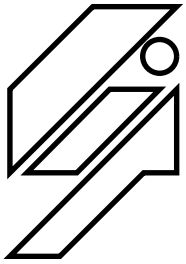
On the optimality of Allen and Kennedy's algorithm for parallelism extraction in nested loops

Alain Darte, Frédéric Vivien



Laboratoire de l'Informatique du Parallélisme

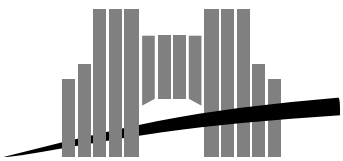
Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

On the optimality of Allen and Kennedy's algorithm for parallelism extraction in nested loops

Alain Darte and Frédéric Vivien

February 1996

Research Report N^{O} 96-05



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

On the optimality of Allen and Kennedy's algorithm for parallelism extraction in nested loops

Alain Darte and Frédéric Vivien

February 1996

Abstract

We explore the link between dependence abstractions and maximal parallelism extraction in nested loops. Our goal is to find, for each dependence abstraction, the minimal transformations needed for maximal parallelism extraction. The result of this paper is that Allen and Kennedy's algorithm is optimal when dependences are approximated by dependence levels. This means that even the most sophisticated algorithm cannot detect more parallelism than found by Allen and Kennedy's algorithm, as long as dependence level is the only information available. In other words, loop distribution is sufficient for detecting maximal parallelism in dependence graphs with levels.

Keywords: nested loops, automatic parallelization, dependence analysis, Allen and Kennedy's algorithm

Résumé

Nous étudions les relations entre représentations des dépendances et extraction maximale du parallélisme dans les nids de boucles. Nous recherchons, pour chaque représentation des dépendances, la plus petite transformation capable d'extraire le maximum de parallélisme. Nous prouvons dans cet article que l'algorithme d'Allen et Kennedy est optimal quand les dépendances sont approximées par des niveaux de dépendance: aucun algorithme, aussi sophistiqué soit-il, ne peut détecter plus de parallélisme que l'algorithme d'Allen et Kennedy, si la seule information disponible sur les dépendances est le niveau de la dépendance. Autrement dit, la distribution de boucles suffit à détecter le maximum de parallélisme dans les graphes de dépendance étiquetés par niveaux.

Mots-clés: nids de boucles, parallélisation automatique, analyse de dépendance, algorithme d'Allen et Kennedy

On the optimality of Allen and Kennedy's algorithm for parallelism extraction in nested loops

Alain Darte Frédéric Vivien

February 1996

Contents

1 Introduction

Many automatic loop parallelization techniques have been introduced over the last 25 years, starting from the early work of Karp, Miller and Winograd [KMW67] in 1967 who studied the structure of computations in repetitive codes called systems of uniform recurrence equations. This work defined the foundation of today's loop compilation techniques. It has been widely exploited and extended in the systolic array community (among others [Mol82, Qui84, Rao85, Roy88, DV94] are directly related to it), as well as in the compiler-parallelizer community: Lamport [Lam74] proposed a parallel scheme - the hyperplane method - in 1974, then several loop transformations were introduced (loop distribution/fusion, loop skewing, loop reversal, loop interchange, ...) for vectorizing computations, maximizing parallelism, maximizing locality and/or minimizing synchronizations. These techniques have been used as basic tools for optimizing algorithms, the most two famous being certainly Allen and Kennedy's algorithm [AK82, AK87], designed at Rice in the Fortran D compiler, and Wolf and Lam's algorithm [WL91], designed at Stanford in the SUIF compiler.

At the same time, dependence analysis has been developed so as to provide sufficient information for checking the legality of these loop transformations, in the sense that they do not change the final result of the program. Different abstractions of dependences have been defined (dependence distance [Mur71], dependence level [AK82, AK87], dependence direction vector [Wol82, Wol89], dependence polyhedron/cone [IT87], ...), and more and more accurate tests for dependence analysis have been designed (Banerjee's tests [Ban88], I test [KKP90, PKK91], Δ test [GKT91], λ test [LYZ89, Gru90], PIP test [Fea91], PIPS test [IJT91], Omega test [Pug92], ...).

In general, dependence abstractions and dependence tests have been introduced with some particular loop transformations in mind. For example, the dependence level was designed for Allen and Kennedy's algorithm, whereas the PIP test is the main tool for Feautrier's method for array expansion [Fea91] and parallelism extraction by affine schedulings [Fea92a, Fea92b]. However, very few authors have studied, in a general manner, the links between both theories, dependence analysis and loop restructurations, and have tried to answer the following two dual questions:

- What is the minimal dependence abstraction needed for checking the legality of a given transformation?
- What is the simplest algorithm that exploits all information provided by a given dependence abstraction at best?

Answering the first question permits to adapt the dependence analysis to the parallelization algorithm, and to avoid implementing an expensive dependence test if it is not needed. This question has been deeply studied in Yang's thesis [Yan93], and summarized in Yang, Ancourt and Irigoin's paper [YAI95].

Conversely, answering the second question permits to adapt the parallelization algorithm to the dependence analysis, and to avoid using an expensive parallelization algorithm if one knows that a simpler one is able to find the same degree of parallelism, and for a smaller cost. This question has been addressed by Darte and Vivien in [DV95] for dependence abstractions based on a polyhedral approximation.

Completing this work, we propose, in this paper, a more precise study of the link between *dependence abstractions* and *parallelism extraction* in the particular case of *dependence levels*. Our main result is that, in this context, Allen and Kennedy's parallelization algorithm is optimal for parallelism extraction, which means that even the most sophisticated algorithm cannot detect more parallel loops than Allen and Kennedy's algorithm does, as long as dependence level is the only information available. In other words, loop distribution is sufficient for detecting maximal parallelism

in dependence graphs with levels. There is no need to use more complicated transformations such as loop interchange, loop skewing, or any other transformations that could be invented, because there is an intrinsic limitation in the dependence level abstraction that prevents detecting more parallelism.

The rest of the paper is organized as follows. In Section 2, we explain what we call maximal parallelism extraction for a given dependence abstraction and we recall the definition of dependence levels. Section 3 presents Allen and Kennedy's algorithm in its simplest form - which is sufficient for what we want to prove. The proof of our result is then subdivided into two parts. In Section 4, we build a set of loops that are equivalent to the loops to be parallelized, in the sense that they have the same dependence graph. In Appendix A, we prove that these loops contain exactly the degree of parallelism found by Allen and Kennedy's algorithm. Finally, Section 5 summarizes the paper.

2 Theoretical framework

We first restrict to the case of perfectly nested loops. We will explain at the end of this paper how our optimality result can be extended to non perfectly nested loops.

2.1 Notations

The notations used in the following sections are:

- $f(N) = O(N)$ if $\exists\ k > 0$ such that $f(N) \leq kN$ for all sufficiently large N.
- $f(N) = \Omega(N)$ if $\exists\ k > 0$ such that $f(N) \geq kN$ for all sufficiently large N.
- $f(N) = \Theta(N)$ if $f(N) = O(N)$ and $f(N) = \Omega(N)$.
- If X is a finite set, $|X|$ denotes the number of elements in X.
- $G = (V, E)$ denotes a directed graph with vertices V and edges E.
- $e = (x, y)$ denotes an edge from vertex x to vertex y. We use the notation: $x = t(e)$, $y = h(e)$.
- I, J denote iteration vectors for nested loops.
- S_i, S_j denote statements within the loops.
- $S_i(I)$ denotes the instance, at iteration I, of statement S_i.
- n is the number of nested loops.
- s is the number of statements within the loops.

2.2 Dependence graphs

The structure of perfectly nested loops can be captured by an ordered set of statements $S_1, \ldots, S_s$ (S_i textually before S_j if $i < j$) and an iteration domain $\mathcal{D} \subset \mathbb{Z}^n$ that describes the values of the loops counters; n is the number of nested loops. Given a statement S, to each n-dimensional vector $I \in \mathcal{D}$ corresponds a particular execution (called instance) of S, denoted by $S(I)$.

EDG, RDG and ADG Dependences (or precedence constraints) between instances of statements define the **expanded dependence graph (EDG)** also called iteration level dependence graph. The vertices of the EDG are all possible instances $\{(S_i, I) \mid 1 \leq i \leq s \text{ and } I \in \mathcal{D}\}$. There is an edge from (S_i, I) to (S_j, J) (denoted by $S_i(I) \Longrightarrow S_j(J)$) if executing instance $S_j(J)$ before instance $S_i(I)$ may change the result of the program. For all $1 \leq i, j \leq s$, one defines the distance sets $E_{i,j}$ as follows:

Definition 1 (Distance Set)

$$E_{i,j} = \{(J - I) \mid S_i(I) \Longrightarrow S_j(J)\} \qquad (E_{i,j} \subset \mathbb{Z}^n)$$

In general, the EDG (and the distance sets) cannot be computed at compile-time, either because some information is missing (such as the values of size parameters or even worse, exact accesses to memory), or because generating the whole graph is too expensive.

Instead, dependences are captured through a smaller, (in general) cyclic directed graph, with s vertices, called the **reduced dependence graph (RDG)** (or statement level dependence graph). Each edge e has a label $w(e)$. This label has a different meaning depending upon the dependence abstraction that is used: it represents [1] a set $D_e \subset \mathbb{Z}^n$ such that:

$$\forall i, j,\ 1 \leq i, j \leq s,\ E_{i,j} \subset \left(\bigcup_{e=(S_i, S_j)} D_e \right) \tag{1}$$

In other words, the RDG describes, in a condensed manner, an iteration level dependence graph, called (maximal) **apparent dependence graph (ADG)**, that is a superset of the EDG. The ADG and the EDG have the same vertices, but the ADG has more edges, defined by:

$$(S_i, I) \Longrightarrow (S_j, J) \text{ (in the ADG)} \Leftrightarrow \exists\ e = (S_i, S_j) \text{ (in the RDG)} \text{ such that } (J - I) \in D_e.$$

Equation 1 and Definition 1 ensure that EDG $\subset$ ADG.

Dependence level abstraction In Sections 3 and 4, we will focus mainly on the case of RDG labeled by one of the simplest dependence abstractions, namely the dependence level. The reader can find a similar study for other dependence abstractions in [DV95].

Definition 2 *The dependence level associated to a dependence distance $J-I$ where $S_i(I) \Longrightarrow S_j(J)$ is:*

- ∞ *if* $J - I = 0$.
- *the smallest integer* l, $1 \leq l \leq n$, *such that the* l*-th component of* $J - I$ *is non zero, otherwise.*

Definition 3 *We call* ***reduced leveled dependence graph (RLDG)*** *associated to a loop nest L, a directed graph with one vertex per statement of L, and one edge e per distance set, where e is labeled by the dependence levels associated to all dependence distances of the distance set.*

Actually, with Definition 3, several values may be associated to a given edge of the reduced leveled dependence graph. To simplify the rest of the paper, we transform each edge labeled by l different levels into l edges with a single level. Therefore, in the following, a reduced leveled dependence graph is a multi-graph, for which each edge e is labeled by an integer $l(e)$ (and $l(e) \in [1 \ldots n] \cup \{\infty\}$). $l(e)$ is called the **level** of edge e.

[1] except for exact dependence analysis where it defines a subset of $\mathbb{Z}^n \times \mathbb{Z}^n$

Example 1 To better understand the links between the three concepts (EDG, RDG and ADG), let us consider a simple example, the SOR kernel:

DO $i = 1, N$
  DO $j = 1, N$
    a(i, j) = a$(i, j-1)$ + a$(i-1, j)$
CONTINUE

The EDG associated to Example 1 is given in Figure 1. The length of the longest path in the EDG is equal to $2 * N - 2$, i.e. $\Theta(N)$. As there are N^2 instances in the domain, we will say that the degree of intrinsic parallelism in this EDG is 1.

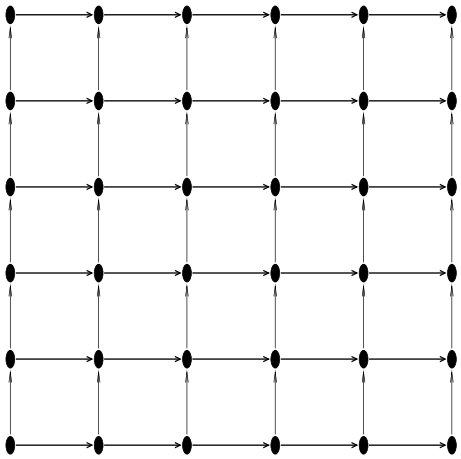


Figure 1: EDG for Example 1

The RDG has only one vertex. If it is labeled with dependence levels (i.e. if it is a RLDG), it has two edges with levels 1 and 2 (see Figure 2). It corresponds to the ADG given in Figure 3, whose ADG contains a path of length N^2. We will say that the degree of intrinsic parallelism in the ADG is 0 as there are N^2 instances in the domain.

Figure 2: RLDG for Example 1

Actually, in this example, it is possible to build a set of loops - that we call the apparent loops L' (given below) - that have exactly the same RDG and that are purely sequential: there is indeed a path of length $\Omega(N^2)$ in the corresponding EDG (see Figure 4).

Apparent loops L':

DO $i = 1, N$
  DO $j = 1, N$
    a(i, j) = 1 + a$(i, j-1)$ + a$(i-1, N)$
CONTINUE

Since L and L' cannot be distinguished by a parallelization algorithm (they have the same RLDG) no parallelism can be detected in this example, as long as the dependence level is the only information available. The goal of this paper is to generalize this fact to arbitrary RLDGs.

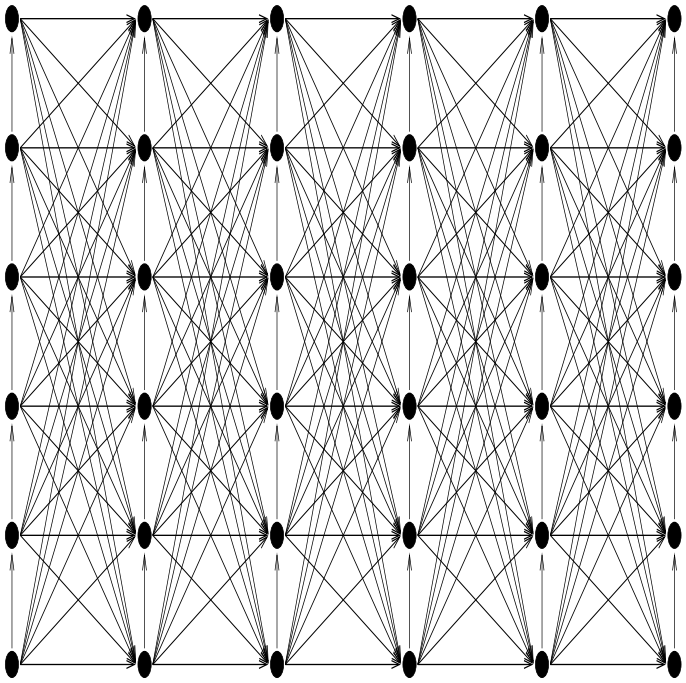


Figure 3: ADG for the RLDG of Example 1

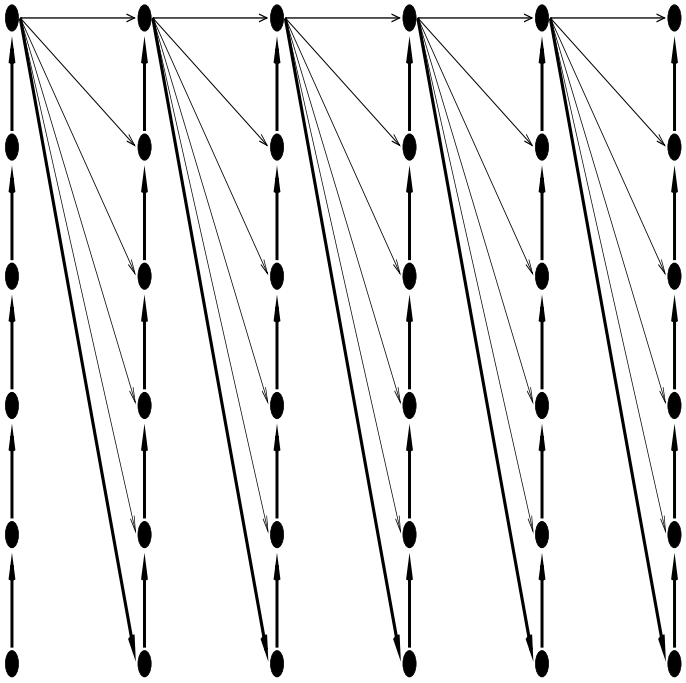


Figure 4: EDG for apparent loops L' of Example 1

2.3 Maximal degree of parallelism

We now define what we call maximal parallelism extraction in reduced dependence graphs.

We consider that the only information available for extracting parallelism in a set of loops L is the RDG associated to L. Any parallelization algorithm that transforms L into an equivalent code L_t has to preserve all dependences summarized in the RDG, i.e. all dependences described in the ADG: if $(S_i, I) \Longrightarrow (S_j, J)$ in the ADG then (S_i, I) must be computed before (S_j, J) in the transformed code L_t.

Definition 4 (Latency)
We define the latency $\mathcal{T}(L_t)$ of a transformed code L_t as the minimal number of clock cycles needed to execute L_t if:

- *an unbounded number of processors is available.*
- *executing an instance of a statement requires one clock cycle.*
- *any other operation requires zero clock cycle.*

Remark For example, if L_t is defined only by a set of parallel and sequential nested loops, the latency can be defined by induction on the structure of the code as follows (";" is the sequencing predicate, DOPAR and DOSEQ define parallel and sequential loops):

- $\mathcal{T}(S) = 1$ if S is a simple statement.
- $\mathcal{T}(S_1; \cdots; S_s) = \sum_{1 \leq i \leq s} \mathcal{T}(S_i)$.
- $\mathcal{T}(\text{DOPAR } i \in \mathcal{D} \text{ do } S(i) \text{ ENDDO}) = \max_{i \in \mathcal{D}} \mathcal{T}(S(i))$.
- $\mathcal{T}(\text{DOSEQ } i \in \mathcal{D} \text{ do } S(i) \text{ ENDDO}) = \sum_{i \in \mathcal{D}} \mathcal{T}(S(i))$.

Of course the latency defined by Definition 4 is not equal to the real execution time. However, it permits to introduce the notion of *degree of parallelism*. The practical interest of this notion is illustrated hereafter.

Since two instances linked by an edge in the ADG cannot be computed at the same clock cycle in the transformed code L_t, the latency of L_t, whatever the parallelization algorithm, is larger than the length of the longest path in the ADG. Now, assume that $\mathcal{D}$ is the n-dimensional cube of size N. Then, we have the following:

- If an algorithm is able to transform the initial loops into a transformed code whose latency is $O(N^d)$, then the length of the longest dependence path in the ADG is $O(N^d)$.
- Equivalently, if the ADG contains a path of length that is not $O(N^d)$, then no matter the parallelizing algorithm you use, the latency of the transformed code cannot be $O(N^d)$.

This short study permits to define a theoretical framework in which the optimality of parallelization algorithms, with respect to a given dependence abstraction, can be discussed. In the following definitions, we assume that all RDG and ADG are defined using the same dependence abstraction.

Definition 5 (degree of intrinsic parallelism of a RDG)
Let G be a RDG and $\mathcal{D}$ be the n-dimensional cube of size N. Let d be the smallest non negative integer such that the length of the longest path in the ADG, defined from G and $\mathcal{D}$, is $O(N^d)$. Then, we say that the degree of intrinsic parallelism in G is $(n-d)$ or that G contains $(n-d)$ degrees of parallelism.

Definition 6 (Degree of parallelism extraction for an algorithm in a RDG)
Let A be a parallelization algorithm. Let L be a set of n nested loops and let G be its RDG. Apply algorithm A to G and suppose that $\mathcal{D}$ is the n-dimensional cube of size N when transforming the loops. Then, the degree of parallelism extraction for A in G is $(n-d)$ if d is the smallest non negative integer such that the latency of the transformed code is $O(N^d)$.

Note that these definitions ensure that the degree of parallelism extraction is always smaller than the degree of intrinsic parallelism.

Definition 7 (Optimal algorithm for parallelism extraction)
An algorithm A performs maximal parallelism extraction (or is said optimal for parallelism extraction) if for each RDG G, the degree of parallelism extraction for A in G is equal to the degree of intrinsic parallelism in G.

With this formulation, the optimality of a parallelization algorithm A can be proved as follows. Consider a set of n perfectly nested loops L. Denote by G the RDG associated to L for the given dependence abstraction and by G_a its corresponding ADG. Let $(n-d)$ be the degree of parallelism extraction for A in G. Then, we have at least two ways for proving the optimality of A.

i. Build in G_a a dependence path whose length is not $O(N^{d-1})$.

ii. Build a set of loops L' whose RDG is also G and whose EDG contains a dependence path whose length is not $O(N^{d-1})$.

Note that (ii) implies (i) since the EDG of L' is included in G_a (L and L' have the same RDG). Therefore, proving (ii) is - a priori - more powerful. In particular, it permits to understand the intrinsic limitations, for parallelism extraction, due to the dependence abstraction itself: even if the degrees of intrinsic parallelism in L and L' may be different at run-time (i.e. their EDG may be different), they cannot be distinguished by the parallelization algorithm, since L and L' have the same RDG. In other words, a parallelization algorithm will parallelize L and L' in the same way. Therefore, since L' is parallelized optimally, the algorithm is considered optimal with respect to the dependence abstraction that is used. Figure 5 recalls the links between L, L' and their EDG, RDG and ADG. The nested loops L' are called **apparent nested loops**.

One can define a more precise notion of optimality. Consider one particular statement S of the initial loops L and define the ***S*-latency** as the minimal number of clock cycles needed to execute the transformed loops L_t, with unbounded number of processors, when any operation needs zero clock cycle except the instances of S that need one clock cycle. Then, the S-latency is related to the ***S*-length** of the longest path in the ADG, where the S-length of a path P is the number of vertices in P that are instances of S. Similarly, one can define the ***S*-degree** of parallelism extraction in L_t and the S-degree of intrinsic parallelism in a RDG. These definitions leads to the following notion of optimality:

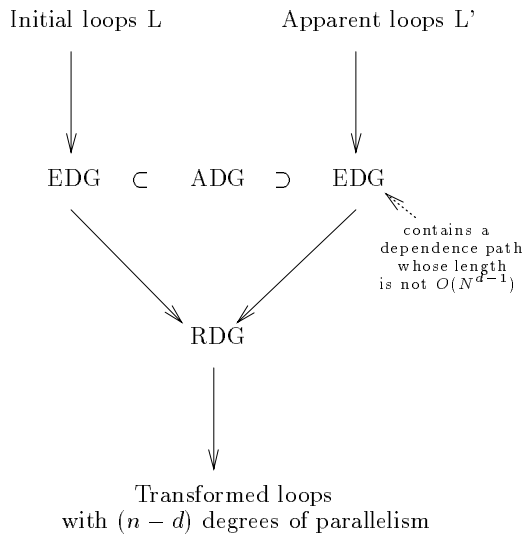


Figure 5: Links between L, L' and their EDG, ADG and RDG

Definition 8 (Optimal algorithm for parallelism extraction (definition 2))
An algorithm A performs maximal parallelism extraction (or is said optimal for parallelism extraction) if for each RDG G and for all statements S of G, the S-degree of parallelism extraction for A in G is equal to the S-degree of intrinsic parallelism in G.

This definition permits to discuss the quality of parallelizing algorithms even for statements that do not belong to the most sequential part of the code. Note that this definition of optimality is more precise than Definition 7 since the degree of intrinsic parallelism (resp. of parallelism extraction) in G is the minimal S-degree of intrinsic parallelism (resp. of parallelism extraction) in G.

One should argue that the latency (and the S-latency) of a transformed code is not easy to compute. Indeed, in the general case, the latency can be computed only by executing the transformed code with a fixed value of N. However, for most known parallelizing algorithms, the degree of parallelism extraction (but not necessarily the latency) can be computed simply by examining the structure of the transformed code, as shown by lemma 1.

Lemma 1 *In addition to the hypotheses of Definition 6, assume that each statement S of the initial code L appears only once in the transformed code L_t and is surrounded by exactly n loops. Furthermore, assume that the iteration domain $\mathcal{D}_t$ described by these n loops contains a n-cube $\underline{\mathcal{D}}$ of size $\Omega(N)$ and is contained in a n-cube $\overline{\mathcal{D}}$ of size $O(N)$.*

Then, the number of parallel loops that surrounds S is the S-degree of parallelism extraction and the minimal S-degree of parallelism extraction is the degree of parallelism extraction.

Proof Consider a given statement S of the initial code L. To simplify the arguments of the proof, define L_r the code obtained by removing from L_t everything that does not involve the instances of S: the latency of L_r is the S-latency of L_t. Furthermore, L_r is a set of n perfectly nested loops that surround statement S. Let $\overline{L}$ (resp. $\underline{L}$) be the code obtained by changing the loop bounds of L_r so that they describe $\overline{\mathcal{D}}$ (resp. $\underline{\mathcal{D}}$) instead of $\mathcal{D}_t$.

Since $\underline{\mathcal{D}} \subset \mathcal{D}_t \subset \overline{\mathcal{D}}$, the latency of L_r is larger than the latency of $\underline{L}$ and smaller than the latency of $\overline{L}$. Furthermore, since $\overline{\mathcal{D}}$ and $\underline{\mathcal{D}}$ are n-cubes, the latency is easy to compute: the latency of $\underline{L}$ is $\Omega(N^d)$ and the latency of $\overline{L}$ is $O(N^d)$, where d is the number of sequential loops that surround S. Therefore, the latency of L_r is $\Theta(N^d)$ and the degree of parallelism extraction in L_r (and thus the S-degree of parallelism extraction in L) is $(n-d)$, i.e. the number of parallel loops that surround statement S.

Let $\mathcal{T}$ and $\mathcal{T}_S$ be respectively the latency and the S-latency of L_t. We have:

$$\max_S \mathcal{T}_S \leq \mathcal{T} \leq \sum_S \mathcal{T}_S$$

Therefore, since the number of statements S is finite, $\mathcal{T} = \Theta(N^d)$ where d is the largest d_S such that $\mathcal{T}_S = \Theta(N^{d_S})$: the degree of parallelism extraction in L_t is the minimum S-degree of parallelism extraction. Equivalently, the degree of parallelism extraction in L_t is the minimum number of perfectly nested parallel loops. ■

We now recall some graphs definitions:

Definition 9 *A **strongly connected component** of a directed graph G is a maximal subgraph of G in which for any vertices p and q $(p \neq q)$ there is a path from p to q.*

Definition 10 *The **acyclic condensation** of a graph G is the acyclic graph whose nodes are the strongly connected components $\mathcal{V}_1, \ldots, \mathcal{V}_c$ of G and there is an edge from $\mathcal{V}_i$ to $\mathcal{V}_j$ if there is an edge $e = (x_i, y_j)$ in G such that $x_i \in \mathcal{V}_i$ and $y_j \in \mathcal{V}_j$.*

Definition 11 *Let G be a reduced leveled dependence graph. Let H be a subgraph of G. Then $l(H)$ (the **level of** H) is the minimal level of an edge of H:*

$$l(H) = \min\{l(e) \mid e \in H\}$$

Remark: not all directed graphs whose edges are labeled by values in $[1 \ldots n] \cup \{\infty\}$ are reduced leveled dependence graphs. A necessary and sufficient condition for such a graph G to be the RLDG of some nested loops is that G_∞, the subgraph of G whose edges are the edges of level ∞, is acyclic. This property is obviously necessary and is proved to be sufficient by lemma 4. This property will be assumed in the next sections. One of the consequences is that the level of a strongly connected subgraph of G is at most n.

3 Allen and Kennedy's algorithm

Allen and Kennedy's algorithm has first been designed for vectorizing loops. Then, it has been extended so as to maximize the number of parallel loops and to minimize the number of synchronizations in the transformed code. It has been shown (see details in [Cal87, ZC90]) that for each statement of the initial code, as many surrounding loops as possible are detected as parallel loops. Therefore, one could think that what we want to prove in this paper has been already proved!

However, looking precisely into the details of Allen and Kennedy's proof reveals that what has actually been proved is the following: consider a statement S of the initial code and L_i one of the surrounding loops. Then L_i will be marked as parallel if and only if there is no dependence at level i between two instances of S. This result proves that the algorithm is optimal among all parallelization algorithms that describe, in the transformed code, the instances of S with exactly the

same loops as in the initial code. This does not prove a general optimality property. In particular, this does not prove that it is not possible to detect more parallelism with more sophisticated techniques than loop distribution and loop fusion. This paper gives an answer to this question.

First, we recall Allen and Kennedy's algorithm, in a very simple form, since we are interested only in detecting parallel loops and not in the minimization of synchronization points.

Allen_and_Kennedy(H, l)

- $H' = H \setminus \{e \mid l(e) < l\}$
- Build H'' the acyclic condensation of H', and number its vertices $\mathcal{V}_1, \ldots, \mathcal{V}_c$ in a topological sort order.
- **For** $i = 1$ **to** c **do**
  - i. **If** $\mathcal{V}_i$ is reduced to a single statement S, with no edge, **then** generate parallel DO loops (DOALL) in all remaining dimensions (i.e. for levels l to n) and generate code for S.
  - ii. Otherwise, let $k = l(\mathcal{V}_i)$. Generate parallel DO loops (DOALL) for levels from l to $k-1$, and a sequential DO loop (DOSEQ) for level k. Call Allen_and_Kennedy$(\mathcal{V}_i, k+1)$.

Finally, to apply Allen and Kennedy's algorithm to a reduced leveled dependence graph G, call: Allen_and_Kennedy(G,1).

Example 2 For illustrating Allen and Kennedy's algorithm, we consider the following example.

```
DO i = 1, N
  DO j = 1, N
    a(i, j) = i
    b(i, j) = b(i, j − 1) + a(i, j) ∗ c(i − 1, j)
    c(i, j) = 2 ∗ b(i, j) + a(i, j)
CONTINUE
```

The RLDG associated to the code of Example 2 is given in Figure 6. There are three statements

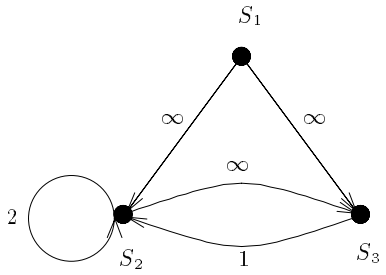


Figure 6: RLDG for Example 2

S_1, S_2 and S_3 in textual order. The first call is Allen_and_Kennedy(G,1) that detects two strongly connected components $\mathcal{V}_1 = \{S_1\}$ and $\mathcal{V}_2 = \{S_2, S_3\}$. The first component $\mathcal{V}_1$ has no edge (case (i)). Therefore, the algorithm generates:

```
DOPAR i = 1, N
  DOPAR j = 1, N
    a(i, j) = i
  ENDDO
ENDDO
```

The second component has level 1 edges, thus the algorithm generates:

DOSEQ $i = 1, N$
  ... (code for $\mathcal{V}_2$)
ENDDO

and recursively calls Allen_and_Kennedy($\mathcal{V}_2$, 2). Edges of level strictly less than 2 are removed. Two strongly connected components appear $\mathcal{V}_{2,1} = \{S_2\}$ and $\mathcal{V}_{2,2} = \{S_3\}$ in this order. The level of $\mathcal{V}_{2,1}$ is 2, therefore the algorithm generates:

DOSEQ $j = 1, N$
  ... (code for $\mathcal{V}_{2,1}$)
ENDDO

and recursively calls Allen_and_Kennedy($\mathcal{V}_{2,1}$, 3). The algorithm reaches step (i) and generates:

$\mathrm{b}(i, j) = \mathrm{b}(i, j-1) + \mathrm{a}(i, j) * \mathrm{c}(i-1, j)$

The second component $\mathcal{V}_{2,2} = \{S_3\}$ has no edge. Therefore, the generated code is:

DOPAR $j = 1, N$
  $\mathrm{c}(i, j) = 2 * \mathrm{b}(i, j) + \mathrm{a}(i, j)$
ENDDO

Finally, fusing all codes leads to:

DOPAR $i = 1, N$
  DOPAR $j = 1, N$
    $\mathrm{a}(i, j) = \mathrm{i}$
  ENDDO
ENDDO
DOSEQ $i = 1, N$
  DOSEQ $j = 1, N$
    $\mathrm{b}(i, j) = \mathrm{b}(i, j-1) + \mathrm{a}(i, j) * \mathrm{c}(i-1, j)$
  ENDDO
  DOPAR $j = 1, N$
    $\mathrm{c}(i, j) = 2 * \mathrm{b}(i, j) + \mathrm{a}(i, j)$
  ENDDO
ENDDO

We gave only, on purpose, the structure of DOSEQ and DOPAR. For a correct code, synchronization points should be added and minimized.

Definition 12 *Let S be a statement of G. We denote by $\tilde{d}_S$ the number of recursive calls needed in Algorithm Allen_and_Kennedy to generate the code for statement S (case (i) in above algorithm).*

In Example 2, three calls were needed for generating code for S_2, two calls for S_3 and one call for S_1. The S_1-degree of parallelism is 2, the S_2-degree of parallelism is 0 and the S_3-degree of parallelism is 1. Although the S_2-degree of intrinsic parallelism in the initial code of Example 2 is equal to 1, we will see that the S_2-degree of parallelism in the RLDG of Figure 6 is 0 as found by algorithm Allen_and_Kennedy.

We have the following lemma:

Lemma 2 *Let S be a statement of G. The number of sequential DO loops generated by Algorithm Allen_and_Kennedy, that surround statement S, is equal to $\tilde{d}_S - 1$.*

Proof Note that the chain of recursive calls ends as soon as case (i) is reached. The first call is Allen_and_Kennedy$(G, 1)$. Then, for $i \geq 2$, the i-th recursive call is Allen_and_Kennedy(H_i, k_i), for some strongly connected subgraph H_i of G, and some integer k_i such that $k_i = l(H_i) + 1$, $i - 1 \leq l(H_i) \leq n$ (this can be proved by induction on i). Therefore, the algorithm stops after at most $n+1$ recursive calls. Furthermore, each call generates exactly one sequential loop, except the last call that generates the code for S (and possibly some surrounding parallel loops). Therefore, the number of sequential DO loops generated for statement S is exactly $\tilde{d}_S - 1$. ■

4 Loop nest generation algorithm

In this section, we present a systematic procedure called *Loop_Nest_Generation*, that builds, from a reduced leveled dependence graph G, a perfect loop nest L' whose RLDG is exactly G. In Appendix A, we will prove that the S-degree of intrinsic parallelism in the EDG of L' is equal to the S-degree of parallelism extraction in G for Algorithm Allen_and_Kennedy, for all statements S of G, thereby proving the optimality of Allen and Kennedy's algorithm for dependence level abstraction.

The construction of L' is based on the notion of critical edges that are built in Section 4.1. The exact formulation of Procedure *Loop_Nest_Generation* is given in Section 4.2. Finally, in Section 4.3, we show that the RLDG associated to L' is G, as desired.

4.1 Critical edges

In this section, we build the data structure that we need for defining the apparent loops L'. The procedures given below (First_Call and Recursive_Calls) define:

- A set of so called *critical* edges E_c.
- An integer ξ.
- A set of cycles (denoted by $C(H)$ below).

Actually, only E_c is needed for building L', ξ and the cycles $C(H)$ will be used only in the main proof of Appendix A.

First_Call(G)

i. $\xi \leftarrow 0$.

ii. Build G'' the acyclic condensation of G, and number its vertices $\mathcal{V}_1, \ldots, \mathcal{V}_c$ in a topological sort order.

iii. **For** $i = 1$ **to** c **do**

- **If** $\mathcal{V}_i$ is reduced to a single vertex, with no edge, **then** do nothing.
- **Otherwise** call Recursive_Calls$(\mathcal{V}_i)$.

Recursive_Calls(H)

i. $l \leftarrow l(H)$.

ii. Select an edge f of H with level l. Call f the **critical** edge of H. $E_c \leftarrow E_c \cup \{f\}$.

iii. Build a cycle $C(H)$ that contains f, and that visits all vertices of H.
If length$(C(H)) > \xi$ **then** $\xi \leftarrow$ length$(C(H))$.

iv. $H' = H \setminus \{e \mid l(e) \leq l\}$.

v. Build H'' the acyclic condensation of H', and number its vertices $\mathcal{V}_1, \ldots, \mathcal{V}_c$ in a topological sort order.

vi. **For** $i = 1$ **to** c **do**

- **If** $\mathcal{V}_i$ is reduced to a single vertex, with no edge, **then** do nothing.
- **Otherwise** recursively call Recursive_Calls($\mathcal{V}_i$).

As for Algorithm Allen_and_Kennedy, we are interested in the number of recursive calls in Procedure Recursive_Calls.

Definition 13 *Let S be a statement of G. We denote by d_S the number of calls to Procedure Recursive_Calls that concern statement S, i.e. the number of calls Recursive_Calls(H) such that S is a vertex of H. d_S is called the **depth** of S.*

Algorithm Allen_and_Kennedy and the two procedures above have exactly the same structure of recursive calls: the first call Allen_and_Kennedy(G,1) is equivalent to the call to First_Call while subsequent calls to Algorithm Allen_and_Kennedy are equivalent to the calls to Recursive_Calls. Therefore $d_S = \tilde{d_S} - 1$ by definition of d_S and $\tilde{d_S}$.

Lemma 3 *The S-degree of parallelism extraction for Allen and Kennedy's algorithm is $n - d_S$ if n is the number of nested loops in the initial code.*

Proof $d_S = \tilde{d_S} - 1$. Furthermore, Lemma 2 proves that $\tilde{d_S} - 1$ is exactly the number of sequential DO loops, generated by Algorithm Allen_and_Kennedy, that surround S. Now, note that Algorithm Allen_and_Kennedy does not change the loop bound, but simply mark loops as parallel or sequential. Therefore, if $\mathcal{D}$ is the n-dimensional cube of size N, the iteration domain of the transformed code L_t is also the n-dimensional cube of size N for each statement S. Thus, L_t satisfies hypotheses of Lemma 1 and $n - d_S$ is nothing but the S-degree of parallelism extraction. ■

4.2 Generation of apparent nested loops

Let $G = (E, V)$ be a reduced leveled dependence graph. We assume that G has been built in a consistent way from some nested loops, i.e. we assume that G contains no cycle such that all edges have level ∞. Therefore, vertices can be numbered according to the topological order defined on them by the edges whose level is ∞: $v_i \xrightarrow{\infty} v_j \Rightarrow i < j$. Note that, if L is a set of loops whose RLDG is G, then the edges whose level is ∞ correspond to the so called *loop independent* dependences and the textual order (order in which statements appear in L) can be chosen for numbering the vertices.

We denote by d the **dimension** of G: $d = \max\{l(e) \mid e \in E \text{ and } l(e) < \infty\}$, i.e. the maximal level of edges of finite level. L' (the apparent loops of G) will consist of d perfectly nested loops, with $|V|$ statements, and each statement will be of the form $a_i[I] = \text{right\_member}(i)$. $a_1, \ldots, a_{|V|}$ are $|V|$ arrays of dimension d and right_member(i) is the right-hand side expressions for array a_i that will be built to capture the $|E|$ edges of G.

In the following, E_c is the set of critical edges of G (defined in Section 4.1) and "@" denotes the operator of expression concatenation.

Loop_Nest_Generation(G):

Initialization:

For $i = 1$ **to** $|V|$ **do** right_member$(i) \leftarrow$ "1"

First_Call(G)

Computation of the statements of L':

For $e = (v_i, v_j) \in E$ **do**

if $l(e) = \infty$ **then**

right_member(j)

$\leftarrow$ right_member(j) @ "$+a_i[I_1, \ldots, I_d]$"

if $l(e) < \infty$ and $e \notin E_c$ **then**

right_member(j)

$\leftarrow$ right_member(j) @ "$+a_i[I_1, \ldots, I_{l(e)-1}, I_{l(e)} - 1, I_{l(e)+1}, \ldots, I_d]$"

if $e \in E_c$ **then**

right_member(j)

$\leftarrow$ right_member(j) @ "$+a_i[I_1, \ldots, I_{l(e)-1}, I_{l(e)} - 1, \underbrace{N, \ldots, N}_{d-l(e)}]$"

Code generation for L':

For $i = 1$ **to** d **do**

generate ("**For** $I_i = 1$ **to** N **do**")

For $i = 1$ **to** $|V|$ **do**

generate ("$a_i[I_1, \ldots, I_d] :=$" @ right_member(i))

4.3 Reduced leveled dependence graph associated to L'

In this section, we show that the reduced leveled dependence graph for L' is exactly G.

Note that, as Procedure Recursive_Calls is always called on strongly connected graphs, the edges f and the cycles $C(H)$ can always be built.

Lemma 4 *The reduced leveled dependence graph of L' is G.*

Proof We denote by $T_1, \ldots, T_{|V|}$ the statements in L' and by G' the reduced leveled dependence graph associated to L'.

Note that, in L', there is only one *write* for each index vector $[I_1, \ldots, I_d]$ and each array a_i: this *write* occurs in statement T_i, at iteration $[I_1, \ldots, I_d]$: $a[I_1, \ldots, I_d] = \text{right\_member}(i)$. Therefore, the dependences in L' that involves array a_i correspond to a dependence between this unique *write* and some *read* on this array. Each *reads* on array a_i in the right-hand side of a statement corresponds, by construction of L', to one particular edge e in the graph G. Therefore, G and G'

have the same vertices and the same edges. One just has to check that the level of all edges is the same in G and G'.

Consider an edge e of G, $e = (v_i, v_j)$. Three cases can occur:

Case $l(e) = \infty$**:** remember that vertices of G have been numbered so that $i < j$ whenever $v_i \xrightarrow{\infty} v_j$. Therefore, since $l(e) = \infty$, statement T_i appears textually before statement T_j in L'. At iteration $[I_1, \ldots, I_d]$, statement T_i defines $a_i[I_1, \ldots, I_d]$. This value is used, in the same iteration, for defining $a_j[I_1, \ldots, I_d]$ in statement T_j. Therefore, there is an edge of level ∞ from statement T_i to statement T_j in G'.

Case $l(e) < \infty$ **and** $e \notin E_c$**:** at iteration $[I_1, \ldots, I_d]$, the definition, in statement T_j, of $a_j[I_1, \ldots, I_d]$ needs the value of $a_i[I_1, \ldots, I_{l(e)-1}, I_{l(e)} - 1, I_{l(e)+1}, \ldots, I_d]$. This creates a dependence from statement T_i to statement T_j with distance vector $[\underbrace{0, \ldots, 0}_{l(e)-1}, 1, \underbrace{0, \ldots, 0}_{d-l(e)}]$. Thus, the level of e in G' is l.

Case $e \in E_C$**:** in this case, the right member of statement T_j references value $a_i[I_1, \ldots, I_{l(e)-1}, I_{l(e)} - 1, \underbrace{N, \ldots, N}_{d-l(e)}]$. This reference creates a dependence from statement T_i to statement T_j of distance vector $[\underbrace{0, \ldots, 0}_{l(e)-1}, 1, I_{l(e)+1} - N, \ldots, I_d - N]$. Thus there is, in G', an edge of level l from T_i to T_j.

This proves that G and G' have same vertices, same edges, and that the labels of all edges are the same in G and in G'. In other words, $G = G'$. ■

Example 1 The reduced leveled dependence graph of Example 1 is drawn in Figure 2. This RLDG contains one edge, e, of level 1. Thus e is selected as *critical* by procedure Recursive_Calls. This critical edge generates a read *a(i − 1, N)*. When all the edges of level 1 are deleted from the RLDG, the new graph only contains a self-dependence e' of level 2. This self-dependence is also selected as *critical* and generates a read *a(i, j − 1)*.

Thus the apparent loops generated for Example 1 are, as promised in Section 2.2, the following:

```
DO i = 1, N
  DO j = 1, N
    a(i, j) = a(i, j − 1) + a(i − 1, N) + 1
CONTINUE
```

Example 2 The reduced leveled dependence graph of Example 2 is drawn in Figure 6. This RLDG contains one edge, e, of level 1, from S_3 to S_2. Thus e is selected as *critical* by procedure Recursive_Calls. This critical edge generates a read *c(i − 1, N)* in the right hand side of statement S_2. When all the edges of level 1 are deleted from the RLDG, the new graph still contains a self-dependence e' for S_2 of level 2. This self-dependence is also selected as *critical* and generates a read *b(i, j − 1)*. The graph obtained from the RLDG of Example 2 by deleting the edges of level 1 and 2 is acyclic. Thus all the edges which were not selected as critical generates simple accesses, i.e. accesses of the form *a(i, j)*.

Thus the apparent loops generated for Example 2 are:

```
DO i = 1, N
  DO j = 1, N
    a(i, j) = 1
    b(i, j) = 1 + b(i, j − 1) + a(i, j) + c(i − 1, N)
    c(i, j) = 1 + b(i, j) + a(i, j)
CONTINUE
```

for which the S_1-degree, S_2-degree, S_3-degree of intrinsic degree of parallelism are respectively 2, 0 and 0, as promised.

The goal of Appendix A is to prove the following result:

Theorem 1 *Let L be a set of loops whose RLDG is G. Use Algorithm Loop_Nest_Generation to generate the apparent loops L'. Let d_S be defined as in Definition 13. Then, for each strongly connected component G_i of G, there is a path in the EDG of the apparent loops L' which visits, $\Omega(N^{d_S})$ times, each statement S in G_i.*

The proof is long, technical and painful. It can be omitted at first reading. The important corollary is the following:

Corollary 1 *Allen and Kennedy's algorithm is optimal for parallelism extraction in reduced leveled dependence graphs (optimal in the sense of Definition 8).*

Proof Let G be a RLDG defined from n perfectly nested loops L. Lemma 3 proves that $n - d_S$ is the S-degree of parallelism extraction in G. Furthermore, Algorithm *Loop_Nest_Generation* generates a set of d perfectly nested loops L', whose RLDG is exactly G (Lemma 4) and such that, for each strongly connected component G_i of G, there is a path in the EDG associated to L', which visits, $\Omega(N^{d_S})$ times, each statement S in G_i (Theorem 1). If $d = n$, the corollary is proved: the S-degree of intrinsic parallelism in the EDG of L' is $n - d_S$.

It may be possible however that $d < n$. In this case, in order to define n apparent loops L'' instead of d apparent loops L', simply add $(n - d)$ innermost loops in L' and complete all array references with $[I_{d+1}, \ldots, I_n]$. This does not change the RLDG since, in L'', there is no dependence in the innermost loops, except loop independent dependences. Actually, the $(n - d)$ innermost loops are parallel loops and the path defined by Theorem 1 in the EDG of L' can be immediately translated into a path of same structure in the EDG of L'', simply by considering the $n - d$ last values of the iteration vectors as fixed (the EDG of L' is the projection of the EDG of L'' along the last $(n - d)$ dimensions). The result follows. ■

This proves that as long as the only information available is the RDG, it is not possible to detect more parallelism that found by Allen and Kennedy's algorithm. Is it possible to detect more parallelism if the structure of the code, i.e. the way loops are nested (but not the loop bounds), are given? The answer is no: it is possible to enforce L' to have the same nesting structure as L. The procedure is similar to Procedure *Loop_Nest_Generation*, but with the following modification:

- The left-hand side of statement S_i is $a_i(I)$ where I is the iteration vector corresponding to the loops that surround S_i in the initial code L. Thus, the dimension of the array associated to a statement is equal to the number of surrounding loops.
- The right-hand side of statement S_i is defined as in Procedure *Loop_Nest_Generation*, except that iteration vectors are completed by values equal to N if needed.

A theorem that generalizes Theorem 1 to the non perfectly nested case can be given, with a similar proof. We do not want to go into the details, the perfect case is painful enough. We just illustrate the non perfect case by the following example:

Example 3 Consider the following non perfectly nested loops (this code is the code called "petersen.t" given with the software Petit, see [KMP+95], obtained after scalar expansion).

```
DO i = 2, N
  s(i) = 0
  DO l = 1, i - 1
    s(i) = s(i) + a(l, i) * b(l)
  ENDDO
  b(i) = b(i) - s(i)
ENDDO
```

This code has exactly the same RDG as Example 2. Furthermore, it is well known that its S_1-degree, S_2-degree, S_3-degree of intrinsic degree of parallelism are respectively 1, 1 and 0. However, with Allen and Kennedy's algorithm, the S_1-degree, S_2-degree, S_3-degree of parallelism extraction are respectively 1, 0 and 0. This is because it is not possible, with only the RDG and the structure of the code, to distinguish between the above code and the following apparent one, for which the S_1-degree, S_2-degree, S_3-degree of intrinsic degree of parallelism are respectively 1, 0 and 0:

```
DO i = 1, N
  a(i) = 1
  DO j = 1, N
    b(i, j) = 1 + b(i, j - 1) + a(i) + c(i - 1)
  ENDDO
  c(i) = 1 + a(i) + b(i, N)
ENDDO
```

5 Conclusion

We have introduced a theoretical framework in which the optimality of algorithms that detect parallelism in nested loops can be discussed. We have formalized the notions of degree of parallelism extraction (with respect to a given dependence abstraction) and of degree of intrinsic parallelism (contained in a reduced dependence graph). This study permits to better understand the impact of a given dependence abstraction on the maximal parallelism that can be detected: it permits to determine whether the limitations of a parallelization algorithm are due to the algorithm itself or are due to the weaknesses of the dependence abstraction.

In this framework, we have studied more precisely the link between dependence abstractions and parallelism extraction in the particular case of *dependence level*. Our main result is the optimality of Allen and Kennedy's algorithm for parallelism extraction in reduced leveled dependence graphs. This means that even the most sophisticated algorithm cannot detect more parallelism, as long as dependence level is the only information available. In other words, loop distribution is sufficient for detecting maximal parallelism in dependence graphs with levels.

The proof is based on the following fact: given a set of loops L whose dependences are specified by level, we are able to systematically build a set of loops L' that cannot be distinguished from L (i.e. they have the same reduced dependence graph) and that have exactly the degree of parallelism found by Allen and Kennedy's algorithm. We call these loops the apparent loops. We believe this

construction of interest since it permits to better understand why some loops appear sequential when considering the reduced dependence graph while they actually may contain some parallelism.

References

[AK82] J.R. Allen and K. Kennedy. PFC: a program to convert programs to parallel form. Technical report, Dept. of Mathematical Sciences, Rice University, Houston, TX, March 1982.

[AK87] J.R. Allen and K. Kennedy. Automatic translations of Fortran programs to vector form. *ACM Toplas*, 9:491–542, 1987.

[Ban88] U. Banerjee. *Dependence Analysis for Supercomputing.* Kluwer Academic Publishers, Norwell, MA, 1988.

[Cal87] D. Callahan. *A Global Approach to Detection of Parallelism.* PhD thesis, Dept. of Computer Science, Rice University, Houston, TX, 1987.

[DV94] Alain Darte and Frédéric Vivien. Automatic parallelization based on multi-dimensional scheduling. Technical Report 94-24, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, France, September 1994.

[DV95] Alain Darte and Frédéric Vivien. A classification of nested loops parallelization algorithms. In *INRIA-IEEE Symposium on Emerging Technologies and Factory Automation*, pages 217–224. IEEE Computer Society Press, 1995.

[Fea91] Paul Feautrier. Dataflow analysis of array and scalar references. *Int. J. Parallel Programming*, 20(1):23–51, 1991.

[Fea92a] Paul Feautrier. Some efficient solutions to the affine scheduling problem, part I, one-dimensional time. *Int. J. Parallel Programming*, 21(5):313–348, October 1992.

[Fea92b] Paul Feautrier. Some efficient solutions to the affine scheduling problem, part II, multi-dimensional time. *Int. J. Parallel Programming*, 21(6):389–420, December 1992.

[GKT91] G. Goff, K. Kennedy, and C.W. Tseng. Practical dependence testing. In *Proceedings of ACM SIGPLAN'91 Conference on Programming Language Design and Implementation*, Toronto, Canada, June 1991.

[Gru90] D. Grunwald. Data dependence analysis: the λ test revisited. In *Proceedings of the 1990 International Conference on Parallel Processing*, 1990.

[IJT91] F. Irigoin, P. Jouvelot, and R. Triolet. Semantical interprocedural parallelization: an overview of the PIPS project. In *Proceedings of the 1991 ACM International Conference on Supercomputing*, Cologne, Germany, June 1991.

[IT87] F. Irigoin and R. Triolet. Computing dependence direction vectors and dependence cones with linear systems. Technical Report ENSMP-CAI-87-E94, Ecole des Mines de Paris, Fontainebleau (France), 1987.

[KKP90] X.Y. Kong, D. Klappholz, and K. Psarris. The I test: a new test for subscript data dependence. In Padua, editor, *Proceedings of 1990 International Conference of Parallel Processing*, August 1990.

[KMP+95] W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott. *New user interface for Petit and other interfaces: user guide.* University of Maryland, June 1995.

[KMW67] R.M. Karp, R.E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14(3):563–590, July 1967.

[Lam74] Leslie Lamport. The parallel execution of DO loops. *Communications of the ACM*, 17(2):83–93, February 1974.

[LYZ89] Z.Y. Li, P.-C. Yew, and C.Q. Zhu. Data dependence analysis on multi-dimensional array references. In *Proceedings of the 1989 ACM International Conference on Supercomputing*, pages 215–224, Crete, Greece, June 1989.

[Mol82] D.I. Moldovan. On the analysis and synthesis of vlsi systolic arrays. *IEEE Transactions on Computers*, 31:1121–1126, 1982.

[Mur71] Y. Muraoka. *Parallelism exposure and exploitation in programs.* PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, February 1971.

[PKK91] K. Psarris, X.Y. Kong, and D. Klappholz. Extending the I test to direction vectors. In *Proceedings of the 1991 ACM International Conference on Supercomputing*, Cologne, Germany, June 1991.

[Pug92] William Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:102–114, August 1992.

[Qui84] Patrice Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *The 11th Annual International Symposium on Computer Architecture*, Ann Arbor, Michigan, June 1984. IEEE Computer Society Press.

[Rao85] Sailesh K. Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays.* PhD thesis, Stanford University, October 1985.

[Roy88] Vwani P. Roychowdhury. *Derivation, Extensions and Parallel Implementation of Regular Iterative Algorithms.* PhD thesis, Stanford University, December 1988.

[WL91] Michael E. Wolf and Monica S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Trans. Parallel Distributed Systems*, 2(4):452–471, October 1991.

[Wol82] M. Wolfe. *Optimizing Supercompilers for Supercomputers.* PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, October 1982.

[Wol89] M. Wolfe. *Optimizing Supercompilers for Supercomputers.* MIT Press, Cambridge MA, 1989.

[YAI95] Y.-Q. Yang, C. Ancourt, and F. Irigoin. Minimal data dependence abstractions for loop transformations (extended version). *International Journal of Parallel Programming*, 23(4):359–388, August 1995.

[Yan93] Yi-Qing Yang. *Tests des dépendances et transformations de programme.* PhD thesis, Ecole Nationale Supérieure des Mines de Paris, Fontainebleau, France, 1993.

[ZC90] Hans Zima and Barbara Chapman. *Supercompilers for Parallel and Vector Computers.* ACM Press, 1990.

A Appendix: proof of optimality

In this section, we denote by L the initial loops and by G the reduced leveled dependence graph associated to L. We denote by L' the "apparent loops" generated by Procedure *Loop_Nest_Generation* applied to G and we denote by d_S (see Definition 13) the number of sequential loops detected by Allen and Kennedy's algorithm, that surround statement S, when processing G.

We show that for each statement S in L', there is a dependence path in the expanded dependence graph of L' that contains $\Omega(N^{d_S})$ instances of the statement S. More precisely, we build a dependence path that satisfies this property simultaneously for all statements of a strongly connected component of G. This path is built by induction on the depth (to be defined) of a strongly connected component of G. In Section A.3, we study the initialization of the induction, whose general case is studied in Section A.4. The proof by induction itself is presented in Section A.5. Finally, in Section A.6, we establish the optimality theorem.

To make the proof clearer, we give in Section A.2 a schematic description of the induction. Before that, we need to introduce some new definitions, which is done in Section A.1.

A.1 Some more definitions

We extend the notion of **depth** to graphs. Remember that we defined the depth d_S of a statement S in Definition 13 as the number of calls to Procedure Recursive_Calls that concern statement S when processing the graph G.

Let H be a subgraph of G that contains S: we define similarly $d_S(H)$ as the number of calls to Procedure Recursive_Calls that concern statement S when processing the graph H (instead of G). Note that $d_S = d_S(G)$. Finally, we define $d(H)$, the depth of H, as the maximal depth of the vertices (i.e. statements) of H:

Definition 14 (Graph depth) *The depth of a reduced dependence graph H is:*

$$d(H) = \max\{d_v(H) \mid v \in V(H)\}$$

where $V(H)$ denotes the vertices of H.

The proof of the theorem is based on an induction on the depths of the strongly connected components of G that are built in Procedure First_call and Procedure Recursive_Calls.

Given a path P in a graph, $P = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{k-1}} x_k$, we define the tail of P as the first statement visited by P, denoted by $t(P) = x_1$, and we define the head of P as the last statement visited by P, denoted by $h(P) = x_k$. We define the **weight** of a path as follows:

Definition 15 (Path weight)
*Let P be a path of a reduced leveled dependence graph G, for which $l(e)$ denotes the level of an edge e. We define the **weight** of P at level l, denoted $w_l(P)$, as the number of edges in P whose level is equal to l:*

$$w_l(P) = |\{e \in P \mid l(e) = l\}|$$

A.2 Induction proof overview

In the following, H denotes a subgraph of G which appears in the decomposition of G by Procedure *First_Call*. The induction is an induction on the depth of H.

We want to prove by induction that if S is a statement of H, there is in the expanded dependence graph of L' a dependence path whose S-length is $\Omega(N^{d_S(H)-1})$. From this result, we will be able to build the desired path.

First of all, we prove in Theorem 3 (Section A.3) that if $d(H) = 1$, there exists in the expanded dependence graph of L' a path that visits all statements of H, whose tail and head correspond to the same statement, for two different iterations, and the starting iteration vector can be fixed to any value in a certain sub-space of the iteration space. We say this path is a cycle, as it corresponds to a cycle in the reduced dependence graph.

Then, we prove that whatever the depth of H, this property still holds: there exists in the expanded dependence graph of L' a path which visits all statements of H, whose tail and head correspond to the same statement, and the starting iteration vector can be fixed to any value in a certain sub-space of the iteration space. Furthermore, we can connect on this path the different paths built for the subgraphs $H_1, \ldots, H_c$ of H which appear at the first step of the decomposition of H by Procedure *Recursive_Calls*. Each of these cycles can be connected a number of times linear in N, the domain size parameter. This leads to the desired property, which is proved in Theorem 3 (Section A.4).

Remark that the subgraphs $H_1, \ldots, H_c$ have depths strictly smaller than the depth of H, this is why the induction is made on the depths of the graphs. Furthermore, all subgraphs H_i are strongly connected by construction. As a consequence, their level is an integer, i.e $l(H_i) \neq \infty$.

A.3 Initialization of the induction: $d(H) = 1$

The initial case of the induction is divided in three parts: in the first one, Section A.3.1, we recall what we built in Section 4; in the second one, Section A.3.2, we prove some intermediate results; in the third one, Section A.3.3, we build the desired path from the results that we previously established.

A.3.1 Data, hypotheses and notations

In this subsection, we recall or define the data, hypotheses and notations which will be valid throughout Section A.3. In particular, Property 1, Lemma 5 and Theorem 2 are proved under the hypotheses listed below.

- H is a subgraph of G which appears in the decomposition of G obtained by Procedure *First_Call*. We suppose that H is of depth one, $d(H) = 1$.
- l is the level of H: $l = l(H)$.
- f is the critical edge of H.
- $C(H)$ is the cycle containing f which visits all statements in H. $C(H)$ is defined by Procedure *Recursive_Calls* during the processing of H.
- The cycle $C(H)$ can be split up into $C(H) = f, P_1, f, P_2, f, \ldots, f, P_{k-1}, f, P_k$, where, for all i, $1 \leq i \leq k$, the path P_i does not contain the edge f. Remark that, for all i, $1 \leq i \leq k$, the first statement of P_i (i.e. $t(P_i)$) is the head of edge f (i.e. $h(f)$), and the last statement of P_i (i.e. $h(P_i)$) is the tail of edge f (i.e. $t(f)$), as $C(H)$ is a cycle. This decomposition is shown of figure 7.

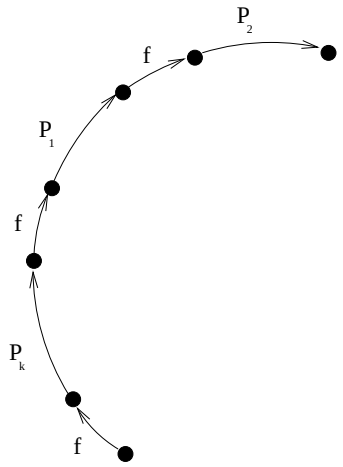


Figure 7: The decomposition of $C(H)$

- ξ is the integer of the same name computed by Procedure *First_Call*. ξ is equal to the maximal length of all cycles $C(H)$ built during the process of G.
- All iteration vectors are of size d. We will use the following notation: if I is a vector of size $l'-1$, for $1 \leq l' \leq d$, and if j is an integer, $(I, j, N, \ldots, N)$ denotes the iteration vector $(I, j, \underbrace{N, \ldots, N}_{d-l'})$.

A.3.2 A few bricks for the wall (part I)

We first prove in Property 1 that there is no critical edge in a path P_i. Then we use this property to build in Lemma 5 a path in the expanded dependence graph of L' which corresponds to $f + P_i$. This last result will be used in Section A.3.3 in order to prove Theorem 3.

Property 1 *Let i be an integer such that $1 \leq i \leq k$. Then, P_i contains no critical edge.*

Proof: See Procedure *Recursive_Calls*: an edge can be selected as a critical edge only at the level it is removed from the graph being processed. As H is supposed to be of depth 1, all edges are removed from H at the same level, and H contains only one critical edge, f, its critical edge. Thus, as f is not part of P_i by construction, P_i contains no critical edge. ■

The previous property established, we are able to build in the expanded dependence graph of L' a path whose tail and head correspond in G to the same statement, and whose projection in the reduced level dependence graph is exactly the path $f + P_i$. The existence of such a path is conditioned by the value of the iteration vector associated to the starting statement.

Lemma 5 *Let i be such that $1 \leq i \leq k$. Let I be a vector in $[1 \ldots N]^{l-1}$, let j be an integer such that $1 \leq j \leq N - (1 + w_l(P_i))$, and let K be a vector in $[\xi \ldots N]^{d-l}$.*

Then, there exists, in the expanded dependence graph of L', a dependence path that corresponds in H to the use of f followed by all edges of P_i, from iteration $(I, j, N, \ldots, N)$ of statement $t(f)$ to iteration $(I, j + 1 + w_l(P_i), K)$ of the same statement.

Proof: Let K' be the $(d-l)$-dimensional vector defined by $K'_r = K_r - w_{l+r}(P_i)$ for $1 \leq r \leq d-l$. Remember that P_i is a sub-path of $C(H)$ not equal to $C(H)$ (at least f is not in P_i), and the length (in terms of number of edges) of $C(H)$ is less than or equal to ξ. Thus, for all r, $1 \leq r \leq d-l$, $w_{l+r}(P_i) \leq \xi - 1$. Since $K \in [\xi \ldots N]^{d-l}$, $K' \in [1 \ldots N]^{d-l}$ and $(I, j+1, K')$ belongs to the iteration domain $[1 \ldots N]^d$.

According to Algorithm *Loop_Nest_Generation* and according to the definition of edge f, instance $(I, j+1, K')$ of statement $h(f)$ depends on instance $(I, j, N, \ldots, N)$ of statement $t(f)$.

From Property 1, we know that none of the edges of P_i are critical edges. Thus, according to Algorithm *Loop_Nest_Generation*, for each edge $e : S_1 \xrightarrow{e} S_2$ of P_i, and for every vector I in $[1 \ldots N]^d$, we have:

- If $l(e) = \infty$, then instance $(I_1, \ldots, I_d)$ of statement S_2 depends on instance $(I_1, \ldots, I_d)$ of statement S_1.
- If $l(e) < \infty$, then instance $(I_1, \ldots, I_{l(e)-1}, I_{l(e)} + 1, I_{l(e)+1}, \ldots, I_d)$ of statement S_2 depends on instance $(I_1, \ldots, I_{l(e)-1}, I_{l(e)}, I_{l(e)+1}, \ldots, I_d)$ of statement S_1, if, and only if, $I_{l(e)} + 1 \leq N$, i.e. if $I + w(e) \leq N$.

Furthermore, by definition of l, all edges of P_i have a level greater than or equal to l: the $l-1$ first components of $w(P_i)$ are null. Thus, we can build a dependence path in the expanded dependence graph of L', that corresponds to all edges of P_i, from instance $(I, j+1, K')$ of statement $h(f)$, to instance $(I, j+1+w_l(P_i), K'_1 + w_{l+1}(P_i), \ldots, K'_{d-l} + w_d(P_i))$ of statement $t(f)$, i.e. instance $(I, j+1+w_l(P_i), K)$.

Finally, since instance $(I, j+1, K')$ of statement $h(f)$ depends on instance $(I, j, N, \ldots, N)$ of statement $t(f)$, we get the desired path from iteration $(I, j, N, \ldots, N)$ of statement $t(f)$ to iteration $(I, j+1+w_l(P_i), K)$ of the same statement. ■

A.3.3 Conclusion for the initialization case

In the next corollary (Corollary 2), we simply concatenate the paths built in Lemma 5 for the different paths P_i. This permits to build a path in the expanded dependence graph of L' whose projection in the reduced leveled dependence graph is $C(H)$. This form the path announced in the proof overview (Section A.2).

Corollary 2 *Let I be a vector in $[1 \ldots N]^{l-1}$, let j be an integer such that $1 \leq j \leq N - w_l(C(H))$, and let K be a vector in $[\xi, \ldots, N]^{d-l}$.*

Then, there exists, in the expanded dependence graph of L', a dependence path Φ from instance $(I, j, N, \ldots, N)$ of statement $t(f)$, to instance $(I, j + w_l(C(H)), K)$ of the same statement, and which visits all nodes of H.

Proof: Let $j_1 = j$ and $j_i = j + \sum_{m=1}^{i-1}(1 + w_l(P_m))$ for $2 \leq i \leq k$. Then, for each value i from 1 to $k-1$, apply Lemma 5 for P_i, with $K = (N, \ldots, N)$ and with the value of j_i defined above. This defines $k-1$ paths that can be concatenated to form a dependence path from instance $(I, j, N, \ldots, N)$ of statement $t(f)$ to instance $(I, j + \sum_{m=1}^{k-1}(1 + w_l(P_m)), N, \ldots, N)$ of the same statement.

Then, applying once again Lemma 5 for P_k in its general setting, we get a dependence path from instance $(I, j, N, \ldots, N)$ of statement $t(f)$ to instance $(I, j + \sum_{m=1}^{k}(1 + w_l(P_m)), N, \ldots, N)$ of the same statement, i.e. instance $(I, j + w_l(C(H)), K)$.

The projection in the reduced leveled dependence graph of L' of the path we just built is the concatenation of the paths $f + P_i$, i.e. $C(H)$. As $C(H)$ visits all statements in H, this path visits all statements in H. ■

Actually, the theorem that we need for the induction proof is a corollary of Corollary 2 which we prove here (Corollary 3). We could use Corollary 2 rather than Corollary 3, but we prefer this last formulation for the sake of regularity as it gives for the subgraphs H of depth 1 exactly the result that will be proved on subgraphs of greater depths (Theorem 3).

Corollary 3 *Let I be a vector in $[1 \dots N]^{l-1}$, let j be an integer such that $1 \leq j \leq N - |V| w_l(C(H))$, and let K be a vector in $[\xi, \dots, N]^{d-l}$.*

Then, there exists in the expanded dependence graph of L' a dependence path Φ which visits all nodes of H, which starts at instance $(I, j, N, \dots, N)$ of statement $t(f)$, and which ends at instance $(I, j + |V| w_l(C(H)), K)$ of the same statement.

Proof: This path is build by the concatenation of $|V|$ paths built by Corollary 2, exactly as the path built for Corollary 2 has been built by the concatenation of k paths built by Lemma 5. ■

Now that the desired result for the initialization case of the induction is established, we study the general case.

A.4 General case of the induction

We first formulate formally the induction hypothesis in Section A.4.1. In Section A.4.2, we define what we are going to work on. In Section A.4.3, we prove some intermediate results. Finally, in Section A.4.4, we build the desired path from the previously established results.

A.4.1 Induction hypothesis

We present in this section the induction hypothesis from which the induction proof will be established in Section A.5. The fact that the induction hypothesis requires quite complicated conditions is mainly technical and will become clear all along Section A.4.

The induction hypothesis $\mathcal{IH}(k)$ is parameterized by the depth k of the subgraph H. Schematically, $\mathcal{IH}(k)$ is true if, for all subgraphs H that appear during the processing of G, there exists a dependence path in the expanded dependence graph of L' which visits each statement S of H $\Omega(N^{d_S(H)-1})$ times.

Definition 16 (Induction hypothesis at depth k: $\mathcal{IH}(k)$)
*We suppose that N is greater than $(d+1)|V|\xi$. The **induction hypothesis** $\mathcal{IH}(k)$ is true, if and only if, for all subgraphs H of G such that:*

- *H is strongly connected with at least one edge (H is a subgraph of G that appears during the processing of G),*
- *$d(H) \leq k$,*

there exists a path Φ, in the expanded dependence graph of L', from instance $(I, j, N, \dots, N)$ of statement $t(C(H))$ to instance $(I, j + |V| w_{l(H)}(C(H)), K)$ of the same statement, $\forall\ I \in [1 \dots N]^{l(H)-1}$, $\forall\ j,\ 1 \leq j \leq N - |V| w_{l(H)}(C(H))$, $\forall\ K \in [N - (d + 1 - d(H))|V|\xi \dots N]^{d-l(H)}$ and such that for each statement S in H the S-length of Φ is $\Omega(N^{d_S(H)-1})$ times.

Before going further, we give here some remarks on the importance of the different hypotheses, conditions and values on the components of the index vector of the starting and ending statements of the built paths:

- The $l(H)-1$ first components of the index vectors are constant along this dependence path. This simplify our work when we connect this cycle with a cycle built for the subgraph of smaller depth: we will just have to look at the $d-l(H)+1$ last components.
- The $l(H)$-th component is increased by a constant factor, namely $|V|w_{l(H)}(C(H))$. In particular this factor is independent of N. Joined to the freedom we have for the value of the $d-l$ last components of the ending statement index vector (variable K), this will allow us to connect consecutively $\Omega(N)$ times the cycles built for smaller depths.

The subgraphs of depth 1 do satisfy this induction hypothesis:

Theorem 2 *$\mathcal{IH}(1)$ is true.*

Proof: This theorem is a simple corollary of Corollary 3 since in this case $N^{d_v(H)-1}=1$, and $N-(d+1-d(H))|V|\xi \geq d(H)|V|\xi \geq \xi$. ■

A.4.2 Data, hypotheses and notations

We recall or define in this subsection the data, hypotheses and notations which will be valid all along Section A.4. In particular, Property 2, Corollary 4, Lemmas 6 and 7, and Theorem 3 are proved under the hypotheses listed here.

- H is a subgraph of G, strongly connected, with at least one edge, that appears during the processing of G.
- l is the level of H: $l=l(H)$.
- We assume that $\mathcal{IH}(k)$ is true for $k<l$: the goal of this section is to show that $\mathcal{IH}(l)$ is true.
- $H_1,\ldots,H_c$ are the c subgraphs of H on which Procedure *Recursive_Calls* is recursively called. In particular, H_i satisfies $\mathcal{IH}(d(H_i))$ since each H_i is strongly connected, with at least one edge and has a depth strictly smaller than the depth of H.
- For i, $1\leq i\leq c$, we denote by Φ_i the dependence path defined for H_i by the induction hypothesis $\mathcal{IH}(d(H_i))$.
- ξ is the integer of the same name computed on G by Procedure *First_Call*.
- f is the **critical** edge of H.
- $C(H)$ is the cycle which contains f and visits all statements in H and which is defined by Procedure *Recursive_Calls* during the processing of H.
- The cycle $C(H)$ can be decomposed into: $C(H)=f,P'_1,f,P'_2,f,\ldots,f,P'_{k-1},f,P'_k$, where, for i, $1\leq i\leq k$, the path P'_i does not contain the edge f. Remark that, since $C(H)$ is a cycle, the first statement in the path P'_i (i.e. $t(P'_i)$) is the head of the edge f (i.e. $h(f)$), and that the last statement in P'_i (i.e. $h(P'_i)$) is the tail of edge f (i.e. $t(f)$).

- C' is the concatenation of $|V|$ times the cycle $C(H)$. C' is naturally decomposed into: $C' = f, P_1, f, P_2, f, \ldots, f, P_{|V|k-1}, f, P_{|V|k}$, where $P_i = P'_{(i \bmod k)}$. Introducing C' permits to simplify the proof that is more technical than difficult.

- If P is a path, and p and q are two vertices of P, then $p \xrightarrow{P} q$ denotes the sub-path of P which starts at vertex p and which ends at vertex q.

- If p is a vertex of a path P of H, we denote by $\delta(P, p, l')$ the first vertex of $p \xrightarrow{P} h(P)$ followed by a critical edge whose level is strictly less than l'.

- As the number of strongly connected components (which is at least c) in H (subgraph of G) is smaller than the number of vertices ($|V|$) in G, we have $c \leq |V|$. As C' contains $|V|$ occurrences of each path P'_i, for $1 \leq i \leq k$, we can define $s_1, \ldots, s_c$ as occurrences of $t(C(H_1)), \ldots, t(C(H_c))$ in C' such that each path P_i (for $1 \leq i \leq |V|k$) contains at most one of the s_m ($1 \leq m \leq |V|$).

A.4.3 A few bricks for the wall (part II)

We prove in Property 2 that a path P of H which does not contain the critical edge of H, usually denoted f, cannot contain any critical edge of level less than or equal to l.

Property 2 *Let P be a path of H which does not contain the edge f. The critical edges contained in P are of levels greater than or equal to $l+1$.*

Proof: An edge can only be selected as critical at the level where it is removed from the calling graphs. Only one edge, f, is selected as critical at level l for H. But f is not part of P by hypothesis. All other edges of H of level l are deleted from H and H contains no edge at level less than l by definition of l ($l = l(H)$). Thus, the critical edges contained in P are of levels greater than or equal to $l+1$. ■

By construction of the paths P_i, $1 \leq i \leq k|V|$, P_i does not contain the edge f. We can thus apply Property 2 to these paths:

Corollary 4 *Let i be such that $1 \leq i \leq k|V|$. The critical edges contained in P_i are of levels greater than or equal to $l+1$.*

In the following lemma, Lemma 6, we prove that for any path P of H which does not contain the edge f and whose length is smaller than ξ, we can build in the expanded dependence graph of L' a path whose projection on H is P. Moreover, we express exactly the instances of the statements visited by P.

This lemma will be applied to sub-paths of the paths P_i in Lemma 7.

Lemma 6 *Let P be a path of H which does not contain the edge f. Let us suppose that the length of P is strictly less than ξ. Let I be a vector in $[1 \ldots N]^{l-1}$, let j be such that $1 \leq j \leq N$, and let K be a vector in $[\xi \ldots N]^{d-l}$.*

Then, there exists a dependence path $\mathcal{P}$ in the expanded dependence graph of L' whose projection onto H is equal to P, and which ends at instance $(I, j + w_l(P), K)$ if $1 \leq j + w_l(P) \leq N$. Furthermore, the path $\mathcal{P}$ visits the statement that corresponds to a vertex p of P at iteration $\left(I, j + w_l(P) - w_l\left(p \xrightarrow{P} h(P)\right), K'\right)$ where, for $l+1 \leq l' \leq d$, $K'_{l'-l} = N - w_{l'}\left(p \xrightarrow{P} \delta(P, p, l')\right)$ if $\delta(P, p, l')$ exists and $K'_{l'-l} = K_{l'-l} - w_{l'}\left(p \xrightarrow{P} h(P)\right)$ otherwise.

Proof: The result is complicated, but the proof is not: it is done by induction and backwards: from the last vertex of P down to the first one.

The formula is obviously true if P is reduced to a null-length path (i.e. with no edge).

Let us suppose that the property has been proved until a vertex q. Then, there is a path $\mathcal{P}(q)$ from an instance of q to instance $(I, j + w(P), K)$ of $h(P)$, which visits, for all vertex r between q and $h(P)$, the statement r at instance $(I, j + w_l(P) - w_l\left(r \xrightarrow{P} h(P)\right), K')$ where, for $l+1 \leq l' \leq d$, $K'_{l'-l} = N - w_{l'}\left(r \xrightarrow{P} \delta(P, r, l')\right)$ if $\delta(P, r, l')$ exists and $K'_{l'-l} = K_{l'-l} - w_{l'}\left(r \xrightarrow{P} h(P)\right)$ otherwise.

Let us consider the edge e of P whose head is q ($p \xrightarrow{e} q$). We have three cases to consider:

- $l(e) = \infty$. Thus, e is not critical, and, for any l', $l+1 \leq l' \leq d$, $\delta(P, p, l')$ exists if, and only if, $\delta(P, q, l')$ exists, and then $\delta(P, p, l') = \delta(P, q, l')$. Furthermore, the edge e has a null weight, which means that the instance where statement p is visited, (I, j', K''), is equal to the instance where statement q is visited.

  Thus, $j' = j + w_l(P) - w_l\left(q \xrightarrow{P} h(P)\right) = j + w_l(P) - w_l\left(p \xrightarrow{P} h(P)\right)$, and for any l', $l+1 \leq l' \leq d$, $K''_{l'-l} = N - w_{l'}\left(q \xrightarrow{P} \delta(P, q, l')\right) = N - w_{l'}\left(p \xrightarrow{P} \delta(P, p, l')\right)$ if $\delta(P, q, l') = \delta(P, p, l')$ exists and $K''_{l'-l} = K_{l'-l} - w_{l'}\left(q \xrightarrow{P} h(P)\right) = K_{l'-l} - w_{l'}\left(p \xrightarrow{P} h(P)\right)$ otherwise.

- $l(e) < \infty$ and $e \notin E_c$.

  e is not critical. Thus, for any l', $l+1 \leq l' \leq d$, $\delta(P, p, l')$ exists if, and only if, $\delta(P, q, l')$ exists, and then $\delta(P, p, l') = \delta(P, q, l')$.

  $w(e)$ is null except for the $l(e)$-th component which is equal to 1. For all null components of $w(e)$, we use the same arguments as for the case $l(e) = \infty$ to conclude. Thus, we just have to look at the $l(e)$-th component.

  $l(e)$ is greater than or equal to l. We denote by (I, j', K'') the instance where the statement p is visited.

  - If $l(e) = l$, then $j' = \left(j + w(P) - w_l\left(q \xrightarrow{P} h(P)\right)\right) - w_l(e)$
    $= j + w(P) - w_l\left(e + q \xrightarrow{P} h(P)\right)$
    $= j + w(P) - w_l\left(p \xrightarrow{P} h(P)\right)$
  - If $l(e) \neq l$ (i.e. $l(e) > l$) and if $\delta(P, q, l(e))$ exists, then:
    $K''_{l(e)-l} = \left(N - w_{l(e)}\left(q \xrightarrow{P} \delta(P, q, l(e))\right)\right) - w_{l(e)}(e)$
    $= N - w_{l(e)}\left(e + q \xrightarrow{P} \delta(P, p, l(e))\right)$
    $= N - w_{l(e)}\left(p \xrightarrow{P} \delta(P, p, l(e))\right)$
  - If $l(e) \neq l$ (i.e. $l(e) > l$) and if $\delta(P, q, l(e))$ does not exist, then:
    $K''_{l(e)-l} = K_{l(e)-l} - w_{l(e)}\left(q \xrightarrow{P} h(P)\right) - w_{l(e)}(e)$
    $= K_{l(e)-l} - w_{l(e)}\left(e + q \xrightarrow{P} h(P)\right)$
    $= K_{l(e)-l} - w_{l(e)}\left(p \xrightarrow{P} h(P)\right)$

- $e \in E_c$. Then $l(e) > l$ because of Property 2.

In this case, instance $(J_1, \ldots, J_{l(e)-1}, J_{l(e)}, J_{l(e)+1}, \ldots, J_d)$ of statement q depends on instance $(J_1, \ldots, J_{l(e)-1}, J_{l(e)} - 1, N, \ldots, N)$ of statement p, for any vector J in $[1 \ldots N]^d$ such that $J_{l(e)} - 1 \geq 1$.

Furthermore, for any l', $l+1 \leq l' \leq l(e)$, $\delta(P, p, l')$ exists if and only if, $\delta(P, q, l')$ exists, and then $\delta(P, p, l') = \delta(P, q, l')$, and for any l', $l(e)+1 \leq l' \leq d$, $\delta(P, p, l') = p$.

We just have to look to the components between $l(e)$ and d, as for the other components, the weight of e is null, δ is constant, and thus the same arguments as for the case $l(e) = \infty$ can be used to conclude.

We denote by (I, j', K'') the instance where statement p is visited.

- $l(e)$-th component (i.e. j'). Since $w_{l(e)}(e) = 1$, we conclude for this component with the same arguments as for the previous case ($l(e) < \infty$ and $e \notin E_c$).
- l'-th component, with $l' > l(e)$. Then, $\delta(P, p, l') = p$ and $K''_{l'} = N$. The desired formula remains true since $K''_{l'} = N = N - w_{l'}\left(p \xrightarrow{P} p\right) = N - w_{l'}\left(p \xrightarrow{P} \delta(P, p, l')\right)$.

We still have to check that the instance where p is visited belongs to $[1 \ldots N]^d$. This is obvious when looking at the instance formula we just proved:

- $j + w_l(P) - w_l\left(p \xrightarrow{P} h(P)\right) \geq 1$ since $p \xrightarrow{P} h(P)$ is a sub-path of P and $j \geq 1$.
- The components of K'' are the components of K minus the weight of a sub-path of P, which is strictly less than ξ. Since all components of K are larger than ξ, all components of K'' are larger than 1.

■

We now apply Lemma 6 to the paths P_i, so as to build a path corresponding to $f + P_i$ in the expanded dependence graph of L'. This is done in Lemma 7. There are two main differences between Lemmas 5 and 7. The first one is the existence in the paths P_i of critical edges which obliged us to previously establish Lemma 6. The second one corresponds to the cycles Φ_m: because of the desired property on the number of occurrences of each statement in the built path, if P_i contains the vertex s_m, we have to turn $\Omega(N)$ times round Φ_m while building the path corresponding to P_i. This construction is illustrated on Figure 8. Formally:

Lemma 7 *Let i be such that $1 \leq i \leq |V|k$. Let I be a vector in $[1 \ldots N]^{l-1}$, let j be such that $1 \leq j \leq N - (1 + w_l(P_i))$, and let K be a vector in $[N - (d + 1 - d(H))|V|\xi \ldots N]^{d-l}$.*

Then, there exists in the expanded dependence graph of L' a dependence path $\mathcal{P}_i$ which goes along f and all edges of P_i, which starts at the instance $(I, j, N, \ldots, N)$ of statement $t(f)$ and which ends at instance $(I, j + 1 + w_l(P_i), K)$ of the same statement.

Furthermore, if s_m is a vertex of P_i, for some m, $1 \leq m \leq c$, then this dependence path visits, $\Omega(N^{d_S(H)-1})$ times, each statement S of H_m.

Proof: Consider a vector I, an integer j and a vector K according to the theorem hypotheses.

Two cases can occur, depending if one of the vertices of P_i is s_m for some m, $1 \leq m \leq c$. First of all, note that:

i. By definition, P_i does not contain the edge f.

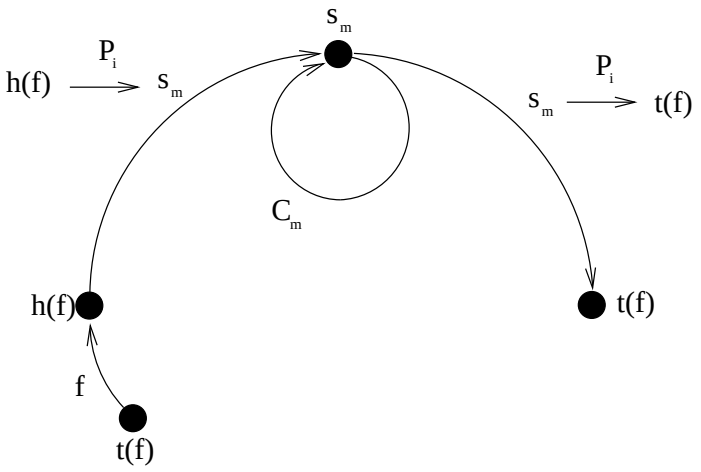


Figure 8: As P_i contains the vertex s_m, we turn round Φ_m (denoted here C_m)

ii. P_i is strictly shorter than $C(H)$ whose length is smaller than ξ. Thus the length of P_i is strictly less than ξ.

iii. By hypothesis, $N \geq (d+1)|V|\xi \geq (d+2-d(H))|V|\xi$ (since $d(H) \geq 1$). Therefore, the $(d-l)$-dimensional set $[N-(d+1-d(H))|V|\xi \ldots N]^{d-l}$ is a subset of $[|V|\xi \ldots N]^{d-l}$, and thus of $[\xi \ldots N]^{d-l}$.

These two properties permit to apply Lemma 6 to sub-paths of P_i in the two following cases:

- We suppose that no vertex s_m belongs to P_i.

  Because of the preceding remarks, we can apply Lemma 6 to P_i. This gives a path from instance $(I, j+1, K')$ of statement $t(P_i) = h(f)$ to instance $(I, j+1+w_l(P_i), K)$ of statement $h(P_i) = t(f)$, where K' is defined by Lemma 6. Then we concatenate, in front of this path, the edge corresponding to f from instance $(I, j, N, \ldots, N)$ of $t(f)$ to instance $(I, j+1, K')$ of statement $h(f)$. This leads to the desired result.

- We now suppose that s_m is a vertex of P_i for some m, $1 \leq m \leq c$. By construction, if $m' \neq m$, $s_{m'}$ is not a vertex of P_i (see Section A.4.2). We build the path backwards, starting from the ending vertex of P_i.

  The previous remarks allow us to apply Lemma 6 to the sub-path of P_i: $s_m \xrightarrow{P_i} h(P_i)$. This gives a path from instance $\left(I, j+1+w_l(P_i) - w_l\left(s_m \xrightarrow{P_i} h(P_i)\right), K'\right)$ of statement s_m to instance $(I, j+1+w_l(P_i), K)$ of statement $h(P_i) = t(f)$, where K' is defined by Lemma 6. Remark that $w_l(P_i) - w_l\left(s_m \xrightarrow{P_i} h(P_i)\right) = w_l\left(t(P_i) \xrightarrow{P_i} s_m\right)$. We call this first path $\mathcal{P}_1$.

  We want, starting from this instance of s_m, to turn (backwards) $\Omega(N)$ times round the cycle Φ_m. To do so, we use the induction hypothesis $\mathcal{IH}(d(H_m))$ for H_m. We have first to check that the path described in $\mathcal{IH}(d(H_m))$ can be inserted in front of instance $(I, j+1+w_l\left(t(P_i) \xrightarrow{P_i} s_m\right), K')$ of s_m. In other words, we have to check that the $d - l(H_m) + 1$ last components of $(I, j+1+w_l\left(t(P_i) \xrightarrow{P_i} s_m\right), K')$ satisfy the hypothesis stated in $\mathcal{IH}(d(H_m))$.

  Since, $l(H_m) > l$, the components that have to be considered are the components of K'. We thus have to check that:

- $K'_{l'-l} > N - (d+1-d(H_m))|V|\xi$ for $l' > l(H_m)$.
- $1 + |V| w_{l(H_m)}(C(H_m)) \leq K'_{l(H_m)-l} \leq N$.

For that, consider the $l'-l$-th component of K': once again, we have two cases to consider:

- $\delta(P_i, s_m, l')$ exists. Then, $K'_{l'-l} = N - w_{l'}\left(s_m \xrightarrow{P_i} \delta(P_i, s_m, l')\right)$. $s_m \xrightarrow{P_i} \delta(P_i, s_m, l')$ is a sub-path of P_i whose length is strictly less than ξ. Thus, $w_{l'}\left(s_m \xrightarrow{P_i} \delta(P_i, s_m, l')\right) < \xi$, and $K'_{l'-l} > N - \xi$. Now, since $d+1-d(H_m) \geq 1$, $N - \xi \geq N - (d+1-d(H_m))|V|\xi$. This proves $K'_{l'-l} > N - (d+1-d(H_m))|V|\xi$.
- $\delta(P_i, s_m, l')$ does not exist. Then, $K'_{l'-l} = K_{l'-l} - w_{l'}\left(s_m \xrightarrow{P_i} h(P_i)\right)$. $s_m \xrightarrow{P_i} h(P_i)$ is a sub-path of P_i whose length is strictly less than ξ. Thus, $w_{l'}\left(s_m \xrightarrow{P_i} h(P_i)\right) < \xi$, and $K'_{l'-l} > K_{l'-l} - \xi > K_{l'-l} - |V|\xi$. Now, since $K_{l'-l} \geq N - (d+1-d(H))|V|\xi$ by hypothesis, we get $K'_{l'-l} > N - (d+2-d(H))|V|\xi = N - (d+1-d(H_m))|V|\xi$.

This proves the first inequality. The second is implied by the first one since $N \geq (d+1)|V|\xi$.

We can thus apply the induction hypothesis and we get a dependence path from instance $(I, j+1+w_l\left(t(P_i) \xrightarrow{P_i} s_m\right), K'_1, \ldots, K'_{l(H_m)-l-1}, K'_{l(H_m)-l} - |V| w_{l(H_m)}(C(H_m)), N, \ldots, N)$ of statement s_m to instance $(I, j+1+w_l\left(t(P_i) \xrightarrow{P_i} s_m\right), K')$ of the same statement. We call this second path $\mathcal{P}_2$.

We now apply the induction hypothesis, in the particular case where all components of K (with the notations of $\mathcal{IH}(d(H_m))$) are equal to N and we get a dependence path from instance $(J, k'' - |V| w_{l(H_m)}(C(H_m)), N, \ldots, N)$ of statement s_m, to instance $(J, k'', N, \ldots, N)$ of the same statement, if J is a vector of $[1, \ldots, N]^{l(H_m)-1}$, and if k'' is an integer such that $1 + |V| w_{l(H_m)}(C(H_m)) \leq k'' \leq N$. Furthermore, this path visits, $\Omega(N^{d_S(H_m)-1})$ times, each statement S in H_m.

We let $\lambda_m = \left\lfloor \frac{K'_{l(H_m)-l} - \xi}{|V| w_{l(H_m)}(C(H_m))} - 1 \right\rfloor$. λ_m is chosen so that we can use λ_m times the induction hypothesis in the form stated above. As $K'_{l(H_m)-l}$ has been proved to be $\Omega(N)$, and as all other quantities are constant that depend only on G and not on N, λ_m is $\Omega(N)$ too. Therefore, we get a dependence path, from instance

$$(I, j+1+w_l\left(t(P_i) \xrightarrow{P_i} s_m\right), K'_1, \ldots, K'_{l(H_m)-1-l}, K'_{l(H_m)-l} - (\lambda_m+1)|V| w_{l(H_m)}(C(H_m)), N, \ldots, N)$$

to instance

$$(I, j+1+w_l\left(t(P_i) \xrightarrow{P_i} s_m\right), K'_1, \ldots, K'_{l(H_m)-1-l}, K'_{l(H_m)-l} - |V| w_{l(H_m)}(C(H_m)), N, \ldots, N)$$

of the same statement s_m. We call this path $\mathcal{P}_3$. By construction, $\mathcal{P}_3$ visits each statement S in H_m $\lambda_m \Omega(N^{d_S(H_m)-1})$ times, i.e. $\Omega(N^{d_S(H_m)}) = \Omega(N^{d_S(H)-1})$ times.

Note that by choice of λ_m, $K'_{l(H_m)-l} - (\lambda_m+1)|V| w_{l(H_m)}(C(H_m)) \geq \xi$ and this also holds for all other components of K'. We can thus apply once again Lemma 6, and we get a path, that we call $\mathcal{P}_4$, from instance $(I, j+1, K'')$ of statement $h(f)$ to instance $(I, j+1+w_l\left(t(P_i) \xrightarrow{P_i} s_m\right), K'_1, \ldots, K'_{l(H_m)-1-l}, K'_{l(H_m)-l} - (\lambda_m+1)|V| w_{l(H_m)}(C(H_m)), N, \ldots, N)$ of statement s_m, where K'' is defined by Lemma 6.

Then, we concatenate in front of this path the dependence corresponding to the edge f from instance $(I, j, N, \ldots, N)$ of statement $t(f)$ to instance $(I, j+1, K'')$ of statement $h(f)$.

Finally, concatenating paths $f + \mathcal{P}_4$, $\mathcal{P}_3$, $\mathcal{P}_2$ and $\mathcal{P}_1$ leads to the desired path.

A.4.4 Conclusion for the general case

The previous lemma leads to the desired theorem (i.e. $\mathcal{IH}(l)$ where $l = l(H)$) simply by concatenating the different paths given by Lemma 7 for each P_i.

Theorem 3 *If $\mathcal{IH}(k)$ is true, for $k < l$, then $\mathcal{IH}(l)$ is true. In other words, we have the following:*

Let I be a vector in $[1 \ldots N]^{l-1}$, let j be an integer such that $1 \leq j \leq N - |V| w_l(C(H))$, and let K be a vector in $[N - (d+1-d(H))\xi \ldots N]^{d-l}$.

Then, there exists a dependence path Φ in the expanded dependence graph of L', from instance $(I, j, N, \ldots, N)$ of statement $t(C(H))$ to instance $(I, j + |V| w_l(C(H)), K)$ of the same statement. Furthermore, Φ visits each statement v in H $\Omega(N^{d_v(H)-1})$ times.

Proof:

For each i, $1 \leq i \leq k|V|$, Lemma 7 permits to define a dependence path $\mathcal{P}_i$, in the expanded dependence graph of L', from instance $(I_1, \ldots, I_{l-1}, j + \sum_{r=1}^{i-1}(1 + w_l(P_r)), N, \ldots, N)$ of statement $t(f)$ to instance $(I_1, \ldots, I_{l-1}, j + \sum_{r=1}^{i}(1 + w_l(P_r)), K)$ of the same statement, for any vector I in $[1 \ldots N]^{l-1}$, any vector K in $[N-(d+1-d(H))\xi \ldots N]^{d-l}$ and any integer j such that $1 \leq j \leq N - \sum_{r=1}^{k|V|}(1 + w_l(P_r))$ (remember that since the weights $w_l(P_r)$ are non-negative, all intermediate sums $j + \sum_{r=1}^{i}(1 + w_l(P_r))$ belong to $[1 \ldots N]$ as soon as the two extremal sums j and $j + \sum_{r=1}^{k|V|}(1 + w_l(P_r))$ do). This path uses the edge f and all edges of P_i, and if there exists an integer m, $1 \leq m \leq c$, such that s_m is a point of P_i, then this dependence path visits each statement S_v in H_m $\Omega(N^{d_v(H)-1})$ times.

For each i, $1 \leq i \leq k|V| - 1$, we choose K to be equal to $(N, \ldots, N)$. Thus, $\mathcal{P}_i$ is a path from instance $(I_1, \ldots, I_{l-1}, j + \sum_{r=1}^{i-1}(1 + w_l(P_r)), N, \ldots, N)$ of statement $t(f)$ to instance $(I_1, \ldots, I_{l-1}, j + \sum_{r=1}^{i}(1 + w_l(P_r)), N, \ldots, N)$ of the same statement. These paths can be concatenated together and concatenated to the path $\mathcal{P}_{k|V|}$ defined above. Finally, we get a path Φ from instance $(I_1, \ldots, I_{l-1}, j, N, \ldots, N)$ of statement $t(f)$ to instance $(I_1, \ldots, I_{l-1}, j + \sum_{r=1}^{k|V|}(1 + w_l(P_r)), K)$ of the same statement.

Φ uses f and all edges of paths P_i, $1 \leq i \leq k|V|$, thus it uses all edges of $C(H)$, and, as $C(H)$ visits all statements in H, all statements in H are visited at least once. Furthermore, as for each value of m, $1 \leq m \leq c$, there exists a path P_i, $1 \leq i \leq |V|k$, which contains s_m, Φ visits each statement S in H_m $\Omega(N^{d_S(H)-1})$ times, for each value of m, $1 \leq m \leq c$. To conclude, remark that $\sum_{n=1}^{k|V|}(1 + w_l(P_n))$ is equal to $|V| w_l(C(H))$. ■

A.5 The induction

We have almost established the proof by induction. We just write it below in a formal form:

Theorem 4 *$\mathcal{IH}(k)$ is true, for all $k \geq 1$. In other words, we have the following:*

Let H be a subgraph of G which appears in the decomposition of G by Procedure First_Call. Assume that N is larger than $(d+1)|V|\xi$.

Then for all I in $[1 \ldots N]^{l(H)-1}$, for all j such that $1 \leq j \leq N - |V| w_{l(H)}(C(H))$, for all K in $[N-(d+1-d(H))\xi \ldots N]^{d-l(H)}$, there exists a dependence path Φ in the expanded dependence graph of L', from instance $(I, j, N, \ldots, N)$ of statement $t(C(H))$, to instance $(I, j + |V| w_{l(H)}(C(H)), K)$ of the same statement. Furthermore, Φ visits each statement S in H $\Omega(N^{d_S(H)-1})$ times.

Proof:

As previously announced the proof is a proof by induction on the depth $d(H)$ of H. Theorem 2 proves that Theorem 4 is true for all subgraphs of depth 1. Furthermore, if the property is true for all subgraphs of depth $k < l$, the property is true for subgraphs of depth l (Theorem 3). Therefore, the theorem is true by induction. ■

This theorem is not the final result, as we want in fact a path that visits each statement S in H $\Omega(N^{d_S(H)})$ times to prove the optimality: however, this is a simple extension of the previous result, as shown in the next section.

A.6 The optimality theorem

Theorem 5 *Let H be a subgraph of G which appears in the decomposition of G by Procedure First_Call. Assume that N is larger than $(d+1)|V|\xi$. Then there exists a dependence path Φ in the expanded dependence graph of L', which visits each statement S in H $\Omega(N^{d_S(H)})$ times.*

Proof:

Theorem 4 gives, for any vector I in $[1 \ldots N]^{l(H)-1}$, for any integer j, $1 \leq j \leq N - |V| w_{l(H)}(C(H))$, and for any vector K in $[N - (d+1-d(H))\xi \ldots N]^{d-l(H)}$, a dependence path, in the expanded dependence graph of L', from instance $(I, j, N, \ldots, N)$ of statement $t(C(H))$, to instance $(I, j + |V| w_{l(H)}(C(H)), K)$ of the same statement. Furthermore, this path visits each statement S in H $\Omega(N^{d_S(H)-1})$ times.

Among others, we can consider the paths from instance $(I, (\mu-1)|V| w_{l(H)}(C(H)) + 1, N, \ldots, N)$ of statement $t(C(H))$ to instance $(I, \mu|V| w_{l(H)}(C(H)) + 1, N, \ldots, N)$ of the same statement, for any value of μ between 1 and $\left\lfloor \frac{N-1}{|V| w_l(C(H))} \right\rfloor$. This permits to define $\Omega(N)$ paths that can be concatenated together into a longer path Φ. Finally, since each of these paths visits each statement S of H, $\Omega(N^{d_S(H)-1})$ times, Φ visits each statement S of H $\Omega(N^{d_S(H)})$ times. ■