \\ \sum_{i=1}^{l-2} u_i & \leq & -\lambda_l - \lambda_{l-1} \end{array}$$

So, $\sum_{i=1}^{l-2} u_i = 0$ and $u_1 = u_2 = ... = u_{l-2} = 0$ (see Figure 13)

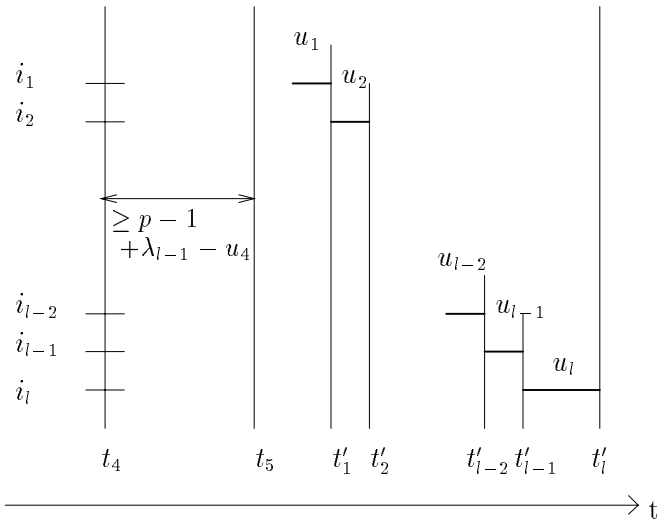


Figure 13: $T \geq 2p-1$

$$\begin{array}{lcl} t_l - t_{l-1} & \geq & p - 1 + \lambda_{l-1} - u_{l-1} \\ t'_l - t_l & \geq & p - 1 + \lambda_l + u_{l-1} \end{array}$$

So, adding the two inequalities, we obtain:

$$t'_l - t_{l-1} \geq 2p - 2 + \lambda_l + \lambda_{l-1}$$

Besides,

$$t'_l - t_{l-1} \leq T - 1$$

So,

$$\begin{array}{rcl} T-1 & \geq & 2p-2+\lambda_l+\lambda_{l-1} \\ T & \geq & 2p-1 \end{array}$$

■

Theorem 2 *For $l \geq 3$ and $n \wedge l = 1$, T is asymptotically equal to $2\lfloor \frac{n}{l} \rfloor$. More precisely, we have:*

$$2\lfloor \frac{n}{l} \rfloor - 1 \leq T \leq 2\lfloor \frac{n}{l} \rfloor + 2$$

Proof We see that in any case, $T \geq 2p-1$. If we can find a feasible solution with $T = 2p+2$, the theorem will be proven.

The *cyclic* algorithm The l components are numbered in such a way that $X_{i_1} \longrightarrow X_{i_l} \longrightarrow X_{i_{l-1}} \longrightarrow ... \longrightarrow X_{i_2} \longrightarrow X_{i_1}$. Our strategy is the following: the first component X_{i_1} is wholly executed, then the second, ..., etc until the l^{th}. The second tile begins after $T = \lfloor 2\frac{n}{l} \rfloor + 2$ tops.

In these conditions, we have:

$$\begin{array}{cc} t_{i_1} = 0 & t'_{i_1} = p-1+\lambda_1 \\ t_{i_2} = p+\lambda_1 & t'_{i_2} = 2p-1+\lambda_1+\lambda_2 \\ & \vdots \\ t_{i_j} = (j-1)p+\sum_{i=1}^{j-1}\lambda_i & t'_{i_j} = jp-1+\sum_{i=1}^{j}\lambda_i \\ & \vdots \\ t_{i_{l-1}} = (l-2)p+\sum_{i=1}^{l-2}\lambda_i & t'_{i_{l-1}} = (l-1)p-1+\sum_{i=1}^{l-1}\lambda_i \\ t_{i_l} = (l-1)p+\sum_{i=1}^{l-1}\lambda_i & t'_{i_l} = lp-1+\sum_{i=1}^{l}\lambda_i \end{array}$$

If we fix $T = \lfloor 2\frac{n}{l} \rfloor + 2$, we obtain a feasible solution. By construction, inside each tile, the dependences are respected. Besides,

$$\begin{array}{rcl} t'_j & = & jp-1+\sum_{i=1}^{j}\lambda_i \\ t_{j-1} & = & (j-2)p+\sum_{i=1}^{j-2}\lambda_i \\ t'_j & = & t_j+2p-1+\lambda_{j-1}+\lambda_j \\ t'_j & \leq & t_j+T-1 \end{array}$$

So, during the transition from a tile to the next, the dependences are also respected. All the constraints are respected and this algorithm gives a feasible solution with $T = 2p+2$.
■

4.4 Scheduling a linear tile with several dependence vectors

Consider now a linear tile but with several dependence vectors.

We first demonstrate a preliminary technical lemma that will useful in the following.

Lemma 7 *Let l_1, l_2 be two integers verifying $l_1 < l_2$ and $l_1 \wedge l_2 = 1$:*

for any n, $n \geq (l_1-1)(l_2-1)$, there exist two non negative integers u and v such that $n = l_1 u + l_2 v$.

Proof Let m be an integer between 1 and $l_1 - 1$, let us divide ml_2 by l_1:

$$ml_2 = q_m l_1 + r_m,\ 1 \leq r_m \leq l_1 - 1$$

$r_m \neq 0$ because $l_1 \wedge l_2 = 1$ and the r_m are all different otherwise by difference there would be a contradiction too with the condition $l_1 \wedge l_2 = 1$.

$r_m > 0$ so,

$$\begin{gathered} q_m l_1 < ml_2 \\ q_m < \tfrac{m}{l_1} l_2 \\ q_m < l_2 \end{gathered}$$

Let us consider the interval $I = [(l_1 - 1)(l_2 - 1), (l_1 - 1)l_2 - 1]$. I contains $l_1 - 1$ consecutive integers. Each integer in I can be written $(l_1 - 1)(l_2 - 1) + r_m - 1$ with all the r_m for $1 \leq m \leq l_1 - 1$ (the r_m are distinct integers between 1 and $l_1 - 1$, $l_1 - 1$ is the length of the interval).

$$\begin{array}{rcccc} (l_1 - 1)(l_2 - 1) + r_m - 1 & = & l_1 l_2 - l_1 - l_2 & + & r_m \\ & = & l_1 l_2 - l_1 - l_2 & + & ml_2 - q_m l_1 \\ & = & (l_2 - 1 - q_m) l_1 & + & (m-1) l_2 \\ & & \geq 0 & & \geq 0 \end{array}$$

All the numbers in I can be written under the form $ul_1 + vl_2$ with $u \geq 0$ and $v \geq 0$. It is also true for the number $(l_1 - 1)l_2$. So, we have l_1 consecutive numbers that can be written under the form $ul_1 + vl_2$ with $u \geq 0$ and $v \geq 0$. It is also true for all numbers greater than these l_1 numbers. ■

Theorem 3 *Consider a linear tile with two dependence vectors of length l_1 and l_2 verifying $l_1 \wedge l_2 = 1$. Then, T is asymptotically equal to the number of points of the tile:*

$$\lim_{n \to \infty} \frac{T}{n} = 1$$

Proof A point must be executed after its predecessors along all dependence paths. If two points p_1 and p_2 are distant from a length d greater than $(l_1 - 1)(l_2 - 1)$, we see with the previous lemma that there is a dependence between p_1 and p_2 (d can be written $ul_1 + vl_2$, $u \geq 0$, $v \geq 0$). Let us consider the point executed at T in the second tile, it depends on at least $n - (l_1 - 1)(l_2 - 1) + 1$ point in the previous tile so,

$$T \geq n - (l_1 - 1)(l_2 - 1) + 1$$

Besides

$$T \leq n$$

so,

$$\lim_{n \to \infty} \frac{T}{n} = 1.$$

■

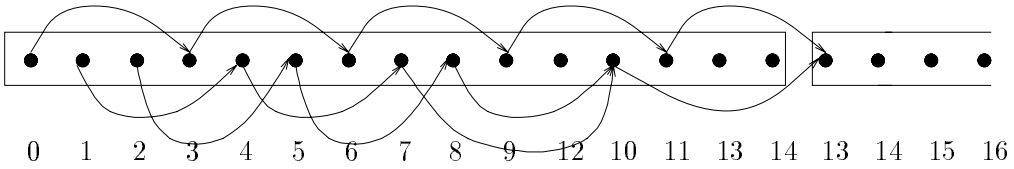


Figure 14: Example of scheduling with two vectors of length $l_1 = 3$, $l_2 = 4$

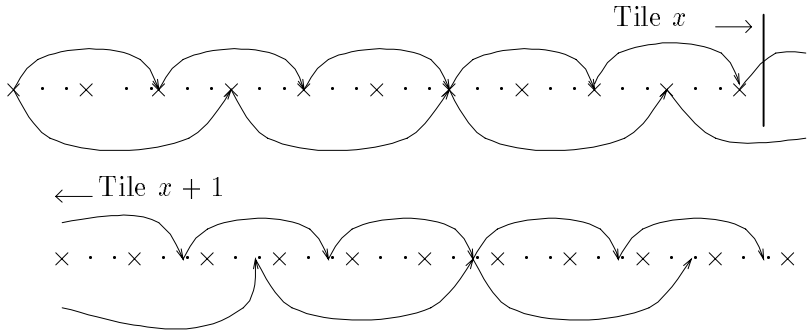


Figure 15: Example: $l_1 = 6$, $l_2 = 9$, $n = 31$

For an example of scheduling with two dependence vectors l_1, l_2 with $l_1 \wedge l_2 = 1$, see Figure 14.

This result basically says that the problem is inherently sequential with two dependence vectors whose lengths are relatively prime.

Theorem 4 *Consider a linear tile with n points and two dependence vectors of size l_1, l_2 such that $l_1 \wedge l_2 = d$, $d \neq 1$ and $n \wedge d$=1. Then,*

(i) *if $d = 2$, T is asymptotically equal to $3n/4$,*

(ii) *if $d \geq 3$ T is asymptotically equal to $2n/d$.*

Proof Since $l_1 \wedge l_2 = d$, as in the case with one dependence vector, there are d components in the path defined by the dependence vectors. As $n \wedge d = 1$, the components are not independent and during the transition from a tile to the next there is a change of component. We define the two integers l'_1 and l'_2 such as $l'_1 = l_1/d$ and $l'_2 = l_2/d$.

We consider now the two following problems:

Problem 1 We consider tiles of n points with one uniform vector of length d. We see from Theorems 1 and 2 that optimal values for the period T_1 are the following:

- if $d = 2$, T_1 asymptotically equal to $3/4n$,
- if $d \geq 3$, T_1 asymptotically equal to $2n/d$.

$l_1 \wedge l_2 = d$, there exist two integers k_1 and k_2 such that $l_1 = k_1 * d$ and $l_2 = k_2 * d$. So, problem 1 is equivalent to our scheduling problem (n points and two dependence vectors) but with some additional dependences, and therefore $T \geq T_1$

Problem 2 We consider tiles of $n' = n - (l_1 - 1)(l_2 - 1) - max(l_1, l_2)$ points and with one uniform dependence vector of length d. As for problem 1, we see that the optimal values for the period T_2 are the following:

- if $d = 2$, $T_2 \approx 3/4n' \approx 3/4n$ [3],
- if $d \geq 3$, $T_2 \approx 2n'/d \approx 2n/d$.

We saw (Lemma 7) that the first point of a component in the basic problem depends on all the $n - (l'_1 - 1)(l'_2 - 1) + 1$ first points in the previous component. Problem 2 is equivalent to our scheduling problem but with less points and with less dependences. So, we have $T \leq T_2$.

$T_1 \leq T \leq T_2$. Besides, T_1 and T_2 have the same asymptotic behavior:

- if $d = 2, T_1 \approx 3/4n, T_2 \approx 3/4n \Longrightarrow T \approx 3/4n$,
- if $d \geq 3, T_1 \approx 2n/d, T_2 \approx 2n/d \Longrightarrow T \approx 2n/d$,

■

Theorem 5 *Consider a linear tile with n points and two dependence vectors of size l_1 and l_2, $l_1 \wedge l_2 = d$ and $n \wedge d = d'$, $d' \neq 1$. Then, the optimal scheduling is obtained for:*

(i) *if $d = d'$, $T \approx n/d$,*

(ii) *if $d/d' = 2$, $T \approx \frac{3n}{4d'}$,*

(iii) *if $d/d' \geq 3$, $T \approx 2n/d$*

Proof The problem is equivalent to d' separate problems with $n' = n/d'$ points and two dependence vectors of length $l'_1 = l_1/d'$, $l'_2 = l_2/d'$. $l'_1 \wedge l'_2 = d/d'$, $n' \wedge d/d'$=1.

(i) If $d = d'$, the result is given by Theorem 3. $T \approx n' = n/d$.

(ii) and (iii) The result is given by Theorem 4. If $d/d' = 2$, $T \approx 3/4n' = \frac{3n}{4d'}$. If $d/d' \geq 3$, $T \approx \frac{2n'}{d/d'} = 2n/d$.

■

Generalization to the case of m dependence vectors Lemma 7 can be generalized to the case of m integers and so, the previous theorems can be generalized to m dependence vectors.

Lemma 8 *Let us consider m integers $(l_1, l_2, ...l_m)$ such that $gcd(l_1, l_2, ...l_m) = 1$. Then, there exists an integer A such that for any $n \geq A$, there exist m non negative integers $\alpha_1, \alpha_2, ..., \alpha_m$ such that:*

$$n = \alpha_1 l_1 + \alpha_2 l_2 + ... + \alpha_m l_m$$

[3] Throughout, we write $f(n) \approx g(n)$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1$

Proof By induction on m. The lemma is true for $m = 2$ (Lemma 7). Let us assume that the lemma is true for $m \geq 2$ and prove that it is also true for $m + 1$.

Let us consider $m + 1$ integers $l_1, l_2, ..., l_m, l_{m+1}$ such as $gcd(l_1, l_2, ...l_m, l_{m+1}) = 1$. $gcd(l_1, l_2, ...l_m) = p$ so $gcd(l_1/p, l_2/p, ...l_m/p) = 1$.

The lemma is true for m integers so, there exists an integer A such that for any $n \geq A$, $n = \alpha_1 l_1/p + \alpha_2 l_2/p + ... + \alpha_m l_m/p$ with $\alpha_i \geq 0$.

$gcd(l_1, l_2, ...l_m, l_{m+1}) = 1$ so p and l_{m+1} are relatively prime.

There exists an integer A' such that for any $n \geq A'$, $n = \alpha p + \beta l_{m+1}$ with $\alpha \geq 0$ and $\beta \geq 0$.

Consider $n \geq A * p + A'$, $n = A * p + m$ with $m \geq A'$. So $\exists(\alpha, \beta)$ such that $m = \alpha p + \beta p$ with $\alpha \geq 0$ and $\beta \geq 0$. $n = A * p + \alpha * p + \beta l_{m+1} = (A + \alpha) * p + \beta l_{m+1}$. But, $\exists(\alpha_1, \alpha_2, ..., \alpha_m)$ such $A + \alpha = \alpha_1 l_1/p + \alpha_2 l_2/p + ... + \alpha_m l_m/p$ so, $n = \alpha_1 l_1 + \alpha_2 l_2 + ... + \alpha_m l_m + \beta l_{m+1}$ with $\alpha_i \geq 0$ and $\beta \geq 0$. ■

Theorem 6 *Generalization: all previous theorems can be generalized to a tile with m dependence vectors, i.e: Consider a tile of n points with m dependence vectors of size* l_1, l_2, ..., l_m, $gcd(l_1, l_2, ..., l_m) = d$ *and* $gcd(n, d) = d'$,

(i) *if* $d = d'$, $T \approx n/d$,

(ii) *if* $d/d' = 2$ $T \approx \frac{3n}{4d'}$,

(iii) *if* $d/d' \geq 3$, $T \approx 2n/d$.

Proof With Lemma 8, all the previous demonstrations can be easily transposed to the case of m dependence vectors. ■

5 Summary and Future Work

Tiling is a quite powerful technique to increase granularity and data locality. The atomicity constraint used by researchers targeting DMPC computers [10, 18, 20, 21, 3] has a great impact upon the simplicity of SPMD code generation. However, such a constraint imposes two restrictions:

- tiles cannot depend upon each other, which restricts the search space for valid schedulings
- the capability of modern DMPC computers to overlap communications and computations of modern DMPC is not taken advantage of

We believe that removing the atomicity constraint can lead to very interesting perspectives, in particular when targeting VLSI processor arrays (see [14] in this respect). The work of Chou and Kung [4] opens new directions for scheduling tiled processor arrays while assuming limited computation and communication resources. Our main contribution in this paper is to have given a formulation of the problem in terms of an ILP problem which takes all constraints into account. Clearly, the size of this ILP problem makes the use of heuristics unavoidable, but the choice of these heuristics can rely upon

the new ILP formulation that leaves the whole solution space open for searching. On the contrary, Chou and Kung's heuristics, while very simple, imposes very restrictive limitations, as shown by the examples given in the paper.

One possibly better heuristic would be first to solve the problem with an unlimited number of links (which is much simpler) so as to derive an initial ordering in the tile and to apply Chou and Kung technique. Another possible solution would be to apply classical linear programming heuristics (like branch and bound for example) to our ILP problem. Anyway, comparing and evaluating heuristics is an interesting direction for future research.

From a theoretical point of view, it would be very interesting to extend the analytical solution given in the case of unidimensional tiles to arbitrary dimensions, as well as to propose optimal algorithms for the general case. However, scheduling problems are known to be difficult [5], and that of multidimensional tiles with limited resources is very challenging !

References

[1] J. B., E.F. Deprettere, and P. Dewilde. A design methodology for fixed-size systolic arrays. In S.Y. Kung and E. Swartzlander, editors, *International Conference on application Specific Array Processing*, pages 591–602, Princeton, New Jersey, sep 1990. IEEE Computer Society.

[2] Jacek Blazewicz, Moshe Dror, and Jan Weglarz. Mathematical programming formulations for machine scheduling : A survey. *European Journal of Operational Research*, 51:283–300, June 1991.

[3] Pierre Boulet, Alain Darte, Tanguy Risset, and Yves Robert. (Pen)-ultimate tiling ? In IEEE Computer Society Press, editor, *Scalable High Performance Computing Conference*, pages 568–576, 1994. Extended version available as Technical Report 93-36, LIP, ENS Lyon (1993).

[4] W.H. Chou and S.Y. Kung. Scheduling partitioned algorithms with limited communication supports. In Luigi Dadda and Benjamin Wah, editors, *Application Specific Array Processors ASAP 93*, pages 53–64. IEEE Computer Society Press, 1993.

[5] Ph. Chretienne. Task scheduling over distributed memory machines. In M. Cosnard, P. Quinton, M. Raynal, and Y. Robert, editors, *Parallel and Distributed Algorithms*, pages 165–176. North Holland, 1989.

[6] Alain Darte. Regular partitioning for synthesizing fixed-size systolic arrays. *INTEGRATION, The VLSI Jounal*, 12:293–304, December 1991.

[7] Alain Darte. *Techniques de parallélisations de nids de boucles*. PhD thesis, ENS-Lyon, avril 1993.

[8] Alain Darte, Leonid Khachiyan, and Yves Robert. Linear scheduling is nearly optimal. *Parallel Processing Letters*, 1(2):73–81, 1991.

[9] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationnelle*, 22:243–268, September 1988.

[10] F. Irigoin and R. Triolet. Supernode partitioning. In *Proc. 15th Annual ACM Symp. Principles of Programming Languages*, pages 319–329, San Diego, CA, January 1988.

[11] K. Jainandunsing. Optimal partitioning scheme for wavefront/systolic array processors. In *IEEE Symposium on Circuits and Systems*, 1986.

[12] S.Y. Kung. *VLSI array processors*. Prentice-Hall, 1988.

[13] Jean B. Lasserre and Maurice Queyrane. Generic scheduling polyhedra and new mixed-integer formulation for single-machine scheduling. In *Integer Programming and Combinatorial Optimization*, pages 136–149, 1992.

[14] J.A. Martens. Partitioning of parametrized dataflow graphs. Technical Report 93-126, Technical University of Delft, 1993.

[15] D.I. Moldovan and J.A.B. Fortes. Partitioning and mapping algorithms into fixed-size systolic arrays. *IEEE Transactions on Computers*, 35(1):1–12, jan 1986.

[16] William Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:102–114, aug 1992.

[17] Patrice Quinton and Yves Robert. *Systolic Algorithms and Architectures*. Prentice Hall, 1991. Translated from French, Masson (1989).

[18] J. Ramanujam. Non-unimodular transformations of nested loops. In *Proc. Supercomputing'92*, pages 214–223. IEEE Computer Society Press, November 1992.

[19] J. Ramanujam and P. Sadayappan. Tiling of iteration spaces for multicomputers. In *Proc. Internal Conference on Parallel Processing*, volume 2, pages 179–186, August 1990.

[20] R. Schreiber and Jack J. Dongarra. Automatic blocking of nested loops. Technical Report 90.38, RIACS, August 1990.

[21] S. Sharma, C.-H. Huang, and P. Sadayappan. On data dependence analysis for compiling programs on distributed-memory machines. *ACM Sigplan Notices*, 28(1), January 1993. Extended Abstract.

[22] B. Sinharoy and N. Szymanski. Finding optimum wavefront of parallel computation. In H. El-Rewini, T. Lewis, and B. D. Shriver, editors, *Proc. of the Twenty-Sixth Annual Hawaii International Conference on System Sciences*, volume 2, pages 225–234. IEEE Computer Society Press, 1993.

[23] L. Thiele. Resource constrained scheduling of uniform algorithms. In Luigi Dadda and Benjamin Wah, editors, *Application Specific Array Processors ASAP 93*, pages 29–40. IEEE Computer Society PressIEEE Computer Society Press, 1993.