12

OPERA : A Toolbox For Loop Parallelization

Vincent Loechner and Catherine Mongenet
Université Louis Pasteur de Strasbourg, Laboratoire ICPS
Pôle API, Boulevard Sébastien Brant, 67400 Illkirch, France
Phone : (33) 88 65 50 37. Fax : (33) 88 65 50 61
email : {loechner,mongenet}@icps.u-strasbg.fr

Abstract

This paper presents the mathematical notions for the parallelization of DO-Loops used in the tool OPERA currently under development in our team. It aims at giving the user an environment to parallelize problems described by systems of parameterized affine recurrence equations which formalize single-assignment loop nests. The parallelization technique used in OPERA is based on a classical linear space×time transformation. Its objectives are to visualize the affine dependences of a problem and to propose a set of different parallel solutions depending on various architectural constraints.

Keywords

loop parallelization, parameterized affine recurrence equations, parameterized domains.

1 INTRODUCTION

Scientific computing as well as many other application domains, has always required large amounts of memory and hours of CPU time. An answer to achieve such high-performance computing can be found in the use of parallelism. Over the past decades many improvements have been made in the evolution of parallel machines, parallel languages and parallel environments.

Scientific programs are characterized by the well-known *90-10 rule* : 90% of the CPU time is spent in 10% of the code, that is to say the DO-Loops. Our objective is to develop techniques and tools to efficiently and automatically parallelize such loops and to show how various architectural constraints can be taken into account in order to synthesize different parallel solutions. These parallelization techniques are founded on sound and formal mathematical notions in order to guaranty the correctness of the synthesized parallel solutions and offer a good framework for their performance evaluation. Our work is based on the use of Systems of Recurrence Equations which are a formal description of single assignment DO-Loops. It finds its foundations on the many works on systolic array synthesis and their latest extensions to deal with affine recurrences. These recurrence equations are defined over convex integer polyhedra. In order to find a parallel solution,

I. Jelly et al. (eds.), *Software Engineering for Parallel and Distributed Systems*

we apply on such polyhedra a linear space×time transformation. Therefore our formal methodology is based on the definition of such polyhedra and their manipulation using linear algebra.

This paper presents the tool OPERA* currently under development in our team. This prototype aims at giving the user an environment to express and solve problems defined by systems of parameterized affine recurrence equations (PARE in the following). Parameterized equations formalize loop nests with loop bounds depending on parameters, i.e. whose values are unknown at compile time. One of the interests of OPERA is to help the user to analyze his problems through the visualization of their geometrical properties. It determines several different parallel solutions and their main characteristics are automatically computed as functions of the parameters : the processor count, the latency, and the number and the nature of the communications (local neighbor-to-neighbor or broadcast communications).

The paper is organized as follows. Section 2 shortly presents the method of PARE parallelization. An overview of OPERA is given in section 3. Section 4 illustrates its use with the Gaussian Elimination algorithm. Finally section 5, as a conclusion, compares OPERA with other existing tools and describes its current developments.

2 PARALLELIZATION OF PARAMETERIZED AFFINE RECURRENCE EQUATIONS

Among the many works concerned with DO-Loops parallelization, we distinguish two approaches. The first one is based on the direct parallelization of DO-Loop programs using various dependence tests (Zima and Chapman (1990), Pugh (1992), Banerjee (1993)). The second approach consists in first transforming the DO-Loops into single-assignment loop nests (Cytron et al. (1991)), which are free of dependences due to the lexicographical order of loop indices. These single assignment programs are then parallelized using synthesis techniques developped by the systolic community. We focus on this second step and deal with single assignment loops defined by systems of parameterized affine recurrence equations (PARE) defined over bounded convex polyhedra.

Example. Gaussian Elimination. In order to solve a linear system of equations $Ax = b$ where $A = (a_{ij})$ is an $n \times n$ matrix and b an n vector, the Gaussian Elimination triangulates the extended matrix $\mathtt{A} = [A \mid b]$. It is defined by the following loop nest :

```
DO k = 1 TO n-1
   DO j = k+1 TO n+1
      DO i = k+1 TO n
         A[i,j] = A[i,j] - A[i,k] * A[k,j] / A[k,k]
      DONE
   DONE
DONE
```

This loop nest can be transformed into the following equivalent single-assignment loop. The corresponding system of PARE given as input to OPERA is given figure 1.

*OPERA is a french acronym for "Toolbox for the Parallelization of Systems of Affine Recurrence Equations".

```
### Gaussian Elimination algorithm ###
#dimension of the problem
   DIMENSION 3
#parameter used in the visualization
   PARAM n=5
#input
   M[i,j] = DATA     { 1 <=i<= n, 1 <=j<= n+1 }
#output
   OUTPUT = A[i,j,k] { 1 <=i<= n , i <=j<= n+1 , k = i-1 }
#initialization equation
   A[i,j,k] = M[i,j] { 1 <=i<= n, 1 <=j<= n+1, k = 0 }
#computation equation
   A[i,j,k] = A[i,j,k-1] - A[i,k,k-1] * A[k,j,k-1] / A[k,k,k-1]
     { k+1 <=i<= n , k+1 <=j<= n+1 , 1 <=k<= n-1 }
```

Figure 1 Input file for the Gaussian Elimination algorithm.

```
DO k = 1 TO n-1
   DO j = k+1 TO n+1
      DO i = k+1 TO n
         A[i,j,k] = A[i,j,k-1] - A[i,k,k-1] * A[k,j,k-1] / A[k,k,k-1]
      DONE
   DONE
DONE
```

A parameterized affine recurrence equation has the following form:

$$X[z] = f\big(\cdots, Y[g(z)], \cdots\big) \qquad z \in D_p \tag{E}$$

where X and Y are array variables, f is any computation function with O(1) complexity, $p \in \mathbf{Z}^m$ is the vector defining the m parameters of the problem, $D_p \subset \mathbf{Z}^d$ is a convex bounded polyhedron associated with equation (E) called the *index domain* or the *iteration space*. It is defined by a set of linear constraints expressed by a system of linear inequalities $D_p = \{z \in \mathbf{Z}^d \mid P \cdot z \leq Q \cdot p + q\}$ where P is a $c \times d$ integer matrix, Q is a $c \times m$ integer matrix and q is a c integer vector. The *index mapping* g is an affine function with constant coefficients from $\mathbf{Z}^d$ to $\mathbf{Z}^{d'}$ $(d' \leq d)$ of the form: $g(z) = R \cdot z + r$ where R is a $d' \times d$ integer matrix called the *index matrix* and r is an integer vector of size d'.

2.1 Dependence modeling and space×time mapping

In order to determine parallel solutions from such a PARE, the dependences have first to be modeled. This relies on the notion of *utilization set*. When considering a variable $Y[z_0]$, we call utilization set and we denote it by $Util_E(Y, z_0)$ the set of all the points of the domain using $Y(z_0)$ in their calculation. It is defined by :

$$Util_E(Y, z_0) = \{z \in D_p \mid g(z) = z_0\}.$$

Mongenet et al. (1991) show that this is a convex bounded polyhedron whose linear space is $Ker(R)$ where R is the index matrix. The dimension of the utilization set is therefore

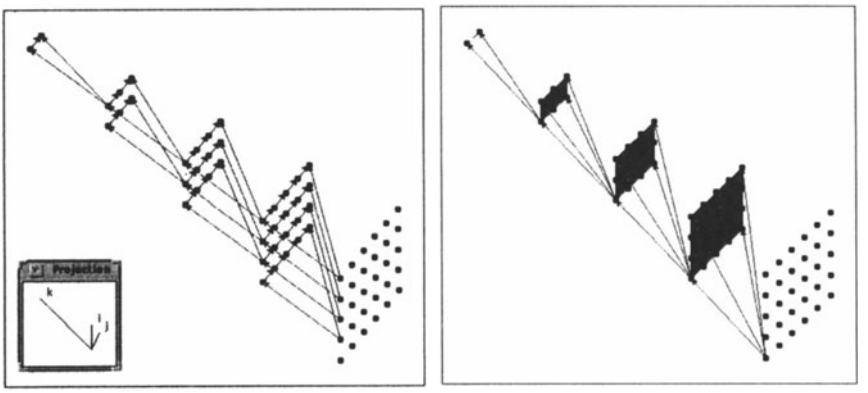

Figure 2 Domain and utilization sets $Util_2$ and $Util_4$.

$d - Rank(R)$ where d is the dimension of the domain. The basis vectors of $Ker(R)$ characterize the utilization set and are called the *utilization vectors.* They are denoted by $u_{E,Y,i}$ $(i = 1, \ldots, d - Rank(R))$. Notice that when R is a full row rank matrix, the utilization set is reduced to a single point and does not require any utilization vector.

The causal dependence associated with a variable Y is classically expressed by *dependence vectors* $d_{Y,g(z)} = z - g(z)$. Mongenet et al. (1991) show that the dependence vectors related to a variable $Y(z_0)$ define a *cone* characterized by its finite set of *extremal vectors.* These extremal vectors are denoted by $d_{Y,z_0,ex_i}, i \in \mathbf{N}$.

All points $z_0 = g(z)$ which are origin of a dependence vector d_{Y,z_0} form a set. This set is called the *emission set* and defined by :

$$Emit_E(Y) = \{z_0 \in D_p \mid \exists z \in D_p \text{ such that } g(z) = z_0\} = g(D).$$

Example. The Gaussian Elimination. In the computation equation there are four references to elements of variable A. We refer to these different references as respectively the first, second, third and fourth argument of the computation function and we index the corresponding index functions, index matrices, dependence vectors, utilization sets and utilization vectors accordingly.

The first argument corresponds to a uniform dependence characterized by index matrix $R_1 = Id$. The utilization set of any variable $A[z_0]$ has dimension 0 and the dependence is therefore characterized by a constant dependence vector $d_1 = (0, 0, 1)$.

The second and third arguments are both characterized by one-dimensional utilization sets since $Rank(R_2) = Rank(R_3) = 2$. The corresponding utilization and dependence vectors are respectively :

$u_2 = (0, 1, 0)$ $\quad d_2 = (0, j - j_0, 1)$ with $1 \leq j - j_0 \leq n - j_0 + 1$

$u_3 = (1, 0, 0)$ $\quad d_3 = (i - i_0, 0, 1)$ with $1 \leq i - i_0 \leq n - i_0$

The fourth argument induces two-dimensional utilization sets since $Rank(R_4) = 1$, and we have : $u_{4,1} = (1, 0, 0)$, $u_{4,2} = (0, 1, 0)$ and $d_4 = (i - i_0, j - j_0, 1)$ with $1 \leq i - i_0 \leq n - i_0$ and $1 \leq j - j_0 \leq n - j_0 + 1$.

The one-dimensional utilization sets $Util_2$ and the two-dimensional utilization sets $Util_4$

are presented in figure 2. The dotted vectors associated with each of these sets correspond to the extremal dependence vectors. The plain vectors correspond to the utilization vectors. □

The interest of considering such PAREs, i.e. single assignment programs, is the simple dependence modeling they induce. Hence, the dependence graphs have their vertices aligned on integer points of convex polytopes. The parallelization applies then on a dependence graph a space×time transformation. The transformations classically used are linear. They consist in particular in :

- An affine **schedule function** of the form $t : D_p \subset \mathbf{Z}^d \longrightarrow \mathbf{N}$ with $t(z) = \lambda \cdot z + \alpha$. It defines successive fronts on the domain which are the set of hyperlanes orthogonal to the *schedule vector* λ. The causal constraints are classically expressed by a system of constraints on the schedule function and the dependence vectors. For a system of PARE it is expressed by $\lambda \cdot d_{ex} > 0$ for all the extremal dependence vectors d_{ex} associated with the problem. Among the extremal vectors we call *minimal extremal vectors* the extremal vectors whose schedule component $\lambda \cdot d_{ex}$ is minimal. We denote them by d_{min}.

- A linear **allocation function** *alloc* : $\mathbf{Z}^d \longrightarrow \mathbf{Z}^{d-1}$ projects the domain along a unimodular vector denoted by $\xi \in \mathbf{Z}^d$ and called the *allocation* or *projection direction*. It results in a set of virtual processors that can be described by a DOALL loop program as given hereunder. All the points of D belonging to a given line directed by ξ define a segment called an *allocation segment* and are projected and executed in the same virtual processor, i.e. executed in the same iteration of the corresponding DOALL loop. Recall that the schedule and allocation functions must garanty that a virtual processor executes at most one computation at any given time step. This requires the condition $\lambda \cdot \xi \neq 0$ to hold.

Once a schedule and an allocation functions have been determined, the synthesis of a parallel solution consists in applying the corresponding linear mapping to the original specification of the problem. It transforms the problem expressed in the original domain D_p into an equivalent problem defined in a space×time domain. A virtual parallel solution can therefore be given in terms of loops characterized by one sequential loop defining the time and a set of DOALL loops characterizing the processors.

Example. For the Gaussian Elimination, a parallel solution can be synthesized with the schedule $\lambda = (1,1,1)$ and $\xi = (1,0,0)$ as allocation direction. We get the following virtual code usually refered as a SPMD code :

```
DOALL P(x,y) with 1 ≤ y ≤ n-1 , y+1 ≤ x ≤ n+1
  DO t = x+2y-3 TO x+y+n-4
     A[t,x,y] = A[t-1,x,y-1] - A[t-1,y,y-1]×A[x+2y-1,x,y-1]/A[3y-1,y,y-1]
  DONE
DONE
```

This loop can not be transformed into a non single-assignment loop by removing the inner index corresponding to the time variable. This is due to the use of `A[x+2y-1,x,y-1]` and `A[3y-1,y,y-1]` for the computation of `A[x,y]` at time `t` on processor `P(x,y)`. Since the source depends on the processor's reference and not on the time index `t`, the first index

of A can not be removed in `A[x+2y-1,x,y-1]` and `A[3y-1,y,y-1]`. These references must be defined explicitly by communications and local storage of the data. □

Such a virtual code does not explicit the communications between the virtual processors. Using OPERA, in particular its dependence modeling, the communications can be synthesized in order to get a parallel code with explicit communication primitives, as seen figure 5 for the explicit code corresponding to the above single assignment program of the Gaussian Elimination. We show in the next section how these communication operations can be automatically determined from the projection of the utilization and minimal extremal vectors.

2.2 Communication synthesis

OPERA analyzes, for a given problem, how the choice of a particular linear space×time mapping influences the communications between the virtual processors of the parallel solution : the number of communications and their nature. We particularly focus on two types of communications : the local neighbor-to-neighbor communications and the broadcast communications. The results presented hereunder are developped by Mongenet (1995).

When dealing with communication synthesis, a user may focus on various constraints such as the nature or the number of the communication primitives. OPERA may help the user to deal with these two questions :

- **Broadcast utilization.** Depending on whether or not we are interested in parallel solutions with broadcast, we must add an appropriate constraint to the set of causal constraints in order to get a schedule. If we want the solution to be free of any broadcast, i.e. no variable can be used at a given time step by several processors of the DOALL loop, then we add the constraint $\lambda \cdot u_Y \neq 0$ for all the utilization vectors u_Y of the problem. Conversely if we want to take advantage of some broadcast primitives, then the constraint $\lambda \cdot u_Y = 0$ should hold for as many utilization vectors u_Y as possible.

- **Communication minimization.** In order to minimize the number of communications, one must try to localize as many arguments as possible. This can be achieved by choosing as allocation direction a vector projecting a utilization set on a minimal number of virtual processors of the DOALL loop. For a one-dimensional utilization set characterized by a unique utilization vector u the communication is optimally reduced with allocation direction $\xi = u$. Hence all the corresponding utilization points are projected on the same virtual *utilization processor* and this dependence requires at most one communication : from the virtual emission processor (the processor where is implemented the corresponding emission point) to the virtual utilization processor. If the emission point belongs to the linear space containing the utilization set, then no communication is required. For a two-dimensional utilization set characterized by two utilization vectors u_1 and u_2, one must choose an allocation direction parallel to the plane defined by u_1 and u_2 in order to project the plane on a single line of virtual processors.

3 AN OVERVIEW OF OPERA

3.1 Structure of OPERA

OPERA runs on *X/XView* environment. It is compiled with the *gcc* compiler, and *flex* and *yacc* for the parser. Polyhedra are represented using the Polyhedral Library developped at IRISA by Wilde (1993).

OPERA takes as input a system of parameterized affine recurrence equations defining the problem. The syntax of such a specification is similar to the ALPHA language proposed by Le Verge et al. (1991) as it was shown in figure 1 for the Gaussian Elimination. Recall that OPERA deals with parameterized equations, and therefore all the computations are realized using the parameters without instanciating them.

OPERA visualizes a synthesized solution by a grid of virtual processors (which can be described by a DOALL loop) and by the communication links between these processors. It is composed by the following main modules.

- the dependence modeling module determines the parameterized utilization sets as well as the extremal dependence vectors.
- the schedule determination module solves the system of linear inequalities $\lambda \cdot d > 0$. Depending on the user choice regarding broadcast utilization, the extra constraints $\lambda \cdot u \neq 0$ may be added or not.
- the allocation determination module. This module currently requires the user to choose an allocation direction. This will eventually be replaced by the automatic determination of several efficient allocation directions, such as the directions minimizing the number of communications (cf. Mongenet (1995)).

These different modules implement algebraic operations on convex parameterized polyhedra. In order to help the user to better apprehend the geometrical nature of his problem, each of these modules is coupled with a visualization module as shown in figure 3.

3.2 Algebraic tools on parameterized polyhedra

OPERA mainly manipulates convex parameterized polyhedra to represent the domains, the utilization sets, the fronts. It uses the two dual representations of a polyhedron :

- the *implicit representation* : the polyhedron is defined by a set of linear constraints.
- the *Minkovsky representation* also called the *parametric representation* by Schrijver (1986) : the polyhedron is characterized by a set of lines, rays and vertices.

To go from one representation to the other, we use the algorithm implemented in the Polyhedral Library. It is based on the Chernikova algorithm, successively improved by Fernández and Quinton in 1988 and Le Verge in 1992. Its complexity is $O(c^{\lfloor \frac{d}{2} \rfloor})$ where c is the number of constraints and d the dimension of the polyhedron.

The parameterized domains manipulated by OPERA are defined by the following implicit form :

$$D_p = \{z \in \mathbf{Z}^d \mid P \cdot z \leq Q \cdot p + q\}$$

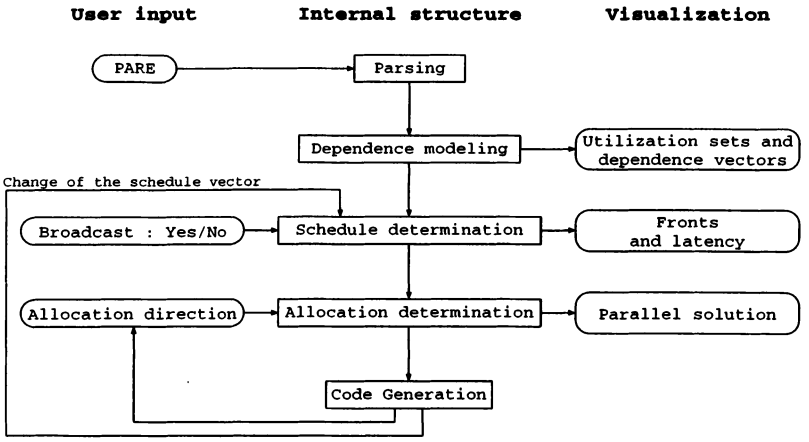

Figure 3 Overview of OPERA.

where P is a $k \times d$ integer matrix, Q is a $k \times m$ integer matrix, q is a k integer vector, or equivalently by the Minkovsky form :

$$D_p = \{z \in \mathbf{Z}^d \mid z = L \cdot \rho + R \cdot \mu + V_p \cdot \nu \mid \quad \mu, \nu \geq 0, \quad \sum \nu = 1\}$$

where L is the matrix whose columns define the lines of the polyhedron, while the columns of R are the rays and the columns of V_p are the vertices. ρ, μ, ν are free valued integer vectors.

In order to be able to use the Polyhedral Library and its revisited Chernikova algorithm, OPERA represents the parameterized polyhedra in the combined data and parameter space as follows :

$$D' = \{\begin{pmatrix} x \\ p \end{pmatrix} \in \mathbf{Z}^{d+m} \mid P' . \begin{pmatrix} x \\ p \end{pmatrix} \leq q\}$$

where $P' = [P \mid -Q]$.

Using the Minkovsky representation of D', we can find its faces of geometric dimension m called m-faces †. The vertices of the parameterized polyhedron D_p correspond to m-faces of its associated polyhedron D' in the combined space. To compute the parameterized vertex $v(p)$ corresponding to a given m-face, the algorithm described by Loechner and Wilde (1995) computes two affine transformations related to this m-face :

- the first one projects any point of the m-face in the data space $\mathbf{Z}^d$. Let $Proj_d$ be its associated matrix.
- the second one projects any point of the m-face in the parameter space $\mathbf{Z}^m$. Let $Proj_p$ be its associated square matrix. If matrix $Proj_p$ is singular then this m-face does not correspond to a vertex of D_p (proof given by Loechner and Wilde (1995)).

† More generally for any convex polyhedron, the faces of geometric dimension 0 define its vertices, the faces of dimension 1 define its edges, ...

Using these two transformations, each vertex $v(p)$ is defined by :

- a domain E corresponding to the projection of the m-face on the parameter space. It defines the restriction on the values of the parameters for which the corresponding vertex does exist.
- a $d \times m$ matrix $S = Proj_d \cdot Proj_p^{-1}$. The vertex $v(p)$ is only defined for $p \in E$ and its coordinates are $S \cdot p$. This information is used in OPERA to compute the extremal dependence vectors, as well as the latency, as function of the parameters.

4 THE GAUSSIAN ELIMINATION ALGORITHM

From a system of PARE, OPERA automatically computes the parameterized domains and visualizes them for a specified value of the parameters. It also performs the dependence modeling and exhibits it both algebraically and geometrically.

A schedule vector must satisfy the causal constraints $\lambda \cdot d_{ex} > 0$ for all the extremal vectors of the problem. Moreover to be free of broadcast, the constraint $\lambda \cdot u \neq 0$ should hold for all the utilization vectors of the problem. For the Gaussian Elimination $\lambda = (1,1,1)$ results in a parallel solution using only local communications. Moreover one can show that, in this case, it is the optimal affine schedule with a latency equal to $\tau = 3n - 4$.

If we allow solutions with one-dimensional broadcast communications, we may symmetrically broadcast the second or third argument, which correspond both to one-dimensional utilization sets. Either choice will result in a one-dimensional broadcast of the fourth argument because the corresponding utilization sets are two-dimensional and directed by $u_{4,1} = u_3$ and $u_{4,2} = u_2$. These two symmetrical solutions correspond to $\lambda = (0,1,1)$ and $\lambda = (1,0,1)$.

One may implement even more broadcast by choosing a schedule vector orthogonal to both u_2 and u_3, i.e. $\lambda = (0,0,1)$. This choice results not only in the broadcast of the second or third argument, but also in the full broadcast of the fourth one, due to the equality of the utilization vectors mentioned above. Notice that this schedule vector may be deduced from the observation of figure 2. Hence to get a 2-dimensional broadcast, one must schedule all the points of the grey planes in the rightmost window at the same instant and therefore choose a schedule vector orthogonal to these planes.

Let us now show for the schedule vector $\lambda = (1,1,1)$ how the parallel code with explicit communication primitives is synthesized. The number of communications can be reduced by localizing for example the third argument. In this case we have to choose $\xi = u_3 = (1,0,0)$ as allocation direction [‡]. Moreover since $u_{4,1} = (1,0,0)$ the fourth argument characterized by two-dimensional utilization sets is also localized on one dimension. The corresponding allocation function is $alloc(i,j,k) = (j,k)$.

The leftmost window in figure 4 shows how the utilization sets $Util_1$ and $Util_3$ are projected. The uniform dependence associated with $Util_1$ results in a local communication along the k-axis. Since the dependence related to $Util_3$ is localized, the only communication is generated by $d3_{min}$. The projection of $d3_{min}$ is similar to the one of d_1 : $alloc(d_1) = alloc(d3_{min}) = (0,1)$. Therefore these two dependences result in identical com-

[‡]Notice that the symmetrical solution $\xi = u_2 = (0,1,0)$ would give analogous results with the localization of the second argument instead of the third.

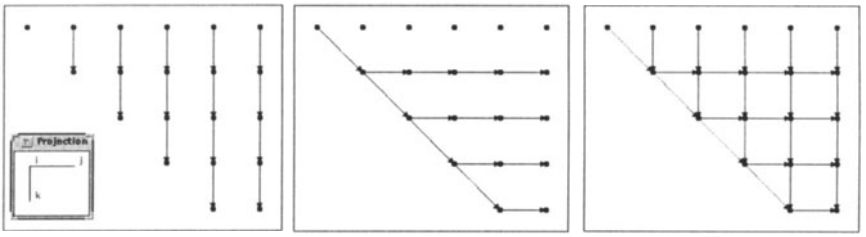

Figure 4 Communication synthesis with $\lambda = (1,1,1)$ and $\xi = (1,0,0)$.

munications from processor $P_{x,y}$ to processor $P_{x,y+1}$ [§]. Because the two corresponding emission sets are disjoint, these two dependences require only one communication per time step : the first time according to $Util_3$ to transmit the third argument and then according to $Util_1$ to successively transmit the different values of the first argument.

Utilization sets $Util_2$ and $Util_4$ require only two communications as shown in the middle window in figure 4. The diagonal arrows correspond to the projection of the minimal extremal vectors $d_{4_{min}}$ and $d_{2_{min}}$: $alloc(d_{4_{min}}) = alloc(d_{2_{min}}) = (1,1)$. The horizontal arrows correspond to the projection of the utilization vectors u_2 and $u_{4,2}$: $alloc(u_2) = alloc(u_{4,2}) = (1,0)$. Notice that the two corresponding emission sets are also disjoint, therefore they only require one communication per time step. Since utilization vector $u_{4,1}$ is parallel to ξ the fourth argument is localized on one dimension, besides being transmitted along the j-axis.

The corresponding parallel solution is visualized in the rightmost window of figure 4. The corresponding parallel code is given figure 5[¶] : in the communication operations the processor references are defined in the **Send** primitives. The processor references in the **Receive** primitives are only given as comments for the reader.

5 COMPARISON WITH OTHER WORKS AND FUTURE DEVELOPMENTS OF OPERA

OPERA's objective is not to capture the general program parallelization problem, like large systems tackling complex real-life applications such as SUIF (Amarasinghe et al. 1995). We deal with the problem of DO-Loop parallelization and focus on a formal version of single-assignment loops : the systems of affine recurrence equations. Since OPERA manipulates single assignment statements, it does not have to analyze memory access conflicts and therefore the dependence analysis is not based on one of the dependence abstractions proposed in the literature, such as the Dependence Distance or the Dependence Direction Vector. In systems of recurrence equations, the dependences are restricted to

[§]In the following the parallel solutions are visualized in a 2-dimensional space corresponding to the 2-dimensional DOALL loop given in section 2.1. We call x the horizontal axis and y the vertical one and we denote the processors by $P_{x,y}$.

[¶]The main objective of this description of a parallel solution is to focus on communications. It does not describe the input and output of data.

```
Processor P(x,y)
(x = y + 1 and 1 ≤ y ≤ n − 1)
   Receive a3 <- P(x, y − 1)
   Receive a4 <- P(x − 1, y − 1)
   R3 = a3
   R4 = a4
   Send a4 to P(x + 1, y)
   Loop Receive a1 <- P(x, y − 1)
        Receive a2 <- P(x − 1, y − 1)
        a = a1 − a2 * R3/R4
        Send a2 to P(x + 1, y)
        Send a to P(x + 1, y + 1)
   EndLoop
```

```
Processor P(x,y)
(y + 1 < x ≤ n + 1 and 1 ≤ y ≤ n − 1)
   Receive a3 <- P(x, y − 1)
   Receive a4 <- P(x − 1, y)
   R3 = a3
   R4 = a4
   Send a4 to P(x + 1, y)
   Loop Receive a1 <- P(x, y − 1)
        Receive a2 <- P(x − 1, y)
        a = a1 − a2 * R3/R4
        Send a2 to P(x + 1, y)
        Send a to P(x, y + 1)
   EndLoop
```

Figure 5 Explicit parallel code for $\lambda = (1,1,1)$ and $\xi = (1,0,0)$.

true dependences and the corresponding modeling is realized in OPERA using the notion of utilization sets.

Because OPERA manipulates single-assignment statements, it does not require symbolic analysis such as loop induction variable identification or loop invariant determination. Since it focuses on fine-grain parallelism, it is not concerned with interprocedural analysis or other techniques related to coarse-grain parallelism. Nevertheless, OPERA as a fine-grain parallelism analyzer could in the future be embedded in a larger system.

Its objective is to deal with problems with affine dependences (vs. uniform ones such as in Bouclettes, Boulet et al. 1994) and to automatically compute a set of parallel solutions for such problems. Among the different parallel solutions, OPERA determines the solutions satisfying some precise criteria such as optimal latency, minimization of the communications, minimization of the number of virtual processors. Communications minimization and locality optimization are determined according to various architectural constraints : we focus not only on local communications, but also on broadcast ones. Proper choices of affine schedules and allocations result in parallel solutions characterized by an efficient use of broadcast primitives if such primitives are available on the target architecture.

To help the user to choose among the many solutions to a problem, we believe that the vizualization tools offered by OPERA allow him to better understand his problem through a geometrical representation of its domain and its dependences. He may successively select various architectural criteria, compare the corresponding solutions in terms of processor count, latency or communication volume and choose the most convenient one.

Compared with other tools dealing with affine problems, such as COMPAR (Arzt et al. 1992), OPERA offers a *full parametrization* of a problem, its domains and dependences, and therefore results in parameterized solutions. These solutions are expressed by loops whose index bounds and array references are defined as functions of the parameters, as well as their latency. This is realized using Loechner and Wilde's representation of the domains and dependences by parameterized polyhedra, and the computation of their vertices as

function of the parameters. Moreover, since OPERA uses theoretical results on polyhedra and linear algebra, it guaranties the correctness of the synthesized parallel solutions.

The current developments in OPERA are concerned with the automatic computation of the allocation directions in accordance to the schedule and to the characteristics of the target architecture. Our next step will be to implement the code generation. We aim at producing either data-parallel code or message passing code similar to the examples given in this paper. We are also working on alignment techniques in the context of affine dependences. This implies, when focusing on problems defined by several variables, the determination of affine-by-variable schedule and allocation functions.

REFERENCES

Amarasinghe, S.P., Anderson, J.M., Lam, M.S. and Tseng C.W. (1995) The SUIF Compiler for Scalable Parallel Machines. *Proceedings of the seventh SIAM Conference on Parallel Processing for Scientific Computing.*

Arzt, U., Teich, J. and Thiele, L. (1992) The Concepts of COMPAR - A Compiler for Massively Parallel Architectures. *Proceedings of IEEE ISCAS, San Diego*, 681–4.

Banerjee, U. (1993) Loop Transformations for Restructuring Compilers, the Foundations. *Kluwer Academic Publishers.*

Boulet, P., Dion, M., Lequiniou, E. and Risset, T. (1994) Reference Manual of the Bouclettes Parallelizer. *Technical report 94-04, ENS-Lyon, France.*

Cytron, R., Ferrante, J., Rosen, B., Wegman, M. and Zadeck, K. (1991) An Efficiently Computing Static Single Assignment Form and the Control Dependence Graph. *ACM Transactions on Programming Languages and Systems*, **13:4**, 451–90.

Fernández, F. and Quinton, P. (1988) Extension of Chernikova's Algorithm for Solving General Mixed Linear Programming Problems. *Technical Report 437, IRISA, Rennes.*

Le Verge, H., Mauras, Ch. and Quinton, P. (1991) The Alpha Language and its use for the Design of Systolic Arrays. *Journal of VLSI and signal processing*, **3**, 173–82.

Le Verge, H. (1992) A note on Chernikova's Algorithm. *Technical Report 635, IRISA, Rennes, France.*

Loechner, V. and Wilde, D. (1995) Parameterized Polyhedra and their Vertices. *Technical Report 95-16, ICPS, Université Louis Pasteur, Strasbourg, France.*

Mongenet, C., Clauss, Ph. and Perrin, G.R. (1991) A Geometrical Coding to Compile Affine Recurrence Equations on Regular Arrays. *5th International Parallel Processing Symposium, IPPS'91, Anaheim, California.*

Mongenet, C. (1995) Mappings for Communication Minimization using Distribution and Alignment. *Conf. on Parallel Architectures and Compilation Techniques, Limassol, Cyprus*, 185–93.

Pugh, W. (1992) A Practical Algorithm for Exact Array Dependence Analysis. *Communications of the ACM, Aug. 1992*, 102–14.

Schrijver, A. (1986) Theory of Linear and Integer Programming. *John Wiley and Sons, New-York.*

Wilde, D. (1993) A Library for doing Polyhedral Operations. *Technical Report 785, IRISA, Rennes, France.*

Zima, H.P. and Chapman, B. (1990) Supercompilers for Parallel and Vector Computers. *ACM Press, Addison Wesley.*