Optimal Exploration of the Unrolling Degree for Software Pipelining *

Fermín Sánchez and Jordi Cortadella and Rosa M. Badia

Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
c/ Jordi Girona 1 i 3, Mòdul D6, 08071 Barcelona, SPAIN
e-mail: fermin@ac.upc.es

Abstract

This paper proposes a new methodology to explore the unrolling degree in software-pipelining approaches. Current software-pipelining techniques explore the solution space in only one dimension. That is, when a pipelined schedule of the loop is not found for the expected initiation interval, it is increased in one or more units and software pipelining is attempted again. This process is repeated until a schedule is found. Therefore, only the expected initiation interval of the pipelined schedule may change, and thus the solution space is explored in one dimension.

In this paper, we propose a methodology to explore pairs (II_K, K) in decreasing order of throughput, K being the unrolling degree of the loop and II_K being the expected initiation interval (length) of a schedule for K loop iterations. Results show that an optimal exploration of the unrolling degree is more effective than the traditional approach of increasing II_K when the expected schedule is not found.

1 Introduction

Resource-constrained scheduling is an NP-hard problem [1]. Since an optimal-time schedule may not be found by a known polynomial algorithm, the solution space must be examined and a criterion must be established to determine when an "acceptable" solution has been found. This paper proposes a new methodology to find a schedule with maximum throughput that improves previous approaches. Sometimes, the increasing of throughput also produces an increasing in the register pressure.

Some software pipelining techniques [2, 3, 4] find a loop schedule by assuming infinite resources. Following this, the schedule is reorganized in order to satisfy resource constraints. Other approaches consider resource constraints during the scheduling process [5, 6, 7]. Previous software pipelining approaches [5, 6], including those that consider loop unrolling [2, 7, 8], explore the solution space in one dimension, i.e. the expected initiation interval (II) of the schedule is increased when a pipelined schedule with the previous expected II is not found.

*This work was supported by CICYT TIC-95-0419

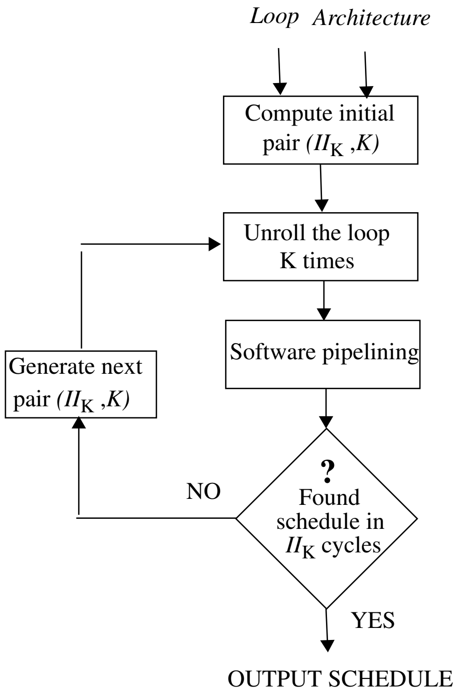

Figure 1: Flow diagram of the proposed strategy

Unlike, we propose exploring the solution space in two dimensions: the unrolling degree of the loop (K) and the expected initiation interval for the schedule of the loop unrolled K times (II_K). The basic idea consists of generating pairs (II_K, K) in decreasing order of the expected throughput of the schedule until a schedule is found. For each pair (II_K, K), the loop is unrolled K times and software pipelined is performed by using any known approach. If no schedule for the unrolled loop is found in II_K cycles, a new pair is explored. The flow diagram depicted in Figure 1 shows the proposed strategy.

An example of the effectivity of doing this exploration is presented in Section 2. Section 3 describes how the pairs (II_K, K) are generated. Section 4 presents an optimization on the exploration of the solution space. Some results are reported in Section 5. Finally, Section 6 concludes the paper.

2 Motivation and Example

2.1 Software pipelining without previous unrolling

In general, a loop is represented by a directed graph (DG) representing the data dependences among the instructions in the loop. Nodes in the DG represent instructions from the loop body, and edges represent data dependences. Dependences existing between instructions from the same iteration are called intra-loop dependences (ILPs), and they are not-labeled. Dependences existing among instructions belonging to different iterations are called loop carried dependences (LCDs), and they are labeled with the number of iterations (distance)

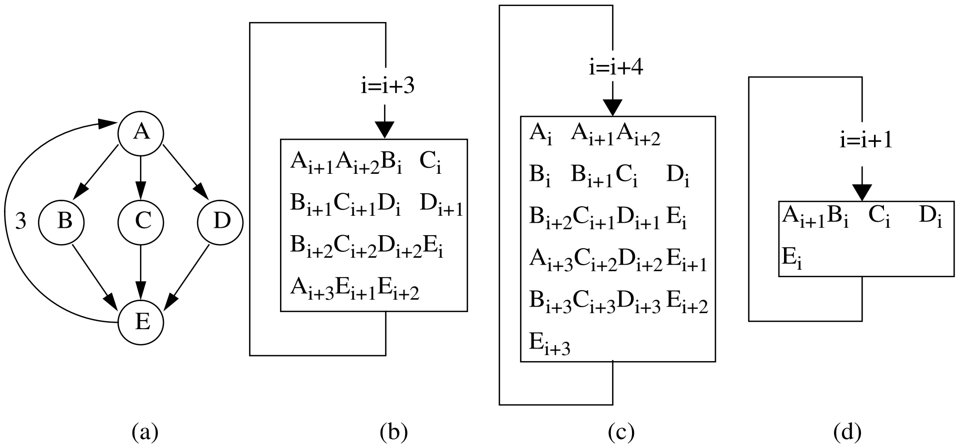

Figure 2: Different schedules for a DG (a) Example of DG (b) Schedule achieved by using the approach proposed in this paper (c) Schedule achieved by using the *optimal* unrolling degree proposed in [9, 10] (d) Schedule achieved without unrolling

crossed by the dependence. Figure 2(a) depicts a DG with 5 additions. (E, A) is an LCD with distance 3, thus indicating that E_i (instruction E from iteration i) must be executed to completion before A_{i+3} starts. The remaining dependences are ILDs. For the sake of simplicity, let us assume that the loop is executed by using 4 adders that add in one cycle. A pipelined schedule in two cycles (*II*=2) can be easily found, as shown in Figure 2(d). The throughput (average number of iterations executed per cycle) of such a schedule is $Th = \frac{1}{2}$.

2.2 Software pipelining with optimal unrolling degree

In order to obtain a minimum initiation interval (*MII*) for the schedule, a new loop representing multiple instances of the loop body is in general required [11, 12]. Unrolling the loop may increase the schedule throughput, but it also produces code and register explosion. Moreover, a prologue and an epilogue are in general necessary to execute an unrolled loop.

Two lower bounds on the *II* must be taken into account [13]:

- the *minimum initiation interval imposed by resource constraints* (*ResMII*)

  If each iteration of the loop requires using a functional unit (FU) during C cycles, and the architecture has N FUs of such type, then $II \geq \lceil \frac{C}{N} \rceil$. Therefore, the FU with the maximum such ratio determines a lower bound on *II*.

- the *minimum initiation interval imposed by the cycles* (recurrences) formed by the dependences of the loop (*RecMII*)

  Let us consider a recurrence R. A feasible schedule must fulfill $II \geq \lceil \frac{L}{D} \rceil$, where L is the sum of the result latency of the instructions in R and D is the sum of the distances of its dependences [5]. The recurrence with the maximum such ratio determines another lower bound on *II*.

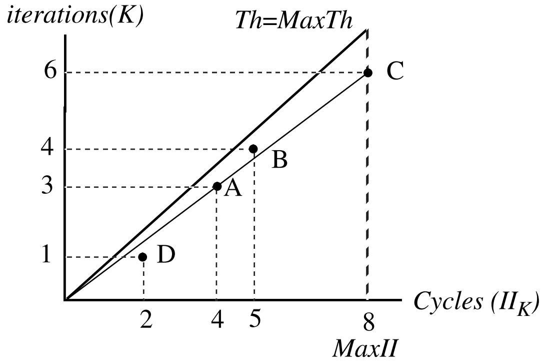

Figure 3: Representing throughput in a diagram

The *MII* of the loop is the maximum of the previous lower bounds. In the example of Figure 2(a), $MII = \frac{5}{4}$ (by assuming 4 adders), thus indicating that 4 iterations could be initiated every 5 cycles. In order to find such a schedule, the loop is unrolled 4 times and software pipelined (by using any known approach), looking for a schedule in 5 cycles [7, 9, 10]. Unfortunately, this schedule does not exist, as it is shown in [14].

When this happens, current approaches [5, 6, 7, 15] increase (by 1 or more than 1 unit [15]) the expected *II* and try to find a longer schedule by using the same unrolling degree. Then, the loop is pipelined again, looking for a schedule of 4 iterations in 6 cycles. Figure 2(c) shows the found schedule, with a throughput $Th = \frac{2}{3}$. This schedule is still far from the optimal one, but it is better than the schedule found without previous unrolling.

2.3 Contribution of this paper

Let II_K be the *initiation interval* of a schedule comprising K iterations of the loop body. In order to find a schedule that maximizes the execution throughput, we propose exploring pairs (II_K, K) in decreasing order of the expected throughput.

After the pair (II_K, K)=(5,4) representing maximum throughput (for which no schedule exists), we explore the pair (II_K, K)=(4,3). Therefore, the loop is unrolled 3 times and a pipelined schedule in 4 cycles is found, as shown in Figure 2(b). The throughput of such a schedule is $Th = \frac{3}{4}$. Although this schedule is not time-optimal (for the lower bound computed), it is better than that obtained by the other approaches. A significant speedup (1.125) is obtained with regard to [5, 6, 7, 15] due to the more exhaustive exploration of the unrolling degree of the loop.

3 Exploring the solution space

3.1 Throughput diagram

The throughput of any schedule of a loop can be represented in a diagram, as shown in Figure 3. The y axis represents the unrolling degree of the loop (K), and the x axis represents the number of cycles of the schedule (II_K). A point (II_K, K) in the diagram represents a

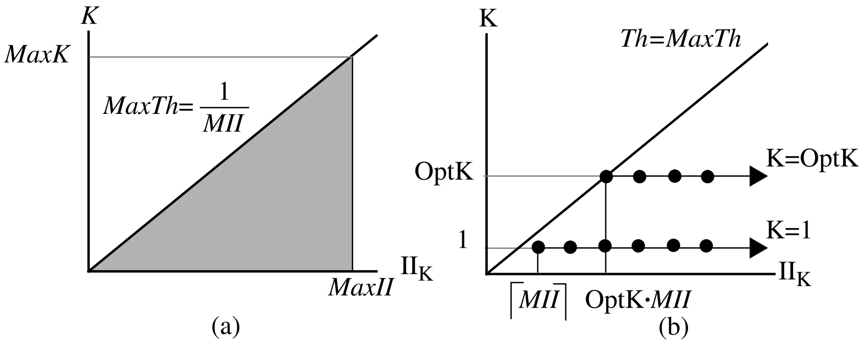

Figure 4: (a) Solution space to explore (b) Solution space explored by other approaches

schedule[1] of K iterations of the loop in II_K cycles. The throughput of such a schedule is $\frac{K}{II_K}$ iterations per cycle.

All points representing schedules with the same throughput fall in a line. For example, points A and C in Figure 3 represent two different schedules with the same throughput $Th = 0.75$. The throughput of the schedules represented by points B and D is $Th = 0.8$ and $Th = 0.5$ respectively. Note that point B is above the line that includes points A and C, since the throughput of B is greater than the throughput of A and C. However, point D is below this line because it represents a schedule with less throughput.

The maximum throughput ($MaxTh$) achievable by a schedule is bounded by the recurrences of the loop and the set of resources of the architecture ($MaxTh = \frac{1}{MII}$ [16]). All the schedules achieving maximum throughput fall in the line $Th = MaxTh$. Any feasible schedule of the loop must be represented by a point (with integer values for K and II_K) below this line. We are interested in exploring the points in the diagram in decreasing order of throughput. Since the number of points below the line $Th = MaxTh$ is infinite, we will limit them by constraining the maximum number of cycles of any schedule. This bound is called $MaxII$.

The points to explore are included in the triangle formed by the II_K axis and the lines $II_K = MaxII$ and $Th = MaxTh$. The shadowed triangle in Figure 4(a) represents the solution space. Figure 4 compares the solution space proposed to be explored to the solution space explored by other approaches.

- Software pipelining techniques that do not perform loop unrolling explore the points shown in the arrow represented on the line $K = 1$. Note that the first point to explore may not be on the line $Th = MaxTh$, and therefore may not represent a time-optimal schedule (its II is $II = \lceil MII \rceil \geq MII$).

- Software pipelining approaches considering loop unrolling explore the points shown in the arrow represented on the line $K = OptK$. $OptK$ is the optimal unrolling degree (UD) of the loop. Given a DG with $MII = \frac{a}{b}$, $OptK = \frac{b}{\gcd(a,b)}$ [9, 10, 12]. Therefore, a theoretical optimal-time schedule may exist in $OptK \cdot MII$ cycles. For a given value of K, the distance between any two points in the line is always an integer value. In

[1] This schedule may not exist.

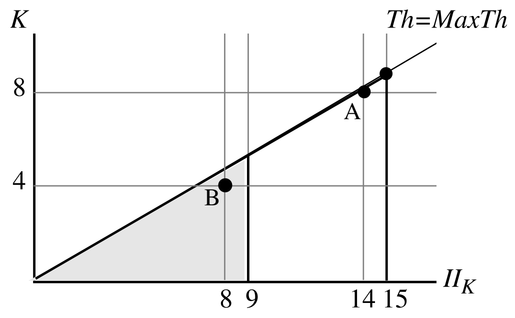

Figure 5: Triangles delimited by $MaxII = 9$ and $MaxII = 15$

general, the distance between two consecutive points is one unit, but this depends on the approach (for example, in [15] the proposed increment is $\max(\lfloor 0.04 \cdot II \rfloor, 1)$. This is done to avoid spending an excessive amount of time compiling large complex loops).

- Whilst previous approaches explore the solution space in only one dimension (they change the expected *II* for a fixed *UD*), we propose exploring the solution space in two dimensions (by changing both the expected *II* and the *UD*), as shown by the shadowed triangle from Figure 4(a). Exploring different *UD*s may increase the throughput in face to only exploring different *II*s. As an example, Figure 5 depicts a diagram with two triangles corresponding to $MaxII = 9$ and $MaxII = 15$. The point $A = (14, 8)$ belongs to the triangle delimited by the line $MaxII = 15$, while the point $B = (8, 4)$ belongs to both triangles. Although A represents a potential schedule of 14 cycles and B a potential one of only 8 cycles, the schedule throughput of A ($Th = \frac{8}{14} = 0.57$) is higher than that of B ($Th = \frac{4}{8} = 0.50$).

3.2 Farey's series

For a fixed $n > 0$, the sequence of all the reduced fractions with nonnegative denominator $\leq n$ arranged in increasing order of magnitude is called the *Farey's series* of order n, and denoted by F_n [17]. For example, F_5 is the series of fractions:

$$\frac{0}{1}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \ldots$$

Figure 6 shows a diagram representing such a sequence. Note that point (4,2) is not in the sequence, since it is not reduced (it represents the same fraction as point (2,1), as the line that crosses both points in Figure 6 shows).

The throughput of a schedule represented in a diagram as the one in Figure 3 is a fraction with a denominator lower than or equal to *MaxII*. Therefore, the Farey's series of order *MaxII* (F_{MaxII}) forms the sequence of points to be explored. We are interested in exploring the elements (fractions) of F_{MaxII} in decreasing order, starting at the first element E_i fulfilling $E_i \leq MaxTh$. In the next section, we will explain an efficient way to perform such an exploration.

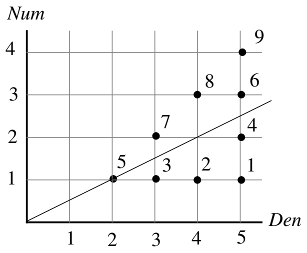

Figure 6: Representing Farey's series F_5 in a diagram

3.3 Exploring Farey's series in decreasing order of magnitude

By using the equations from [17], we derive new equations to recursively generate the elements from Farey's series in decreasing order of magnitude. The ith element of the Farey's series F_{MaxII}, $E_i = \frac{X_i}{Y_i}$, can be recursively calculated by using the two following elements as follows:

$$X_i = \left\lfloor \frac{Y_{i+2} + MaxII}{Y_{i+1}} \right\rfloor \cdot X_{i+1} - X_{i+2} \qquad Y_i = \left\lfloor \frac{Y_{i+2} + MaxII}{Y_{i+1}} \right\rfloor \cdot Y_{i+1} - Y_{i+2} \qquad (1)$$

Note that two consecutive elements of Farey's series F_{MaxII} are required to recurrently compute the previous one. We will now describe the way to compute the two first elements to be considered. Such elements must be as close as possible to the fraction representing maximum throughput, which is the first fraction that may represent a feasible schedule. The throughput of the schedule represented by the first element must be $MaxTh$ whenever possible.

Theorem 3.1 *The lowest fraction with denominator $MaxII$, with a value greater than or equal to $MaxTh$ is:*

$$\frac{\left\lceil \frac{MaxII}{MII} \right\rceil}{MaxII} \qquad (2)$$

Proof:

Let us consider the throughput diagram from Figure 7. Point Y is in the line $Th = MaxTh$, but it cannot be considered because it is not represented by integer values of II_K and K. For a given $MaxTh$ and $MaxII$, we are interested in finding the fraction corresponding to point X. Since point Y fulfills $Th = \frac{1}{MII}$, we have:

$$Th(Y) = \frac{1}{MII} = \frac{K_Y}{MaxII} \Rightarrow K_Y = \frac{MaxII}{MII}$$

Since Farey fractions are formed by integer numerators and denominators, we are interested in finding $\lceil K_Y \rceil$. Point $X = (MaxII, \lceil K_Y \rceil)$ represents the first point that we are looking for, with a throughput:

$$Th(X) = \frac{\lceil K_Y \rceil}{MaxII} = \frac{\left\lceil \frac{MaxII}{MII} \right\rceil}{MaxII}$$

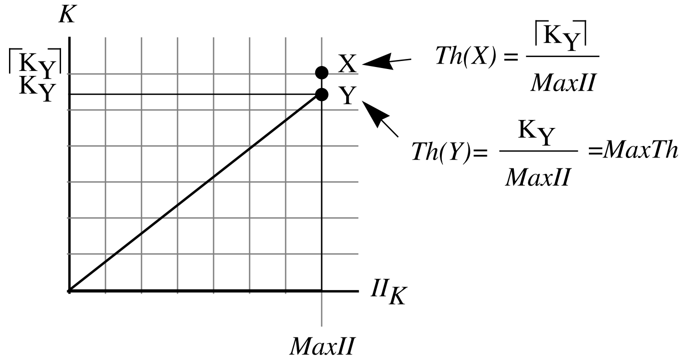

Figure 7: First element of Farey's series to be considered

Point X may not represent a feasible schedule, since $Th(X)$ may be greater than $MaxTh$. □

The fraction expressed by Equation (2) is the first one to be analyzed. However, to use the recurrence from Equation (1) we need two consecutive fractions. By using theorems from [18], we derive a recurrent equation to compute a fraction from F_{MaxII} by using the next one:

$$X_i = x + X_{i+1} \cdot \left\lfloor \frac{MaxII - y}{Y_{i+1}} \right\rfloor \qquad Y_i = y + Y_{i+1} \cdot \left\lfloor \frac{MaxII - y}{Y_{i+1}} \right\rfloor \tag{3}$$

where x and y are two integers satisfying the relation $\gcd(-Y_i, X_i) = (-Y_i) \cdot x + X_i \cdot y$. The coefficients x and y can easily be computed by using the *extended gcd* [18]. Figure 8 presents an algorithm to calculate the second (previous) Farey fraction by using Equation (3). In the algorithm, called *Previous_fraction*, function ExtendedGCD(n, m) returns the list $[g, x, y]$, where g is $\gcd(n, m)$, and x and y are two integers satisfying the relation $g = nx + my$. Once both fractions have been obtained, we can recurrently use Equation (1) to compute the remainder elements of the series. The first fraction that may correspond to a feasible schedule is the fraction $E_j = \frac{x_j}{y_j}$, such that $\frac{x_j}{y_j} \leq MaxTh$. Despite all the fractions can be generated by using the algorithm proposed in Figure 8, we use Equation (1) as soon as possible because of its lower execution time.

Since the fractions are reduced, not all points in the throughput diagram are represented by the elements of Farey's series (an example is given in Figure 6. Point (4,2) is not represented in the series, since it is not reduced). However, if a schedule is not found for a given point (II_K, K), this does not mean that a schedule for the point $(n \cdot II_K, n \cdot K)$ does not exist[2] for any integer $n > 0$. Therefore, for each Farey fraction we generate all the equivalent ones with a denominator lower than or equal to *MaxII*. Such equivalent fractions are generated in increasing order of denominator, since we are interested in first exploring the shortest schedules (for schedules with the same throughput). For example, if we are interested in the series F_5 in decreasing order of magnitude, and the maximum throughput achievable by a

[2] Despite not finding any such case in our experiments, we have not been able to demonstrate the contrary.

function *Previous_fraction*$(X_k, Y_k, MaxII)$;

$\{$Reads the fraction $\frac{X_{k+1}}{Y_{k+1}}$ of the Farey series F_{MaxII} and computes the previous fraction $\frac{X_k}{Y_k}$ $\}$

$[g, x, y]$:=ExtendedGCD$(-Y_{k+1}, X_{k+1})$;

$r := \lfloor \frac{MaxII-y}{Y_{k+1}} \rfloor$;

return $\frac{x+r\cdot X_{k+1}}{y+r\cdot Y_{k+1}}$

Figure 8: Algorithm to compute a Farey fraction by using the next one

schedule is $MaxTh = \frac{4}{5}$, the fractions to explore are:

$$\frac{4}{5}, \frac{3}{4}, \frac{2}{3}, \frac{3}{5}, \frac{1}{2}, \frac{2}{4}, \frac{2}{5}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}$$

In summary, the algorithm to compute all the points (II_K, K) in decreasing order of throughput is as follows:

1. Compute the lowest reduced fraction $\frac{x_k}{MaxII}$ such that $\frac{x_k}{MaxII} \geq MaxTh$. This is the first fraction $\frac{x_k}{y_k}$ of the series in which we are interested. Equation (2) shows such a fraction.

2. Compute the previous fraction, $\frac{x_{k-1}}{y_{k-1}}$, by using Equation (3). Generate all the fractions equivalent to $\frac{x_{k-1}}{y_{k-1}}$ with a denominator lower than or equal to $MaxII$.

3. Compute the remainder fractions by recurrently using Equation (1). For each fraction, generate all the equivalent fractions with a denominator lower than or equal to $MaxII$. This calculation is faster than using step 2 for each element.

4 Reducing the solution space

4.1 Number of points to explore

The number of points to explore may be quite large. Since only integer values for K and II_K are explored, the number of points inside the triangle that forms the solution space is in the order of the area of the triangle (see Figure 4(a)): *Number of points* $\approx \frac{MaxK\cdot MaxII}{2}$.

$MaxK$ corresponds to the number of iterations involved into a schedule with maximum throughput, whose length is $MaxII$ cycles. The throughput of such a schedule is $\frac{1}{MII}$ iterations per cycle. Since $\frac{MaxK}{MaxII} = \frac{1}{MII}$, we conclude that $MaxK = \frac{MaxII}{MII}$. Therefore, the number of points in the throughput diagram is in the order of:

$$\textit{Number of points} \approx \frac{MaxK \cdot MaxII}{2} = \frac{\frac{MaxII}{MII} \cdot MaxII}{2} = \frac{MaxII^2}{2 \cdot MII}$$

We conclude that the number of points is proportional to the square of $MaxII$. Thus, by reducing $MaxII$, the number of points in the solution space quadratically decreases.

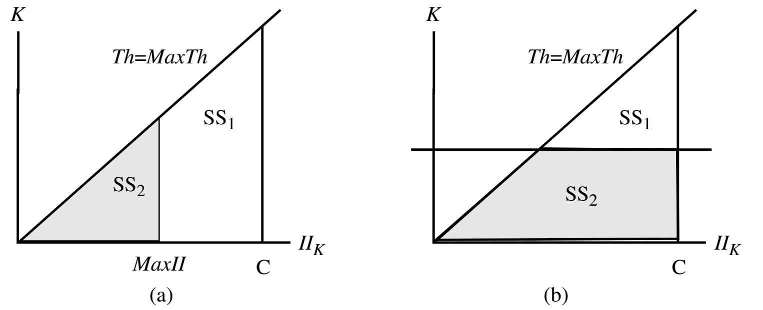

Figure 9: Reducing solution space (a) By limiting *MaxII* (b) By limiting K

4.2 Reducing MaxII

The maximum number of cycles of a schedule is determined by the designer. However, when *MaxII* is a large value, the solution space is too large. In these cases, it is useful to look for a *good* solution rather than an optimal solution if this significantly reduces the solution space.

Definition 4.1 Good subspace to explore
Let us consider two solution spaces SS_1 *and* SS_2*, such that* $SS_2 \subset SS_1$*. We define* SS_2 *as a* good subspace to explore SS_1 *if, for each point* (II_K, K) *representing a schedule with throughput* $\frac{K}{II_K}$*, there exists a point* (II'_K, K') *such that* $\frac{K'}{II'_K} \geq z \cdot \frac{K}{II_K}$*, where* $z \in (0,1]$ *and* z *tends to 1.*

$$\forall (II_K, K) \in SS_1,\ \exists (II'_K, K') \in SS_2 \textit{ such that } \frac{K}{II_K} \geq \frac{K'}{II'_K} \geq z \cdot \frac{K}{II_K}$$

The previous definition indicates that, for each point belonging to SS_1, a point exists in SS_2 with a *similar* throughput.

The results of our experiments suggest that, if a schedule exists for a point (II_K, K), a schedule also exists for any point (II'_K, K') such that $\frac{K'}{II'_K} \leq \frac{K}{II_K}$.

The solution space can be reduced in several ways. Figure 9 shows two possibilities:

- By limiting II_K, the maximum number of cycles of the schedule (Figure 9(a)).
- By limiting K, the maximum number of iterations of the loop (Figure 9(b)).

Limiting II_K (*MaxII*) also produces a reduction in the number of iterations to unroll. Furthermore, it is better than a simple reduction in K as far as reducing the number of registers required by the schedule is concerned (in general, the greater the number of cycles, the greater the register requirements). Thus, for a given maximum number of cycles C determining a solution space SS_1, our objective is to find a $MaxII < C$ determining a solution space SS_2 such that, for any schedule S in SS_1 there exists a schedule in SS_2 with at least z times the throughput of S.

The number of fractions in F_C is greater than the number of fractions in F_{MaxII}. The fractions in a Farey's series are not uniformly distributed. Therefore, the number of fractions

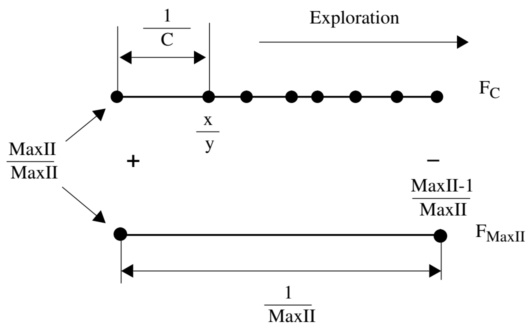

Figure 10: Comparing number of points in F_C and F_{MaxII}

belonging to F_C that are between two consecutive fractions from F_{MaxII} is not constant. We are interested in studying two consecutive fractions such that the difference between them is maximum. Note that this is the worst case for our proposal, which is finding a *good* subspace determined by *MaxII*.

For a given Farey's series F_C, the maximum difference between two consecutive fractions corresponds to fractions $\frac{1}{1}(\frac{C}{C})$ and $\frac{C-1}{C}$ (there are other consecutive fractions with the same difference). Such a difference is $\frac{1}{C}$. Figure 10 shows the number of fractions from F_C that are between two consecutive fractions from F_{MaxII} between which there is the largest difference.

The point labelled $\frac{MaxII-1}{MaxII}$ in Figure 10 must represent a schedule with at least z times the throughput of the point labelled $\frac{x}{y}$. Therefore, we have:

$$z \cdot (1 - \frac{1}{C}) \leq \frac{MaxII - 1}{MaxII}$$

and, by manipulating with the former equation, we obtain:

$$MaxII \geq \frac{1}{\frac{z}{C} + 1 - z}$$

$$MaxII = \left\lceil \frac{1}{\frac{z}{C} + 1 - z} \right\rceil$$

In order to illustrate the effectivity of reducing the solution space, we will show the *MaxII* calculated for $C = 10$, $C = 50$, $C = 100$ and $C = 200$, by assuming $z = 0.95$. That is, the reduced solution space for which a schedule with at least a 95 per cent of the throughput exists for a given C:

$$C = 10 \Longrightarrow MaxII = 7$$

$$C = 50 \Longrightarrow MaxII = 15$$

$$C = 100 \Longrightarrow MaxII = 17$$

$$C = 200 \Longrightarrow MaxII = 19$$

We conclude that using a *small MaxII* is possible to achieve a *good* solution. Reducing the solution space may significantly reduce the time required to find a *good* schedule.

4.3 Figures of merit

This section presents a figure of merit to evaluate how good a schedule is.

Definition 4.2 : Time-optimal schedule
A schedule S is time-optimal *when its throughput is $Th(S) = MaxTh$.*

Definition 4.3 ε: Schedule Efficiency Ratio
We define the efficiency ratio *of a schedule S, ε, as:*

$$\varepsilon = \frac{Th(S)}{MaxTh}$$

Definition 4.4 σ: Schedule Size Ratio
We define the size ratio *of a schedule, σ, as*

$$\sigma = \gcd(K, II_K)$$

The *efficiency ratio* ε is a real number belonging to the interval (0..1]. It measures the time efficiency of a schedule with respect to a (theoretical) time-optimal one. The efficiency ratio of a time-optimal schedule is $\varepsilon = 1$.

The *size ratio* σ is a positive integer and denotes the number of times the loop has been unrolled with respect to the smallest value of K required to obtain a schedule with the same *efficiency ratio*. This figures of merit are independent from the technique used for loop pipelining.

5 Experimental Results

In order to show the efficiency of exploring all the solution space instead of only exploring one dimension (incrementing the II when the expected schedule is not found), we have used a family of loops with a single topology, but quite difficult to pipeline in some cases.

Figure 11(a) shows the pattern of the proposed family of loops. All the loops have a similar topology, and the parameters that define a loop are:

- the number of nodes connected between the node source and the node sink (n)
- the distance of the edge that connects node sink to node source (d)

Figure 11(b) shows an example for $n = 4$ and $d = 2$. For the sake of simplicity, we will assume here that all the instructions in the loops can be executed in a single cycle in any functional unit (FU).

Table 1 shows the results obtained for $n = 3, 4, 5$ and $d = 1, 2, 3$ by using different number of FUs. First columns show the characteristics of each loop and each set of resources. Next column ($MaxTh$) shows the maximum throughput achievable for each loop with the given FUs. The following columns show the results obtained by exploring Farey's series. The software pipelining algorithm used is *UNRET* [16]. For each loop we show the throughput obtained, the point (II_K, K) explored to obtain such a throughput and the *efficiency ratio* of the found schedule. The *size ratio* is not shown for Farey's series because it is one in all cases. We have considered $MaxII = 50$. Next columns show the results obtained by incrementing II instead of exploring Farey's series (II++), also by using *UNRET* as software pipelining

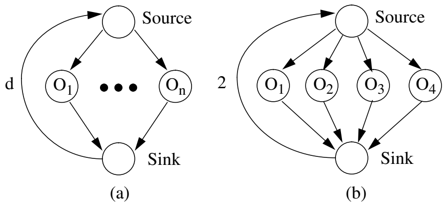

Figure 11: (a) Pattern of the loops family (b) Loop example for $n = 4$ and $d = 2$

algorithm. The initial unrolling degree has been calculated by using the optimal unrolling degree proposed in [9, 10]. The *size ratio* has been included because in several cases it is greater than one. For the sake of clarity, σ is only explicit when it is different from one. Finally, last column shows the *speed up* achieved by exploring Farey's series instead of only incrementing II.

The obtained results suggest interesting conclusions.

- On one hand, the *schedule size ratio* obtained by exploring Farey's series is one in all cases. Therefore, Farey's series allows to obtain schedules as short as possible for a given throughput. This also produces the lowest register pressure. We conclude that the number of cycles of the loop schedule and the register pressure can be minimized by using Farey's series.

  On the other hand, incrementing II produces schedules with a *size ratio* greater than one in 26 % of the cases. Thus, the number of cycles required of the schedule is greater than the necessary, as well as the register pressure. This drawback can be solved by reducing the fraction $\frac{K}{II_K}$ (obtained after increasing II) that produces the schedule. For example, for $n = 3, d = 2, FUs = 3$ in Table 1, $II++$ explores 3 iterations in 6 cycles. However, $\frac{3}{6}$ can be reduced to $\frac{1}{2}$, thus indicating that the same throughput (and less register pressure) is obtained by exploring 1 iteration in 2 cycles. To our knowledge, this reduction has not been proposed in the literature up to now.

- The use of Farey's series instead of the classical approach of incrementing II increases the throughput in 11 % of the cases. Moreover, this improvement goes from a 5 % to a 12.5 %. These results show that using Farey's series improves the results obtained by simply increasing the initiation interval.

- After performing several experiments [16] we have verified that, for a given loop, if no schedule is found for the pair (II_K, K), no schedule is not found for the pair $(i \cdot II_K, i \cdot K)$ either, i being a positive integer number. This fact suggests that only reduced fractions should be explored, thus reducing the number of points in the solution space. Moreover, since non-reduced fractions usually represent large schedules and large loops (unrolled several times), not exploring these points produces a tangible decrease in the time required to find the final schedule. This is because the time required to schedule a loop increases with the number of times the loop has been unrolled.

Loop			MaxTh	Farey			II++				Speed up
n	d	FUs		Th	(II_K, K)	ε	Th	(II_K, K)	ε	σ	
		2	1/3	1/4	(4,1)	0.75	1/4	(4,1)	0.75		
	1	3	1/3	1/3	(3,1)	1	1/3	(3,1)	1		
		4	1/3	1/3	(3,1)	1	1/3	(3,1)	1		
		3	3/5	1/2	(2,1)	0.83	1/2	(6,3)	0.83	3	
3	2	4	2/3	2/3	(3,2)	1	2/3	(3,2)	1		
		5	2/3	2/3	(3,2)	1	2/3	(3,2)	1		
		3	3/5	3/5	(5,3)	1	3/5	(5,3)	1		
	3	4	4/5	3/4	(4,3)	0.93	2/3	(4,6)	0.83	2	1.125
		5	1	1	(1,1)	1	1	(1,1)	1		
		2	1/3	1/4	(4,1)	0.75	1/4	(4,1)	0.75		
	1	3	1/3	1/4	(4,1)	0.75	1/4	(4,1)	0.75		
		4	1/3	1/3	(3,1)	1	1/3	(3,1)	1		
		3	1/2	1/2	(2,1)	1	1/2	(2,1)	1		
4	2	4	2/3	1/2	(2,1)	0.75	1/2	(4,2)	0.75	2	
		5	2/3	2/3	(3,2)	1	2/3	(3,2)	1		
		3	1/2	1/2	(2,1)	1	1/2	(2,1)	1		
	3	4	2/3	2/3	(3,2)	1	2/3	(3,2)	1		
		5	5/6	3/4	(4.3)	0.9	5/7	(7,5)	0.85		1.05
		2	2/7	1/5	(5,1)	0.7	1/5	(10,2)	0.7	2	
	1	3	1/3	1/4	(4,1)	0.75	1/4	(4,1)	0.75		
		4	1/3	1/4	(4,1)	0.75	1/4	(4,1)	0.75		
		3	3/7	2/5	(5,2)	0.93	3/8	(8,3)	0.87		1.06
5	2	4	4/7	1/2	(2,1)	0.87	1/2	(4,8)	0.87	4	
		5	2/3	1/2	(2,1)	0.75	1/2	(4,2)	0.75	2	
		4	4/7	4/7	(7,4)	1	4/7	(7,4)	1		
	3	5	5/7	5/7	(7,5)	1	5/7	(7,5)	1		
		6	6/7	3/4	(4,3)	0.87	3/4	(8,6)	0.87	2	

Table 1: Experimental results

6 Conclusions

This paper has presented a new methodology to explore the solution space in software pipelining. The methodology is based on exploring both the unrolling degree of the loop and the expected initiation interval of the schedule. The space is explored in two dimensions, instead of the classical exploration in only one dimension (by increasing the expected initiation interval), as it is done by current software pipelining techniques. Results show that improvements are obtained in several cases in both increasing throughput and reducing register pressure.

References

[1] M.R. Garey and D.S. Johnson. *A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[2] B. Su, S. Ding, and J. Xia. URPR: An extension of URCR for software pipelining. In *Proc. of the 19th Annual Workshop on Microprogramming and Microarchitecture (MICRO19)*, pages 104–108, October 1986.

[3] A. Aiken and A. Nicolau. Optimal loop parallelization. In *Proc. of the ACM SIGPLAN88 Conf. on Programming Languages Design and Implementation*, pages 308–317, 1988.

[4] R. Potasman, J. Lis, A. Nicolau, and D.D. Gajski. Percolation based synthesis. In *Proc. of the 27th Design Automation Conf. (DAC)*, pages 444–449, 1990.

[5] B.R. Rau and C.D. Glaeser. Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing. In *Proc. of the 14th Annual Workshop on Microprogramming*, pages 183–198, October 1981.

[6] M. Lam. Software pipelining: An effective scheduling technique for VLIW machines. In *Proc. of the ACM SIGPLAN88 Conf. on Programming Languages Design and Implementation*, pages 318–328, June 1988.

[7] M. Rim. *High-Level Synthesis of VLSI Designs for Scientific Programs.* PhD thesis, University of Wisconsin-Madison, 1993.

[8] F. Bodin and F. Charot. Loop optimization for horizontal microcoded machines. In *Proc. of the ACM Int. Conf. on Supercomputing (ICS)*, pages 164–176, June 1990.

[9] R.B. Jones and V.H. Allan. Software pipelining: A comparison and improvement. In *Proc. of the 23rd Annual Workshop on Microprogramming and Microarchitecture (MICRO23)*, pages 46–56, November 1990.

[10] K.K. Parhi and D.G. Messerschmitt. Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding. *IEEE Trans. Computers*, 40(2):178–195, February 1991.

[11] F. Catthoor, W. Geurts, and H. De Man. Loop transformation methodology for fixed-rate video, image and telecom processing applications. In *Proc. of the Int. Conf. on Application-Specific Array Processors (ASAP94)*, 1994.

[12] L-G. Jeng and L-G. Chen. Synthesis of rate-optimal DSP algorithms by pipeline and minimum unfolding. In *Proc. of the 6th Int. Workshop on High-Level Synthesis*, pages 159–168, November 1992.

[13] B.R. Rau, M.S. Schlansker, and P.P. Tirumalai. Code generation schema for modulo scheduled loops. In *Proc. of the 25th Annual Int. Symp. on Microarchitecture (MICRO25)*, pages 158–169, December 1992.

[14] J. Cortadella, R. M. Badia, and F. Sánchez. A mathematical formulation of the loop pipelining problem. Technical Report UPC-DAC-1995-36, Department of Computer Architecture (UPC), October 1995. Available in www.ac.upc.es.

[15] R. A. Huff. Lifetime-sensitive modulo scheduling. In *Proc. of the 6th Conf. Programming Languages Design and Implementation*, pages 258–267, 1993.

[16] F. Sánchez. *Loop Pipelining with Resource and Timing Constraints.* PhD thesis, Universitat Politècnica de Catalunya (Spain), October 1995. Also available as technical Report UPC-DAC-1996-17, Department of Computer Architecture (UPC, Spain). Available in www.ac.upc.es.

[17] M.R. Schroeder. *Number Theory in Science and Communication.* Springer-Verlag, 1990.

[18] G.H. Hardy and E.M. Wright. *An Introduction to the Theory of Numbers.* Oxford University Press, 1979.