A toolbox for affine recurrence equations parallelization

Vincent Loechner, Catherine Mongenet
Université Louis Pasteur, ICPS
Pôle API, Boulevard Sébastien Brant, 67400 Illkirch, France
Phone : (33) 88 65 50 37
Fax : (33) 88 65 52 49
E-mail : mongenet@icps.u-strasbg.fr

Introduction

Since the concept of systolic architectures has been introduced by H.T. Kung and C.E. Leiserson in the early 80's ([KUL 78], [KUN 82]), many researchers have proposed methodologies for the design of systolic arrays from a problem specification ([MOL 83], [QUI 84], [RAO 85], [MOP 87], among many others).

All these methods, even though they differ in their formalism and tools, rely on the same ideas. From the problem specification expressed in terms of systems of recurrence equations, or equivalently in terms of DO-loops with single assignment statements, a dependency analysis is first realized. A *space* $\times$ *time* mapping is then determined. It consists in a schedule or timing function and in an allocation function. The timing function schedules the different elementary calculations of the problem while the allocation function defines the elementary processor or cell on which each elementary calculation will be computed.

The problems we consider in this paper are defined by systems of affine recurrence equations (SARE in the following). The *space* $\times$ *time* mapping is based on an affine schedule function and a linear allocation. Systolic or more generally processor array solutions can be synthesized. The so-obtained architectures usually require a number of cells much larger than the number of active cells at any instant. This initial solution can be improved by applying specific mapping techniques in order to reduce the number of cells. Two techniques have been proposed to deal with two different situations of low utilization of the processors : the clustering technique ([CMP 90], [BDD 90], [DAR 91]) and the serializing technique ([CMP 90], [CLM 93]). These techniques rely on geometrical tools to represent specific schedule and allocation informations on the domain.

The objective of this paper is to present how the method of parallel architecture synthesis developped by Mongenet, Clauss and Perrin ([MOP 87], [CMP 90], [CLM 93], [MCP 94]) has been implemented in a prototype software. We'll use the classical LL^t Cholesky factorization problem to shortly present the method in section 1, and to illustrate the software capabilities in section 2.

1. From a problem specification to an optimal processor array solution

The problems we consider are characterized by systems of affine recurrence equations (SARE) defined on bounded convex polyhedra. The Cholesky factorization calculates, from an $n \times n$ symmetric positive definite matrix A, a lower triangular matrix L such that $A = LL^t$. It is defined by the following SARE.

$$\begin{array}{rcll}
L(i,j,k) & = & L(i,j,k-1) - L(i,k+1,k) \times L(j,k+1,k) & (1)\\
& & \quad 2 \le i \le n, \quad 2 \le j \le i, \quad 0 \le k \le j-2 & \\
L(i,j,k) & = & L(i,j,k-1)/L(j,j,j-1) & (2)\\
& & \quad 2 \le i \le n, \quad 1 \le j \le i-1, \quad k = j-1 & \\
L(i,j,k) & = & L(i,j,k-1)^{1/2} & (3)\\
& & \quad 1 \le i \le n, \quad j = i, \quad k = i-1 &
\end{array}$$

In order to determine parallel solutions from such a SARE, the dependency analysis has first to be realized from which an affine schedule and a linear allocation will be computed.

The dependency analysis from a SARE relies on the notion of utilization set denoted $\mathrm{Util}_E(Y, z_0)$. This is the set of all the points of the domain using some occurrence $Y(z_0)$ of a variable Y in the calculation. We show in [MCP 94] that this is a convex bounded polyhedron whose linear space is generated by a basis of vectors called the *utilization vectors*. The causal dependency is classically expressed by *dependence vectors*. We show in [MCP 94] that the dependence vectors related to an item $Y(z_0)$ define a *cone* characterized by its finite set of extremal vectors.

Example. The Cholesky factorization. In the 3 subdomains there is a constant dependence vector $\Psi_0 = (0,0,1)$ due to the use of $L(i,j,k-1)$ in all three equations. Equation (1) shows that a given item $L(i,k+1,k)$ is used for the computation of all elements $L(i,j,k)$ with $2 \le i \le n, 2 \le j \le i, 0 \le k \le j-2$. Therefore the corresponding utilization set is a line, generated by the utilization vector $\Phi_1 = (0,1,0)$. The two other utilization sets are respectively generated by $\Phi_2 = (1,0,0)$ and $\Phi_3 = (1,0,0)$. The dependence vectors are respectively :

$$\begin{array}{rcll}
\Psi_1 & = & (0, j-j_0, 0) & \text{with} \quad 1 \le j - j_0 \le i_0 - j_0\\
\Psi_2 & = & (i-i_0, i_0-j_0, 0) & \text{with} \quad 0 \le i - i_0 \le n - i_0\\
\Psi_3 & = & (i-i_0, 0, 0) & \text{with} \quad 1 \le i - i_0 \le n - i_0
\end{array}$$

The definition of a space-time mapping consists of substituting a domain $S \times T$ for the index domain D. This is realized by introducing an affine *schedule function* $t : D \subset \mathbf{Z}^n \longrightarrow \mathbf{N}$ characterised by $t(z) = \lambda \cdot z + \alpha$, and a linear *allocation function* $a : \mathbf{Z}^n \longrightarrow \mathbf{Z}^{n-1}$. The schedule function defines on the domain a set of hyperlanes orthogonal to λ called the *timing surfaces*. The allocation function projects the domain along a direction called *projection direction* and defined by a unimodular projection vector $\xi \in \mathbf{Z}^n$. 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 processor.

Since the seminal work of Karp, Miller and Winograd [KMW 67] the causal constraints are expressed by a system of constraints on the schedule function and the dependence vectors. For a SARE it is expressed by $\lambda \cdot \Psi_{ex} > 0$ for all the extremal dependence vectors Ψ_{ex} associated with the problem. Moreover, if the resulting architecture has to be free of any broadcast, i.e. no variable can be used by several processors at a given time step, the schedule function must satisfy $\lambda \cdot \Phi_Y \neq 0$ for all the utilization vectors Φ_Y of the problem. Finally, the schedule and allocation functions must garanty that a processor executes at most one computation at any given time step. This requires the condition $\lambda \cdot \xi \neq 0$ to hold.

The determination of a processor array solution is realized by applying the schedule and allocation functions to the domain, the utilization and dependence vectors. The projection of the domain results in the shape of the processor array while the projection of the utilization and dependence vectors defines the communications between the processors. This well-known process is fully presented in [MCP 94].

Example. The Cholesky factorization. Any schedule function must satisfy the property $\lambda \cdot \Psi > 0$, which results in $\lambda_1 > 0$, $\lambda_2 > 0$ and $\lambda_3 > 0$. A schedule is therefore $\lambda = (1,1,1)$. Notice that it automatically removes the broadcast since $\lambda \cdot \Phi_1 = \lambda \cdot \Phi_2 = \lambda \cdot \Phi_3 = 1$.

Different allocation directions can then be chosen. They result in different processor array solutions. On these initial architectures different mapping techniques are applied as described in section 2.

2. Mapping techniques and geometrical tools

Arrays derived from linear allocation present many qualities of regularity, locality of the connections. However the disadvantage of most solutions is that they can have a non-optimal parallelism rate : the number of active processors at any instant is much smaller than the total number of processors of the solution.

For a given schedule, a non-optimal rate can be due to two situations. The first one is characterized by processors rythmically active ($\lambda \cdot \xi > 1$). In this case, the processor count can be minimized using a specific mapping technique called the *clustering technique* ([BDD 90], [CMP 90], [DAR 91]). It consists of grouping together onto a single processor neighbouring cells of the initial architecture which have different idle cycles, i.e. which are active at different instants. The second situation ($\lambda \cdot \xi = 1$) occurs when some cells have disjoint activity periods, i.e. some cells become definitely inactive while others have not been active yet. In this case, the low parallelism rate can be avoided by applying the *serializing technique* which maps cells with disjoint activity periods on a single processor ([CMP 90], [CLM 93]).

These techniques require the determination of activity parameters. In order to determine which mapping technique can apply and to help its realization, the activity of the different cells has to be displayed. On the domain the activity is expressed by the timing surfaces and by the facets of the polyhedral domain representing the first (respectively last) calculations on the different cells of the resulting array. These facets are called the *first* (respectively *last*) *faces*. On the initial architecture the activity is expressed by the *activity cone* which indicates the active cells at a given time step. The activity cone at a given time step t_0 is the projection of the timing surface associated with t_0 along the allocation direction ξ.

This geometrical information on the active cells will be used to apply the specific allocation techniques whose objective is to minimize the processor count, i.e. reduce the total number of processors of the resulting architecture, without increasing the global computation time. In many successful cases, we shall reach *size-optimal architectures*, i.e. architectures with a number of processors equal to the parallelism rate (maximum number of simultaneous calculations).

Using the visualization of these activity information, the clustering and serializing techniques can then be applied ([CMP 90], [CLM 93]). We now present the implementation of these geometrical information in a software toolbox.

3. OPERA : the toolbox

The objectives of our software OPERA are to determine from a SARE a set of regular processor arrays synthesized using an affine schedule and a linear allocation functions and to implement geometrical tools described in section 2 to visualize the activity of a parallel solution. This tool is developped on a SUN station running under Solaris2.3 and Openwin3.3.

The initial SARE is described in a file using a syntax similar to the one of ALPHA ([LMQ 91]). Once such a file is loaded, the domain window is automatically computed as shown in figure 1 for the Cholesky factorization (with the size parameter $n = 5$) and the dependency analysis is performed. The causal constraints as well as the broadcast removal contraints are shown in the main window (figure 1). This implementation is based on the method presented in section 1. It manipulates the specification of the domain, timing surfaces, first and last faces by convex pohyhedrons and solves integer linear systems. This is realized using the library POLYHEDRON for the definition and manipulation of polyhedrons developped at IRISA (France) by D. Wilde.

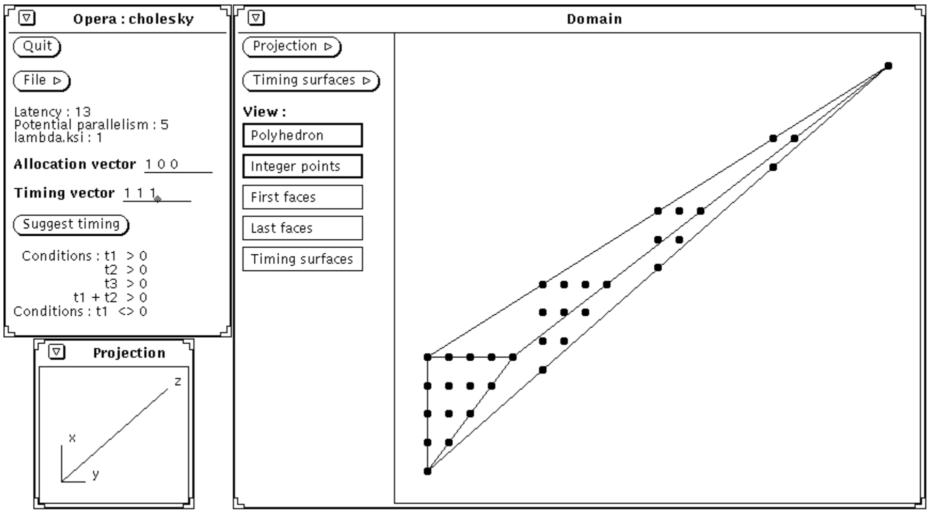

Figure 1: Domain of the Cholesky factorization

The domain can be seen with a different perpective by displacing the axes in the bottom left window. An affine schedule function is automatically computed when using the *suggest timing* button. The user is free to modify the computed values to check other timing solutions. He can then choose and enter an allocation direction. Once these two values have been fixed the processor array solution is visualized in a new window. When the computation functions of the different cells are different, their shapes are drawn differently.

The activity information is implemented using the geometrical modelling described in section 2. It is visualized in the domain window using its buttons. The timing surfaces are shown with wire frames and each timing surface can be separately enlightened and its number of computations determined (see figure 2). The first and last faces can also be visualized as shown in figure 3.

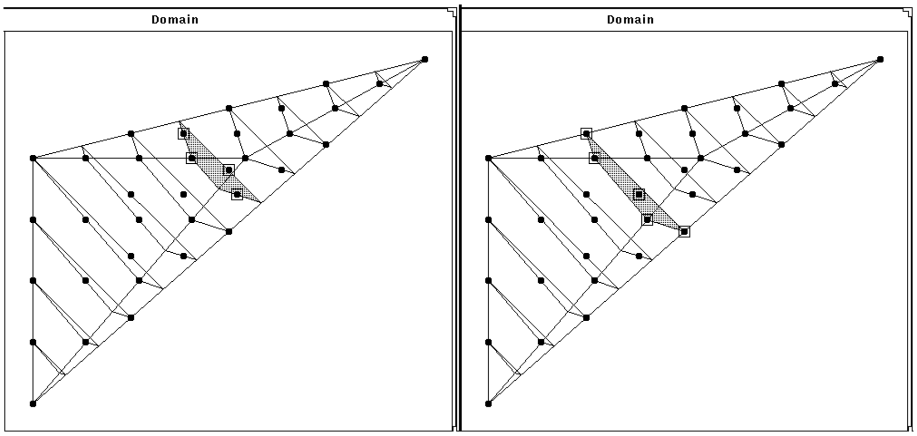

Figure 2: Timing surfaces for $t_0 = 6$ and $t_0 = 7$

Using the *projection* button one can specify a perspective with the allocation direction orthogonal to the screen. In this case the domain's picture corresponds to the processor array solution. On this architecture the activity cone can be visualized at successive time steps as seen in figure 4 for time steps 7,8 and 9. In the grey zone delimiting the activity cone, the active cells are marked.

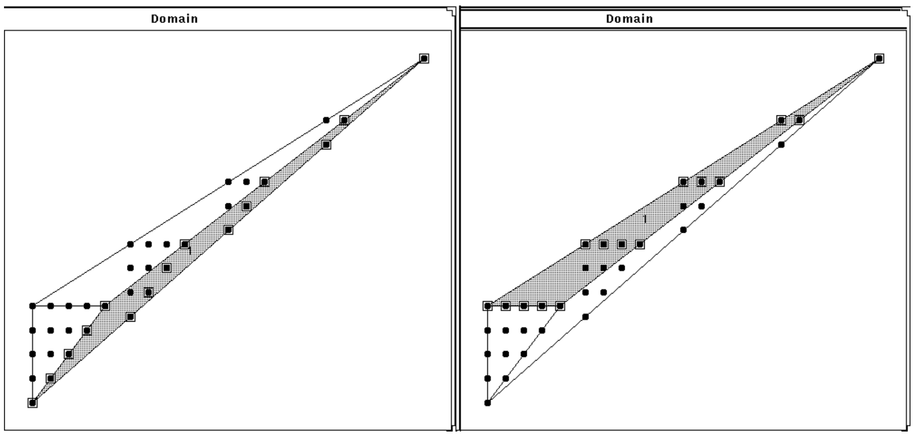

Figure 3: First and last faces of the Cholesky factorization for $\xi = (1, 0, 0)$

In figure 4 all the cells inside the cone are active. It corresponds to the situation $\lambda \cdot \xi = 1$ and hence the serializing technique applies. Conversely when choosing $\xi = (1, 1, 1)$ as allocation direction, $\lambda \cdot \xi$ has value 3 and only one out of three cells inside the cone is active. Hence the clustering technique can apply. The current developments of OPERA are mainly concerned with the implementation of the clustering and serializing techniques and with effective code generation.

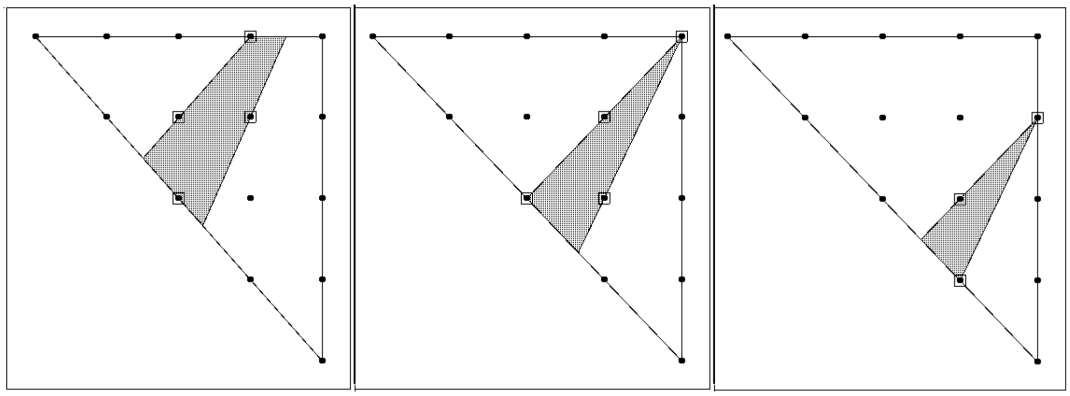

Figure 4: Activity cone of the Cholesky factorization for $\xi = (1, 0, 0)$ and $t_0 = 7, 8, 9$

4. Conclusion

The objective of this work is to implement well-known techniques for processor array synthesis and to develop geometrical tools useful for the optimization of the latency and processor count. The tools proposed here are requested for the application of specific processor mapping techniques such as the clustering and serializing techniques.

The work presented in this paper describes a first version of OPERA. Our goal is to extend this software in order to propose a more general environment for parallel architecture synthesis. It should deal with SARE or equivalent nested DO-loops programs with single-assignment statements and automatically produce various optimal solutions.

References

[BDD 90] Bu J., Deprettere E.F., Dewilde P., (1990) A design methodology for fixed-size systolic arrays. *IEEE Int. Conf. on Application-Specific Array Processors, ASAP'90, Princeton, USA*, pp. 591–602.

[CMP 90] Clauss Ph., Mongenet C., Perrin G.R, (1990). Calculus of Space-Optimal Mappings of Systolic Algorithms on Processor Arrays, *IEEE Int. Conf. on Application-Specific Array Processors, ASAP'90, Princeton, USA*, pp. 4–18.

[CLM 93] Clauss Ph., Mongenet C., (1993). Synthesis aspects in the design of efficient processor arrays from affine recurrence equations. *Journal of Symbolic Computation*, 15, pp. 547-569.

[DAR 91] Darte A., (1991). Regular partitioning for synthesizing fixed-size systolic arrays. *Integration*, 12, pp. 293-304.

[KMW 67] Karp R.M., Miller R.E., Winograd S. (1967). The organization of computations for uniform recurrence equations. *JACM*, 14-3. pp. 563–590.

[KUL 78] Kung H.T, Leiserson C.E., (1978) Systolic Arrays of VLSI, *Sparse Matrix Proc., SIAM* pp. 245-282.

[KUN 82] Kung H.T., (1982). Why systolic architectures? *Computer*, 15-1. pp. 37–46.

[LMQ 91] Le Verge H., Mauras Ch., Quinton P., (1991). The Alpha language and its use for the design of systolic arrays. *Journal of VLSI and signal processing*, 3, pp.173–182.

[MCP 94] Mongenet C., Clauss Ph., Perrin G.R. (1994) Geometrical Tools to Map Systems of Affine Recurrence Equations on Regular Arrays. *Acta Informatica*, 31, pp. 137-160.

[MOL 83] Moldovan D.I. (1983) On the Design of Algorithms for VLSI Systolic Arrays. *Procedings of IEEE*, 71-1. pp. 113–120.

[MOP 87] Mongenet C., Perrin G.R. (1987). Synthesis of Systolic Arrays for Inductive Problems. *Parallel Architectures and Languages Europe Conf., PARLE'87, LNCS 259, Springer Verlag Ed.* pp. 260–277.

[QUI 84] Quinton P. (1984). Automatic Synthesis of Systolic Arrays from Uniform Recurrence Equations. *11th Int. Symp. on Computer Architecture, Ann Arbor, MI, USA, IEEE Computer Society.* pp. 208–214.

[RAO 85] Rao S.K. (1985). Regular Iterative Algorithms and their Implementation on Processor Arrays. *Ph.D. Thesis, Information Systems Lab., Stanford University.*