On scanning space-time mapped WHILE loops

Martin Griebl and Christian Lengauer

Fakultät für Mathematik und Informatik, Universität Passau,
D–94030 Passau, Germany, email: {griebl,lengauer}@fmi.uni-passau.de

Abstract. The mathematical model for the parallelization, or "space-time mapping", of loop nests is the polyhedron model. The presence of WHILE loops in the nest complicates matters because the parallelized loop nest does not correspond to a polyhedron but instead to a subset that resembles a (multi-dimensional) comb. This comb can take on shapes that make the precise enumeration of its points by any parallel target loop nest impossible. We describe how to augment the target loop code to scan a finite superset of the comb and restrict execution of the loop body to the points of the comb.

Keywords: loop parallelization, parallelizing compilation, space-time mapping, WHILE loop

1 Introduction

With the advent of increasingly large and affordable parallel computing systems, the topic of loop parallelization has recently received increased attention. Modelling loops with convex polyhedra has proved useful in characterizing and analyzing the space of possible parallelizations of set of nested loops. This model has, so far, been applied only to loops all of whose bounds are known at compile time (FOR loops). Nests of FOR loops are modelled with polytopes, i.e., finite polyhedra. We present aspects of an extension of the model to loops whose upper bounds are not known before compile time (WHILE loops).

2 Notation

Our quantifier notation follows Dijkstra and Scholten [3]. Quantification over a dummy variable x is written $(Q\ x : R.x : P.x)$. Q is the quantifier, R is a predicate in x representing the range, and P is a term that depends on x. Formal logical deductions are given in the form:

$$\begin{array}{ll} & \mathit{formula}_1 \\ \mathit{op} & \{\ \text{comment explaining the validity of relation } \mathit{op}\ \} \\ & \mathit{formula}_2 \end{array}$$

where op is an operator from the set $\{\Leftarrow, \Leftrightarrow, \Rightarrow\}$.

A WHILE loop is commonly written: `WHILE` *condition* `DO` *body*. Since we add an artificial index that counts the number of executed iterations, we denote a WHILE loop by: `FOR` *index* := 0 `WHILE` *condition*(*index*) `DO` *body*. Rewriting the standard syntax into our notation requires that *index* is a new variable that does not occur in the original loop. By supplying *index* as a parameter to *condition*, we make the dependence of *condition* on the WHILE iteration explicit.

3 FOR loops and the polytope model

Parallelization in the polytope model proceeds as follows [6].

First, one transforms d perfectly nested source loops into a d-dimensional polytope. We call this polytope the *index space* and denote it with $\mathcal{I}$. Each loop defines the extent of $\mathcal{I}$ in one dimension. Each point of $\mathcal{I}$ represents one iteration step of the loop nest. The coordinates of the point are given by the values of the loop indices at that step, i.e., the $\mathcal{I} \subset \mathbb{Z}^d$.

Then, one applies an affine coordinate transformation $\mathcal{T}$, the "space-time mapping", to the polytope and obtains another polytope in which some dimensions enumerate space and the others enumerate time. (In the simplest case, there is only one time dimension.) We call the transformed polytope the *target space* and denote it with $\mathcal{TI}$. The standard techniques for space-time mapping in the polytope model require uniform dependences [5, 7, 8]. In this case, all points of $\mathcal{I}$ have identical dependence vectors that are independent of the problem size.

Finally, one translates this polytope back to a nest of target loops, where each space dimension becomes a parallel loop and each time dimension becomes a sequential loop. The target loop nest specifies asynchronous parallelism if the outer loops are the parallel ones, and synchronous parallelism if the outer loops are the sequential ones. When $\mathcal{T}$ is not unimodular, i.e., its inverse is not an integer matrix, $\mathcal{TI}$ contains "holes", i.e., it is not convex even though $\mathcal{I}$ is [1]. In this case, one has to take care that the target loops do not enumerate the holes. We call the process of enumerating the points of the target space *scanning*. Luckily, non-unimodular mappings distribute holes evenly throughout the target space. Therefore, there is always a target loop nest that scans $\mathcal{TI}$ precisely—whether $\mathcal{T}$ is unimodular [2] or not [9].

4 WHILE loops and the polyhedron model

A loop nest with WHILE loops is modelled by a polyhedron, not by a polytope. Since, at compile time, the upper bound of a WHILE loop is unknown, the polyhedron is infinite in the corresponding dimension.

Also, a point of the polyhedron has a slightly different interpretation than in the FOR loop model: it represents not only one iteration step but also the boundary test preceding the step. One consequence is that an empty loop is modelled by one point, not by the empty set. Thus, any point of the index space that represents the first action of, say, loop w at level d stands for boundary test for all outer loops up to and including w (i.e., d tests) and for the first iteration of loop w, if any.

A nest with WHILE loops executes only a subset of the infinite index space $\mathcal{I}$. We call this subset the *execution space* and name it $\mathcal{X}$. The dependences of points in $\mathcal{X}$ make it look like a (possibly, multi-dimensional) comb [4], as demonstrated in Figure 1. Therefore, we call it also the *execution comb*. We give its transformation under the space-time mapping the name $\mathcal{TX}$.

In its pure form, the polyhedron model is restricted to uniform dependences [5, 7, 8]. The dependences introduced by a WHILE index are not uniform: they do not apply to every point of the comb. (For example, in Figure 1, the i dependences are only present for one value of j.) One can view any affine space-time mapping of

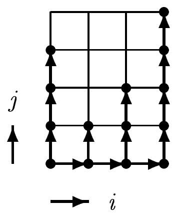

Fig. 1. A non-convex execution space with data dependences.

the comb as a restriction of the same mapping on the comb's "uniform closure". In a uniform closure, one adds arrows to make any non-uniform dependence uniform. The affinity of the mapping ensures that the added arrows do not violate the mapping's consistency—that is, the restriction of the mapping derived for the uniform closure to the comb is exactly the mapping we want. This gives us the right to make use of the theory for FOR loop parallelization.

If only the outermost loop of the nest is a WHILE loop, the execution space of the nest is still convex and the methods for FOR loops apply without reservation. But if we allow at least one WHILE loop properly inside the nest, only space-time mappings that fulfill certain conditions yield a precisely scanable execution space [4]. For example, consider Figure 2: there is no nest of target loops that scans the points without encountering the hole at coordinates $(3, 2)$.

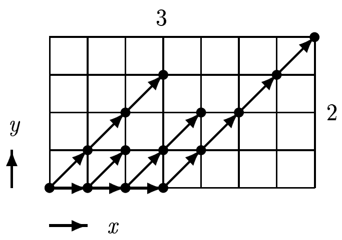

Fig. 2. Unscanable transformed execution space with data dependences.

One more remark is in order. In the source loop nest, we set the lower bound of a WHILE loop to zero, for any level r. In the transformed execution space, this bound maybe mapped to an arbitrary integer lb_r.

Points in the execution space are d-dimensional but, to save space, we abbreviate the representation of a point at the base of the multi-dimensional comb in dimension r from $(p_1, \cdots, p_r, lb_{r+1}, \cdots, lb_d)$ to $(p_1, \cdots, p_r)$.

5 Controlling the scan of an unscanable execution space

We call an execution space that cannot be scanned by any target loop nest without encountering holes *unscanable*. Scanning an unscanable execution space $\mathcal{TX}$ with some fixed target loop nest yields a proper superset of $\mathcal{TX}$; we name this set $\mathcal{TS}$ and its inverse image under the space-time mapping $\mathcal{S}$. In this section, we define a logical predicate that is an accurate recognizer of the points in $\mathcal{TX}$, i.e., that distinguishes the points in $\mathcal{TX}$ from those outside. In the following section, we use this predicate to prevent the execution of holes in the target polyhedron at run time.

Definition 1 (Activity recognizer $active_r$). Let r by some level of the source loop nest and w the WHILE loop at that level. $active_r$ holds for any point p in $\mathcal{I}$ iff the source program enumerates p, that is, iff at least the WHILE condition $cond_r$ of loop w is evaluated at point p. Formally:

$$
\begin{array}{lll}
\multicolumn{3}{l}{(\forall\ (p_1,\cdots,p_d)\ :\ (p_1,\cdots,p_d)\in\mathcal{I}\ :\ (\forall\ r:1\leq r\leq d\ :\ active_r(p_1,\cdots,p_r) =} \\
\quad \textbf{if}\ p_r > lb_r & \to active_r(p_1,\cdots,p_r-1)\wedge cond_r(p_1,\cdots,p_r-1) & (1) \\
\quad [\!]\ p_r = lb_r \wedge r>1 & \to active_{r-1}(p_1,\cdots,p_{r-1})\wedge cond_{r-1}(p_1,\cdots,p_{r-1}) & (2) \\
\quad [\!]\ p_r = lb_r \wedge r=1 & \to true & (3) \\
\quad [\!]\ p_r < lb_r & \to false & (4) \\
\quad \textbf{fi}\ )) & &
\end{array}
$$

The alternatives of the defining equation can be explained as follows. Point $(p_1,\cdots,p_r)$ is active with respect to level r iff

(1) the point represents some non-leading step of a loop, the WHILE condition holds and the previous step is active with respect to level r, or
(2) the point represents the first step of an inner loop, the WHILE condition holds for the immediately enclosing loop and the point is active with respect to the level of that loop, or
(3) the point represents the first step of the entire loop nest.

In all other cases, $(p_1,\cdots,p_r)$ is inactive with respect to level r. These include the case of alternative (4), where the WHILE condition of w is violated, and the case that the point is not even in the index space. Note, that points for which the WHILE condition holds at level r but not at level $r+1$ are active with respect to level r but not with respect to levels $r+1$ and deeper.

The recursive definition of predicate $active_r$ follows the dependences that are introduced by the WHILE indices. Since our space-time mapping must respect these dependences, we can be sure that, during scanning, the activity of any point p in $\mathcal{TI}$ need not be checked before the activity of its predecessor has been checked. Therefore, we can compute predicate $active_r$ for every point on every tooth of the execution comb, in sequence from the root to the tip, and store the result until it is needed.

Note that $active_r$ at point $(p_1,\cdots,p_r)$ depends on $active_r$ at exactly one other point (see the definition); $active_r$ at point $(p_1,\cdots,p_r)$ is used for the computation of $active_r$ at point $(p_1,\cdots,p_r+1)$ and, if r is not the innermost loop level, also for the computation of $active_{r+1}$ at itself.

Since the polyhedron of a WHILE loop nest contains points that do not model a loop step but only a terminating test, we also require a recognizer, *executed*, for points of $\mathcal{TI}$, or $\mathcal{I}$, that do represent the execution of the loop body.

Definition 2 (Recognizer *executed*).

$$(\forall\ p\ :\ p\in\mathcal{I}\ :\ executed(p) \Leftrightarrow (active_d(p)\wedge condition_d(p)))$$

At this point, we have the machinery for a formal definition of the execution space:

Definition 3 (Execution space). $\mathcal{X} = \{p \in \mathcal{I} \; : \; executed(p)\}$

6 Termination Detection

The most difficult problem in the derivation of target loop code is the determination of the upper bound for the outermost loop (if the source contains at least one inner WHILE loop). The following example demonstrates that even the simplest transformation may cause trouble.

Example 1. Take the following loop nest:

FOR $i := 0$ TO 3 DO
  FOR $j := 0$ WHILE *condition*(i, j) DO
    body

and the space-time mapping the identity $(i, j) \mapsto (p, t)$, where p is space and t is time. (We assume that this transformation respects the dependences of the body. The WHILE dependence is satisfied.) With this mapping, every processor executes the WHILE loop for a fixed value of i. Clearly, each processor can determine the condition for its termination locally, but no processor can detect the termination of any other processor, unless additional measures are taken. Say some statement succeeds the parallel loop nest:

PARFOR $p := 0$ TO 3 DO
  SEQFOR $t := 0$ WHILE *condition*(p, t) DO
      body
statement

How do we know when to start the execution of *statement*?

Here, we chose the asynchronous case (where the time loop is innermost), but the same problem occurs in the synchronous case (where the time loop is outermost): there is no elegant way of expressing the termination condition of the outermost loop. In this example, it is given by $(\forall\, p \; : \; 0 \leq p \leq 3 \; : \; (\exists\, t' \; : \; 0 \leq t' \leq t \; : \; \neg condition(p, t')))$. These quantifications are potentially costly because, in general, there is no bound on their ranges.

6.1 Solution with neigbouring communcations only

In this subsection, we propose a solution of the termination problem that requires only communication between neigbouring processors. The idea is as follows: if carefully selected teeth along dimension r of the execution space inform their neighbours in dimensions $1, \cdots, r$–1 (if they are still executing) of their termination, the maximal coordinates of every dimension of the execution space are propagated.

If we ensure that no tooth terminates before it has been informed of the termination of its neighbours, points that are involved in these communications are partially maximal until the point $(p_1, \cdots, p_d)$ in $\mathcal{S}$ is reached whose coordinates are all maximal, i.e., have the property $(\forall\, p' : p' \in \mathcal{X} : (\forall\, r : 1 \le r \le d : p'_r \le p_r))$. When scanning this point, we can terminate all target loops.

The propagation of the maxima, up to depth r, proceeds by valueless signals. Signal $sig^r_k(p_1, \cdots, p_r)$ starts at point $(p_1, \cdots, p_r)$ at depth r and is sent to the neighbouring tooth in direction k, where k is some outer level with respect to r, i.e., $k < r$.

The main problem is how to establish whether a tooth can terminate immediately when the corresponding WHILE condition is violated or whether it has to wait for some signal first.

Example 2. Consider the three-dimensional comb of Figure 3.

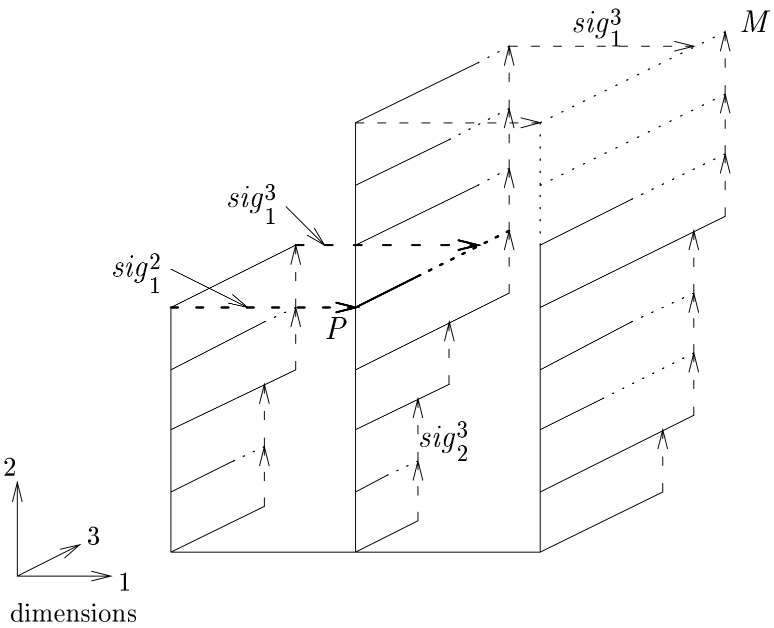

Fig. 3. A three-dimensional comb.

Our scheme is more easily understoood in the synchronous model.

In the figure, the teeth of $\mathcal{X}$ are represented by solid lines. Some teeth are extended by dotted lines, indicating that they are waiting for at least one signal. Points (of $\mathcal{S}$) on dotted lines do not execute the loop body, they only wait for signals. Signals are represented by dashed arrows.

Our aim is to identify the point M whose coordinates are maximal in every dimension. The first coordinate of M is quite easy to determine: it is the value at which the outermost loop terminates.

The second coordinate of M is the maximum of the lengths of all teeth pointing up (in the figure). To determine it, every vertical tooth tells its right neighbour the maximum of its own height and the maximal height left of it. This is the meaning of sig^2_1. If a vertical tooth is ready to terminate but did not yet receive sig^2_1 from its left neighbour, it must wait (except for the leftmost tooth) until this signal is received. (In the figure, the tooth at the right must wait.) Then, it itself sends sig_1^2 on to its right neighbour and terminates. The following formal property holds for all teeth in dimension 2:

$$sig_1^2(p_1,p_2) \Rightarrow (\forall\ p_1',p_2'\ :\ (p_1',p_2')\in\mathcal{X} \wedge p_1'\leq p_1\ :\ p_2'\leq p_2)$$

The determination of the maximal depth of teeth in each vertical plane (p_1 constant) proceeds analogously. Signals sig_2^3 are sent from every (perhaps extended, since waiting) tooth τ along dimension 3 to its upper neighbour of that plane, indicating that the current depth (the length of τ) is maximal for all teeth in dimension 3 to the left and including τ, for fixed p_1. Formally:

$$sig_2^3(p_1,p_2,p_3) \Rightarrow (\forall\ p_2',p_3'\ :\ (p_1,p_2',p_3')\in\mathcal{X} \wedge p_2'\leq p_2\ :\ p_3'\leq p_3)$$

To combine the maxima of all vertical planes, the maximal point of each plane sends a signal sig_1^3 to its right neighbour. Again, it is important that this right neighbour must not terminate before the signal is received. Which teeth must wait? The maximal depth in every vertical plane is reached at the end of the (perhaps extended) vertical tooth that forms the base of this plane. This height was propagated to the right neighbour by sig_1^2. At that height, the maximal depth will also be propagated. Therefore, the tooth, rooted at that point (e.g., P in the figure) which received sig_1^2 and which points into dimension 3 (the thick tooth in the figure), must wait until sig_1^3 is received. Again, formally:

$$sig_1^3(p_1,p_2,p_3) \Rightarrow (\forall\ p_1',p_2',p_3'\ :\ (p_1',p_2',p_3')\in\mathcal{X} \wedge p_1'\leq p_1\ :\ p_2'\leq p_2 \wedge p_3'\leq p_3)$$

The formal properties implied by the signals form a pattern that we call the partial maximality (of a point). M is partially maximal with respect to all dimensions and is, therefore, the maximal point.

In the following, we define partial maximality recursively for an arbitrary number of dimensions. Then, we construct a mechanism that sends signals from partially maximal points to the appropriate destinations.

To include the host of the processor array, we introduce a little hack. We imagine one more dimension, 0, which has extent 2. At position 0 is the polyhedron, and at position 1 is the host. Then, we introduce signals that travel from position 0 to position 1. They are meant to communicate the termination of the target loops to the host.

Definition 4 (Partial maximality m_k^r). $m_k^r(p_1,\cdots,p_r)$ iff for fixed indices at levels 1 to $k-1$ and for points $(p_1',\cdots,p_r')$ below $(p_1,\cdots,p_r)$ at level k ($p_k'\leq p_k$), point $(p_1,\cdots,p_r)$ is maximal in all dimensions $k+1,\cdots,r$. Formally:

$$\begin{array}{l}(\forall\ r\ :0<r\leq d\ :\ (\forall\ (p_1,\cdots,p_r):(p_1,\cdots,p_r)\in\mathcal{X}\ : \\ \quad (\forall\ k\ :\ 0\leq k<r\ :\ m_k^r(p_1,\cdots,p_r)\ = \\ \qquad (\forall\ p_k',\cdots,p_r'\ :\ (p_1,\cdots,p_{k-1},p_k',\cdots,p_r')\in\mathcal{X}\wedge p_k'\leq p_k\ : \\ \qquad\qquad p_{k+1}'\leq p_{k+1},\cdots,p_r'\leq p_r))))\end{array}$$

If $m_k^r(p_1,\cdots,p_r)$, we call point $(p_1,\cdots,p_r)$ *partially maximal* with respect to dimensions $k+1$ to r.

Note that, for $k=0$, the right-hand side of Definition 4 simplifies to:

$$(\forall\, p_1', \cdots, p_r' \,:\, (p_1', \cdots, p_r') \in \mathcal{X} \,:\, p_1' \leq p_1 \wedge \cdots \wedge p_r' \leq p_r)$$

For our communication scheme of m by signals, we need an additional predicate, $w_k^r(p_1, \cdots, p_r)$, which indicates that the tooth that is rooted at $(p_1, \cdots, p_{r-1})$ and extends along dimension r must wait until signal sig_k^r arrives at some point $(p_1, \cdots, p_{r-1}, p_r')$ with $p_r' \geq p_r$. These additional points at which a tooth waits but executes nothing make the difference between $\mathcal{S}$ and $\mathcal{X}$. The lemmata that follow are valid for all points in $\mathcal{S}$.

Definition 5 (*sig* and *w*).

$$\begin{array}{l}
(\forall\, (p_1, \cdots, p_d) \,:\, (p_1, \cdots, p_d) \in \mathcal{I} \,:\, (\forall\, k, r \,:\, 0 \leq k < r \leq d \,:\, sig_k^r(p_1, \cdots, p_r) \;= \\
\quad \neg active_r(p_1, \cdots, p_r) \wedge \\
\quad\; (\forall\, s \,:\, 1 \leq s < r \,:\, \neg w_s^r(p_1, \cdots, p_r)) \wedge (r > k+1 \Rightarrow sig_k^{r-1}(p_1, \cdots, p_{r-1}))\;))
\end{array}$$

$$\begin{array}{lll}
\multicolumn{3}{l}{(\forall\, (p_1, \cdots, p_d) \,:\, (p_1, \cdots, p_d) \in \mathcal{I} \,:\, (\forall\, k, r \,:\, 0 \leq k < r \leq d \,:\, w_k^r(p_1, \cdots, p_r) \;=} \\
\quad \mathbf{if}\; k=0 & \rightarrow \mathit{false} & (1) \\
\quad [\,]\;\; k>0 \wedge p_{k+1} = lb_{k+1} & \rightarrow p_k^{k+1} \neq lb_k & (2) \\
\quad [\,]\;\; r>k+1 \wedge k>0 \wedge p_{k+1} = lb_{k+1} & \rightarrow sig_k^{r-1}(p_1, \cdots, p_k - 1, \cdots, p_{r-1}) & (3) \\
\quad [\,]\;\; r>k+1 \wedge k>0 \wedge p_{k+1} > lb_{k+1} & \rightarrow & (4) \\
\qquad\quad w_k^r(p_1, \cdots, p_{r-1}, p_r - 1) \;\wedge\; \neg sig_k^r(p_1, \cdots, p_k - 1, \cdots, p_{r-1}, p_r - 1) & & \\
\quad \mathbf{fi}\;)) & &
\end{array}$$

These equations can be explained as follows.

sig states that any point of a tooth that is not active and that does not have to wait for any signal, sends signal sig_k^r if either the tooth and the signal lie in a two-dimensional plane (recursion base) or the root $(p_1, \cdots, p_{r-1})$ of the tooth has already sent the signal into the same direction (recursion).

w states the following:

(1) No point has to wait for signals from the host.
(2) In every two-dimensional subspace, every tooth, i.e., at least the first point of it, has to wait for a signal from the tooth immediately preceding it (if any).
(3) In every at least three-dimensional subspace, every tooth parallel to dimension r, i.e., at least the first point of it, has to wait for some signal at depth r from every direction k iff its root at level $r-1$ has received a signal from the same direction k.
(4) Any point of a tooth that is not the first point has to wait for a signal iff its predecessor on the tooth had to wait and did not receive the signal it was waiting for.

The main result of our paper is that signals *sig* correctly propagate property m of local maximality. We state this in two separate lemmata.

Lemma 6 (Local maximum). *A point of some tooth along dimension r that is not active with respect to r and need not wait for a signal is maximal with respect to dimension r. Formally:*

$$\begin{array}{l}(\forall\ (p_1,\cdots,p_r)\ :\ (p_1,\cdots,p_r)\in\mathcal{S}\ :\ \neg active_r(p_1,\cdots,p_r)\wedge\\ \quad(\forall\ s\ :\ 0\leq s<r\ :\ \neg w_s^r(p_1,\cdots,p_r))\Rightarrow(\forall\ p_r'\ :\ (p_1,\cdots,p_{r-1},p_r')\in\mathcal{S}\ :\ p_r'\leq p_r))\end{array}$$

Proof. We prove the inverse implication: $(H\Rightarrow C)\Leftrightarrow(\neg C\Rightarrow\neg H)$.

$$\begin{array}{ll}& \neg(\forall\ p_r'\ :\ (p_1,\cdots,p_{r-1},p_r')\in\mathcal{S}\ :\ p_r'\leq p_r)\\ \Leftrightarrow & \quad\{\text{ predicate calculus }\}\\ & (\exists\ p_r'\ :\ (p_1,\cdots,p_{r-1},p_r')\in\mathcal{S}\ :\ p_r'>p_r)\\ \Rightarrow & \quad\{\text{ if point }(p_1,\cdots,p_{r-1},p_r')\text{ is scanned, it must be active or waiting }\}\\ & (\exists\ p_r'\ :\ (p_1,\cdots,p_{r-1},p_r')\in\mathcal{S}\ :\\ & \quad(active_r(p_1,\cdots,p_{r-1},p_r')\ \vee\ (\exists\ s\ :\ 1\leq s<r\ :\ w_s^r(p_1,\cdots,p_r))))\\ \Leftrightarrow & \quad\{\text{ predicate calculus }\}\\ & \neg(\forall\ (p_1,\cdots,p_r)\ :\ (p_1,\cdots,p_r)\in\mathcal{S}\ :\\ & \quad\neg active_r(p_1,\cdots,p_r)\ \wedge\ (\forall\ s\ :\ 1\leq s<r\ :\ \neg w_s^r(p_1,\cdots,p_r)))\end{array}$$

In the following main lemma of the paper, $\mathcal{SUB}_k^r(p_1,\cdots,p_{k-1})$ is the subspace of $\mathcal{S}$ in dimensions k to r $(1\leq k<r\leq d)$ and at fixed coordinates $(p_1,\cdots,p_{k-1})$.

Lemma 7 (sig implements m). $sig_k^r(p_1,\cdots,p_r)\Rightarrow m_k^r(p_1,\cdots,p_r)$

Proof. By induction on the dimension of $\mathcal{SUB}_k^r(p_1,\cdots,p_{k-1})$.

- Induction base: $r=k+1$
  For $sig_k^{k+1}(p_1,\cdots,p_{k+1})\Rightarrow m_k^{k+1}(p_1,\cdots,p_{k+1})$ we must distinguish two cases, since p_0 does not exist.
  - First case: $k=0$

$$\begin{array}{ll}& sig_0^1(p_1)\\ \Leftrightarrow & \quad\{\text{ definition of } sig \text{ with } k=0,\ r=1\ \}\\ & \neg active_1(p_1)\wedge\neg w_0^1(p_1)\\ \Rightarrow & \quad\{\text{ Lemma 6 }\}\\ & (\forall\ p_1'\ :\ p_1'\in\mathcal{S}\ :\ p_1'\leq p_1)\\ \Leftrightarrow & \quad\{\text{ definition of } m_0^1\ \}\\ & m_0^1(p_1)\end{array}$$

  - Second case: $k>0$
    We prove $sig_k^{k+1}(p_1,\cdots,p_{k+1})\Rightarrow m_k^{k+1}(p_1,\cdots,p_{k+1})$ by induction on p_k.

* Induction base: $p_k = lb_k$

$$
\begin{array}{ll}
& sig_k^{k+1}(p_1,\cdots,p_{k+1}) \\
\Leftrightarrow & \{ \text{ definition of } sig \text{ with } p_k = lb_k \\
& \quad \text{and simplification for } r = k+1 \ \} \\
& \neg active_{k+1}(p_1,\cdots,p_{k+1}) \ \wedge \\
& \quad (\forall\ s\ :\ 1 \le s < k+1\ :\ \neg w_s^{k+1}(p_1,\cdots,p_{k+1})) \\
\Rightarrow & \{ \text{ Lemma 6 } \} \\
& (\forall\ p'_{k+1}\ :\ (p_1,\cdots,p_k,p'_{k+1}) \in \mathcal{S}\ :\ p'_{k+1} \le p_{k+1}) \\
\Leftrightarrow & \{ \ lb_k \text{ is the smallest } p_k; \\
& \quad \text{introduction of a new dummy with empty range } \} \\
& (\forall\ p'_k, p'_{k+1}\ :\ (p_1,\cdots,p_{k-1},p'_k,p'_{k+1}) \in \mathcal{S}\ \wedge\ p'_k \le p_k\ : \\
& \qquad p'_{k+1} \le p_{k+1}) \\
\Leftrightarrow & \{ \text{ definition of } m \ \} \\
& m_k^{k+1}(p_1,\cdots,p_{k+1})
\end{array}
$$

* Induction step: $p_k - 1 \to p_k$ $(p_k > lb_k)$

$$
\begin{array}{ll}
& sig_k^{k+1}(p_1,\cdots,p_{k+1}) \\
\Leftrightarrow & \{ \text{ definition of } sig \text{ with } p_k > lb_k \\
& \quad \text{and simplification for } r = k+1 \ \} \\
& \neg active_{k+1}(p_1,\cdots,p_{k+1}) \ \wedge \\
& \quad (\forall\ s\ :\ 1 \le s < k+1\ :\ \neg w_s^{k+1}(p_1,\cdots,p_{k+1})) \\
\Rightarrow & \{ \text{ duplication of } \neg w_k^{k+1} \ \} \\
& \neg active_{k+1}(p_1,\cdots,p_{k+1}) \ \wedge \\
& \quad (\forall\ s\ :\ 1 \le s < k+1\ :\ \neg w_s^{k+1}(p_1,\cdots,p_{k+1})) \ \wedge \\
& \quad \neg w_k^{k+1}(p_1,\cdots,p_{k+1}) \\
\Rightarrow & \{ \text{ Lemma 6 } \} \\
& (\forall\ p'_{k+1}\ :\ (p_1,\cdots,p_k,p'_{k+1}) \in \mathcal{S}\ :\ p'_{k+1} \le p_{k+1}) \ \wedge \\
& \quad \neg w_k^{k+1}(p_1,\cdots,p_{k+1}) \\
\Rightarrow & \{ \text{ by structural induction on the definition of } w \ \} \\
& (\forall\ p'_{k+1}\ :\ (p_1,\cdots,p_k,p'_{k+1}) \in \mathcal{S}\ :\ p'_{k+1} \le p_{k+1}) \ \wedge\ (p_k = lb_k\ \vee \\
& \quad (\exists\ \widehat{p_{k+1}}\ :\ \widehat{p_{k+1}} \le p_{k+1}\ :\ sig_k^{k+1}(p_1,\cdots,p_k-1,\widehat{p_{k+1}}))) \\
\Rightarrow & \{ \ p_k \ne lb_k \text{ (induction step) } \} \\
& (\forall\ p'_{k+1}\ :\ (p_1,\cdots,p_k,p'_{k+1}) \in \mathcal{S}\ :\ p'_{k+1} \le p_{k+1}) \ \wedge \\
& \quad (\exists\ \widehat{p_{k+1}}\ :\ \widehat{p_{k+1}} \le p_{k+1}\ :\ sig_k^{k+1}(p_1,\cdots,p_k-1,\widehat{p_{k+1}})) \\
\Rightarrow & \{ \text{ induction hypothesis for } sig_k^{k+1}(p_1,\cdots,p_k-1,\widehat{p_{k+1}}) \ \} \\
& (\forall\ p'_{k+1}\ :\ (p_1,\cdots,p_k,p'_{k+1}) \in \mathcal{S}\ :\ p'_{k+1} \le p_{k+1}) \ \wedge \\
& \quad (\exists\ \widehat{p_{k+1}}\ :\ \widehat{p_{k+1}} \le p_{k+1}\ :\ m_k^{k+1}(p_1,\cdots,p_k-1,\widehat{p_{k+1}})) \\
\Rightarrow & \{ \ \widehat{p_{k+1}} \le p_{k+1}; \text{ definition of } m) \ \} \\
& (\forall\ p'_{k+1}\ :\ (p_1,\cdots,p_k,p'_{k+1}) \in \mathcal{S}\ :\ p'_{k+1} \le p_{k+1}) \ \wedge \\
& \quad m_k^{k+1}(p_1,\cdots,p_k-1,p_{k+1}) \\
\Leftrightarrow & \{ \text{ definition of } m \ \} \\
& (\forall\ p'_{k+1}\ :\ (p_1,\cdots,p_k,p'_{k+1}) \in \mathcal{S}\ :\ p'_{k+1} \le p_{k+1}) \ \wedge\ (\forall\ p'_k, p'_{k+1}\ : \\
& \quad (p_1,\cdots,p_{k-1},p'_k,p'_{k+1}) \in \mathcal{S}\ \wedge\ p'_k \le p_k - 1\ :\ p'_{k+1} \le p_{k+1}) \\
\Leftrightarrow & \{ \text{ combination of two quantification ranges } \} \\
& (\forall\ p'_k, p'_{k+1}\ :\ (p_1,\cdots,p_{k-1},p'_k,p'_{k+1}) \in \mathcal{S}\ \wedge\ p'_k \le p_k\ :\ p'_{k+1} \le p_{k+1}) \\
\Leftrightarrow & \{ \text{ definition of } m \ \} \\
& m_k^{k+1}(p_1,\cdots,p_{k+1})
\end{array}
$$

– Induction step: $k \to k-1$ $(r-k > 1)$

$$
\begin{array}{ll}
& sig^r_{k-1}(p_1,\cdots,p_r) \\
\Leftrightarrow & \{ \text{ definition of } sig \text{ with } r > k+1 \ \} \\
& \neg active_r(p_1,\cdots,p_r) \ \wedge \ (\forall\ s : 1 \leq s < r : \neg w^r_s(p_1,\cdots,p_r)) \ \wedge \\
& \quad (sig^{r-1}_{k-1}(p_1,\cdots,p_{r-1})) \\
\Rightarrow & \{ \text{ Lemma 6 } \} \\
& (\forall\ p'_r : (p_1,\cdots,p_{r-1},p'_r) \in \mathcal{S} : p'_r \leq p_r) \ \wedge \ sig^{r-1}_{k-1}(p_1,\cdots,p_{r-1}) \\
\Rightarrow & \{ \text{ induction hypothesis for } sig^{r-1}_{k-1} \ \} \\
& (\forall\ p'_r : (p_1,\cdots,p_{r-1},p'_r) \in \mathcal{S} : p'_r \leq p_r) \ \wedge \ m^{r-1}_{k-1}(p_1,\cdots,p_{r-1}) \\
\Leftrightarrow & \{ \text{ definition of } m \ \} \\
& (\forall\ p'_r : (p_1,\cdots,p_{r-1},p'_r) \in \mathcal{S} : p'_r \leq p_r) \ \wedge \\
& \quad (\forall\ p'_{k-1},\cdots,p'_{r-1} : (p_1,\cdots,p_{k-2},p'_{k-1},\cdots,p'_{r-1}) \in \mathcal{S} \ \wedge \ p'_{k-1} \leq p_{k-1} : \\
& \quad p'_k \leq p_k \wedge \cdots \wedge p'_{r-1} \leq p_{r-1}) \\
\Rightarrow & \{ \text{ combination of two quantification ranges } \} \\
& (\forall\ p'_k,\cdots,p'_r : (p_1,\cdots,p_{k-2},p'_{k-1},\cdots,p'_r) \in \mathcal{S} \ \wedge \ p'_{k-1} \leq p_{k-1} : \\
& \quad p'_k \leq p_k \wedge \cdots \wedge p'_r \leq p_r) \\
\Leftrightarrow & \{ \text{ definition of } m \ \} \\
& m^r_{k-1}(p_1,\cdots,p_r)
\end{array}
$$

Our aim has been to identify the point with maximal coordinates in all dimensions of $\mathcal{S}$. The scanning of this point indicates the termination of the entire target loop nest. We have constructed a signalling scheme in which this point sends signal sig^d_0 to the host.

One simple optimization can be made immediately: signals need not be sent to points p along a tooth that has terminated, i.e., to points that neither are active nor wait for a signal. Thus, $sig^r_k(p_1,\cdots,p_r)$ implies that, for any k' with $k' < k$, $sig^r_{k'}(p_1,\cdots,p_r)$ need not be sent. In Example 2, we have already omitted these signals, but they are, of course, specified in Definition 5.

Future work will reduce the size of $\mathcal{S}$ by extracting tighter individual termination conditions for each target loop instead of one common termination condition for all loops.

7 Conclusions

We are considering a loop nest that contains WHILE loops and a space-time mapping $\mathcal{T}$ that parallelizes this nest. Iff $\mathcal{T}$ is unscanable in the technical sense of [4], we cannot scan the image $\mathcal{TX}$ of the execution space $\mathcal{X}$ under $\mathcal{T}$ precisely.

For this case, we have proposed a signalling scheme here with which a target loop nest can scan a finite superset $\mathcal{TS}$ of $\mathcal{TX}$ and distinguish the points of $\mathcal{TX}$ from the points of $\mathcal{TS} \backslash \mathcal{TX}$. The determination of the upper bounds for the target loops that scan $\mathcal{S}$ proceeds by neighbouring communications only; after the mapping of $\mathcal{S}$ to $\mathcal{TS}$ communication distances may exceed 1 but are still constant, i.e., independent of the problem size.

The price we have to pay for unscanable space-time mappings is that the necessary scanning of a superset of $\mathcal{TX}$ increases the execution time and/or the amount of processors needed. A more detailed analysis is left to future work.

One might wonder whether unscanable space-time mappings are at all useful at such a price. It turns out that they are definitely interesting for synchronous parallelism because, in this case, there are no scanable space-time mappings for a loop nest with more than one WHILE loop.

Acknowledgements

The first author is grateful to Max Geigl for a careful reading and discussions on multi-dimensional spaces that led to the definition of *sig*.

References

1. M. Barnett and C. Lengauer. Unimodularity and the parallelization of loops. *Parallel Processing Letters*, 2(2–3):273–281, 1992.
2. J.-F. Collard. Code generation in automatic parallelizers. Technical Report 93-21, LIP, Ecole Nationale Superieure de Lyon, July 1993.
3. E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics*. Texts and Monographs in Computer Science. Springer-Verlag, 1990.
4. M. Griebl and C. Lengauer. On the space-time mapping of WHILE-loops. *Parallel Processing Letters*, 1994. To appear. Also: Technical Report MIP-9304, Fakultät für Mathematik und Informatik, Universität Passau, July 1993.
5. R. M. Karp, R. E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *J. ACM*, 14(3):563–590, July 1967.
6. C. Lengauer. Loop parallelization in the polytope model. In E. Best, editor, *CONCUR'93*, Lecture Notes in Computer Science 715, pages 398–416. Springer-Verlag, 1993.
7. P. Quinton. The systematic design of systolic arrays. In F. F. Soulié, Y. Robert, and M. Tchuente, editors, *Automata Networks in Computer Science*, chapter 9, pages 229–260. Manchester University Press, 1987. Also: Technical Reports 193 and 216, IRISA (INRIA-Rennes), 1983.
8. S. K. Rao and T. Kailath. Regular iterative algorithms and their implementations on processor arrays. *Proc. IEEE*, 76(3):259–282, March 1988.
9. J. Xue. An algorithm to automate non-unimodular transformations of loop nests. In *Proc. 5th IEEE Symp. on Parallel and Distributed Processing (SPDP '93)*, pages 512–519. IEEE Computer Society Press, December 1993.