Alpha Du Centaur : a prototype environment for the design of parallel regular algorithms

Pierrick Gachet, Patrice Quinton, Christophe Mauras, Yannick Saouter

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 953

Programme 2

ALPHA DU CENTAUR : A PROTOTYPE ENVIRONMENT FOR THE DESIGN OF PARALLEL REGULAR ALGORITHMS

Pierrick GACHET
Patrice QUINTON
Christophe MAURAS
Yannick SAOUTER

Décembre 1988

RR_0953

ALPHA DU CENTAUR :
A PROTOTYPE ENVIRONMENT
FOR THE DESIGN OF PARALLEL REGULAR ALGORITHMS

Pierrick GACHET, Patrice QUINTON
Christophe MAURAS, Yannick SAOUTER

Publication Interne n° 439

Novembre 1988

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Alpha Du Centaur : A Prototype Environment for the Design of Parallel Regular Algorithms *

Pierrick Gachet Christophe Mauras
Patrice Quinton Yannick Saouter

Novembre 88
20 pages
Publication interne $n^0$439

Abstract

We describe Alpha du Centaur (ADC), a prototype environment for the design of parallel regular algorithms. In ADC, a program is specified using the Alpha language, using system of parameterized linear recurrence equations. The goal of ADC is to make it possible to transform the initial specifications into a parallel algorithm, that is to say, another system of recurrence equations, in which the time and the space index are separated.

The first section of the paper is devoted to a presentation of the model underlying ADC, i.e., system of recurrence equations. The second section summarizes briefly the knowledge we have on this formalism, and presents some open problems. In the third section, we describe the architecture of ADC, which is based on the CENTAUR environment, and we present an example of utilization of ADC for designing a simple algorithm.

Alpha Du Centaur : vers un environnement de conception d'algorithmes parallèles

Résumé

Ce rapport décrit Alpha Du Centaur (ADC), un environnement de parallélisation d'algorithmes réguliers. Le langage ALPHA, qui constitue l'ossature du prototype, permet de spécifier un problème sous la forme d'un système d'équations récurrentes linéaires, et de représenter l'algoritmme à toutes les phases de sa conception. ADC gère ces réécritures et délivre finalement un algorithme parallèle, en fait un système d'équations récurrentes particulier: les notions de temps et d'allocation des calculs y sont séparées et complètement définies.

La première partie présente le modèle sous jacent. La deuxième énonce les principaux résultats théoriques et pose quelques problèmes ouverts. La dernière partie est consacrée au prototype d'ADC: son architecture basée sur le système CENTAUR, et son utilisation pour la conception d'un algorithme simple.

*Cet article a été soumis au "Workshop on derivation of parallel algorithms and architectures", MCNC, Juin 1989.

1 Introduction

Since the early attempts to use parallel machines, one of the most critical problems has always been how to program such machines. Indeed, departing from the Von Neuman model has many consequences, among which the absence of a model that captures enough of the constraints that the programmer of a parallel architecture has to face, so that his search for efficient algorithms can be formalized. In particular, the *locality* of todays parallel architectures, that is the fact that, for technological reasons, a given processor can be connected to only a few neighbors, is one of these constraints. A consequence is also that not all processors can simultaneously communicate with the host system, thus creating an I/O bottleneck.

One way to take into account this locality property is to constrain the type of algorithms which are considered for parallel implementation. The systolic model defined by Kung and Leiserson ([25, 23, 14]) is one such attempt. By imposing strong rules about the desirable features of a parallel architectures (locality, regularity, modularity, etc.), their model makes it possible to restrain the search to a class of algorithms which are well suited to parallel architectures. Following their approach, researchers have designed many systolic architectures (or algorithms) for a wide variety of algorithms ([24, 26]). On the other hand, attempts to formalize the way such algorithms could be obtained by automatic means have been made ([35, 39, 3, 34, 8, 45, 5]). More or less explicitly based on the early work of Karp, Miller and Winograd ([22]), these attempts try to make use of the property that many algorithms, at least in the numerical fields, are naturally expressed as recurrences indexed by n-dimensional integer points.

In this paper, we present our effort towards designing an environment for the manipulation of system of linear recurrences. This environment is called ADC (Alpha du Centaur). An earlier attempt in this direction was the DIASTOL system ([41, 15]), whose limitations made possible to handle only very simple algorithms. Other similar attempts were the ADVIS software package ([36]). More recently, Chen started a more ambitious effort ([6]) for designing the CRYSTAL system. Among other works, let us mention also the work of Lengauer [30], the SYSTOL system [37], and the PRESAGE software [10].

2 The model

The underlying model of ADC is so called *system of parameterized affine recurrence equations.* Such equations have the form :

$$\forall z \in D_P, U(z) = f[V(I(z)), ...]$$

where:

- z is a point in the set $\mathbf{Z}^n$ ($\mathbf{Z}$ denotes the set of integers),
- D_P is a convex polyhedral domain of $\mathbf{Z}^n$, parameterized by P, a point in $\mathbf{N}^p$, the *parameter* of the equation,
- U and V are variables indexed by points in $\mathbf{Z}^n$,

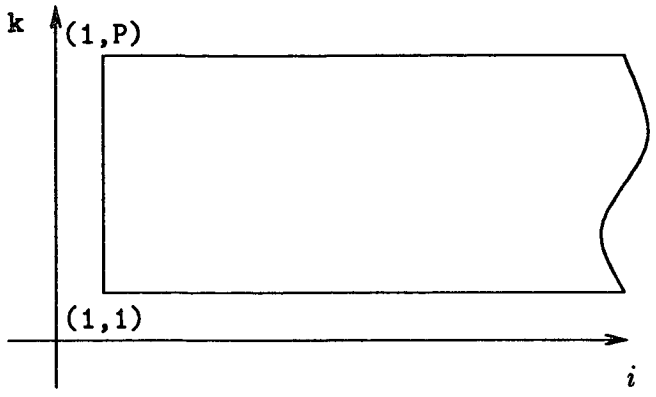

Figure 1: Domain of equation (1)

- I is a affine mapping from $\mathbf{Z}^n \times \mathbf{N}^p$ to $\mathbf{Z}^m$, $m \leq n$,
- f is a function whose computation takes $O(1)$ unit of time. The right side member can have more than one argument.

A set of such equations defines a *problem*, each instance of which being obtained when the value of P is fixed.

For example, the following set of equations defines the convolution algorithm:

$$i \geq 1, 1 \leq k \leq P \rightarrow Y(i,k) = Y(i,k-1) + w(k) \times x(i-k+1) \quad (1)$$

$$i \geq 1, k = 0 \rightarrow Y(i,k) = 0 \quad (2)$$

$$i \geq 1 \rightarrow y(i) = Y(i,P) \quad (3)$$

Equation (1) defines a variable Y over a 2-D convex polyhedral domain D_P shown in figure 1. This domain is linearly parameterized by $P \in \mathbf{N}$. When P ranges in $\mathbf{N}$, D_P can be seen as the intersection of a convex domain in $\mathbf{Z}^3$ and the plane $p = P$ (see figure 2). The linearity hypothesis we make about the domains of the equations will prove to be extremely useful when trying to apply some of the transformations needed to obtain a parallel algorithm. Any equation obtained from equation (1) by setting the value of i and k in D_P is called an *equation instance*, $Y(i,k)$ is called the *result* of the equation instance, and $w(k)$ and $x(i-k+1)$ are the *arguments*. Given i and k, $Y(i,k)$ is an *instance of the variable* Y. Moreover, $Y(i,k)$ is said to be an *intermediate instance*, as it appears both as the result of an equation instance, and as an argument of another equation instance. Similarly, $w(k)$ and $x(i-k+1)$ are said to be *input instances*, as they appear only as arguments, and $y(i)$ is an *output instance* appearing only as a result. In the following, we shall only consider system of equations in which all instances of a given variable are either intermediate, input or output. A simple transformation (see [38]) makes it possible to have this situation.

In ADC, a system of recurrence equation constitutes the starting specification of an algorithm. At the other end of the process, we seek a new form of the algorithm which can be

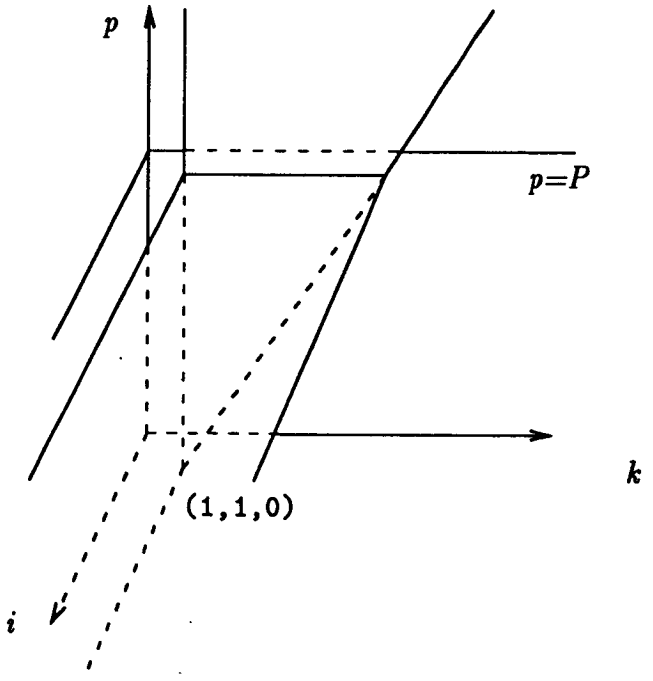

Figure 2: Domain of equation (1) as represented in $\mathbf{Z}^3$

directly mapped on a parallel (systolic) architecture. This new form is a special case of recurrence equations where the indexes explicitly represent the time (assumed to be discrete) and the processor number. To illustrate this, consider the well known systolic implementation of the convolution shown in figure 3. Let $t \in \mathbf{N}$ denote the time, and let p denote the processor number. The algorithm can be described using the following set of recurrence equations, where $\perp$ denotes the undefined value:

$$
\begin{aligned}
t \geq 1, 1 \leq p \leq P &\rightarrow Y(t,p) = Y(t-1,p-1) + W(t-1,p) \times X(t-1,p-1) \\
t = 0, 1 \leq p \leq P &\rightarrow Y(t,p) = \perp \\
t \geq 0, p = 0 &\rightarrow Y(t,p) = 0 \\
t \geq 1, 1 \leq p \leq P &\rightarrow X(t,p) = X1(t-1,p) \\
t = 0, 1 \leq p \leq P &\rightarrow X(t,p) = \perp \\
t \geq 1, 1 \leq p \leq P &\rightarrow X1(t,p) = X(t-1,p-1) \\
t = 0, 1 \leq p \leq P &\rightarrow X1(t,p) = \perp \\
t \geq 1, 1 \leq p \leq P &\rightarrow W(t,p) = W(t-1,p) \\
t = 0, 1 \leq p \leq P &\rightarrow W(t,p) = w(p) \\
t \geq 0 &\rightarrow y(t) = Y(t,P) \\
t \geq 0, p = 0 &\rightarrow X1(t,p) = x(t)
\end{aligned}
$$

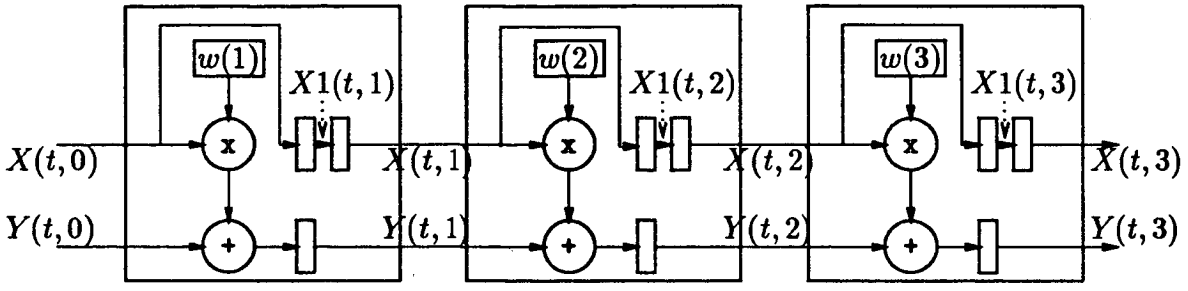

Figure 3: Systolic array for the convolution

These equations have the particularity that the time t and space index p in an argument is a translation of t, p. This reflects the spatial and temporal locality of the systolic array. Notice that, depending on the interconnection network of the target architecture, it would be possible to accept that the space index of the arguments be another function, provided that a direct connection corresponding to this function be available on the network. For the sake of simplicity, in the following, we shall only translations.

The transformation we have to do in order to go from the initial specification to the architecture is basically a time-space reindexing. In simple cases such as our example, a simple linear reindexing scheme makes it possible to obtain directly the result. However, in general, many intermediate transformations have to be done, and we have yet only a very partial knowledge of what is feasible. In the following, we briefly summarize the main results obtained so far, and we exhibit some of the problems which are open.

3 Results and open problems

3.1 Computability

The seminal work of Karp et al. ([22]) introduced a somewhat simpler type of recurrence equations, called *uniform recurrence equations*. There are two main differences between Karp's et al. definition and our one:

- the index function I in an argument is a translation (i.e., the equations are uniform),
- the domain of the equations are identical for all equations, and is restricted to simple cases (the first orthant, or semi-infinite domains); in this case, we shall say that the system is *strict*.

As a result, it is not always natural to express an algorithm using such a formalism. For example, even defining initial conditions of an algorithm is not possible, as it would imply defining some variables on a subdomain (see equation (2) for example). Under these hypotheses, Karp et al. have been able to give solutions to several problems. One of this problem, known as the

computability problem, is to decide whether a system is implicit or not, i.e., whether none of the variable instance depend on itself. Karp et al. have shown that this problem is solvable.

If we remove the assumption that all equations have the same domain, this simple problem becomes undecidable. Joinnault ([20]) has shown that given one system of recurrence equations (with convex polyhedral domains), the computability problem is undecidable. Indeed, it is possible to encode the operation of a Turing Machine using such equations, and one can show that the computability amounts to proving that the Turing machine halts. Of course, if we restrict the domain to be bounded, it is always possible to check the computability by looking at the (finite) dependence graph of the system. However, when we consider a parameterized system, it would be nice to check the property at once for all the values of the parameters. Unfortunately, this property is undecidable ([43]).

Therefore, an open problem is to find good restrictions of the general model, that is restrictions which make it possible to check such a property, and still permit to express a large class of algorithms, for which we know that the property holds.

3.2 Uniformization of recurrence equations

In general, the initial specification of an algorithm is not uniform, i.e., the index mapping are not translations, as shown by the simple example of the convolution.

Consider for example equation (1):

$$i \geq 1, 1 \leq k \leq P \rightarrow Y(i,k) = Y(i,k-1) + w(k) \times x(i-k+1)$$

where the index of the variable x is the affine function $i-k+1$, that we can rewrite:

$$I(i,k) = \begin{pmatrix} 1 & -1 \end{pmatrix} \begin{pmatrix} i \\ k \end{pmatrix} + (1)$$

This index function is a mapping from $\mathbf{Z}^2$ to $\mathbf{Z}$, and therefore, the same instance variable is broadcasted to several calculations. If we want to avoid this broadcasting, which can be interpreted as the need of a bus in the final architecture, it is possible to *pipeline* variable x. To do so, it suffices to notice that a given instance $x(I(z))$ is common for all points of the form $x(I(z+v))$ where v is a vector of the null-space of the linear part of I, as $I(v) = 0$ (see [13, 16] for more details). This vector defines, up to a multiplicative factor, the direction along which the data can be pipelined. In the case of the convolution, the vector $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ generates the null-space of I. Therefore, we can replace equation (1) by a new set of equations:

$$\begin{aligned} i \geq 1, 1 \leq k \leq P &\rightarrow Y(i,k) = Y(i,k-1) + w(k) \times X(i,k) && (4) \\ i-k \geq -P+1, 1 \leq k \leq P &\rightarrow X(i,k) = X(i-1,k-1) && (5) \\ i-k \geq -P+1, k = 0 &\rightarrow X(i,k) = x(i+1) && (6) \end{aligned}$$

Equation (5) defines a new variable X on a domain which is an extension of the domain of equation (4), so that X is initialized along the line $k = 0$. Equation (6) define the initial values

of X, on the line $k = 0$. The same operation can be done in order to remove the broadcasting of $w(k)$.

In general, removing broadcasting of data is a more complex task, especially when the variable to be broadcasted is not an input of the algorithm, but is computed by the algorithm itself. The interested reader can find in [46, 11, 40] a detailed treatment of the subject.

3.3 Time space transformations

It has been recognized early that the design of systolic arrays amounts to find a suitable linear time space transformation of the initial index space. This result, extending the work of Lamport on do-loops ([27]), was shown independently by Moldovan ([35]), Cappello and Steiglitz ([3]), Miranker and Winkler ([34]), and Quinton ([39]). These results differ in the hypothesis they make, and the constructiveness of the conditions they give.

These methods can be sketched in the following way. Let $\mathbf{V}$ denote the set of variable names of the system. The idea is to seek a linear (or affine) timing mapping t from $\mathbf{V} \times \mathbf{Z}^n$ to $\mathbf{N}$ and a linear allocation mapping a from $\mathbf{Z}^n$ to $\mathbf{Z}^{n-1}$ that satisfy the following conditions:

- (i) if $U(z)$ depends on $V(z)$, then $t(U,z) > t(V,z)$,
- (ii) if $t(U,x) = t(U,y)$ then $a(x) \neq a(y)$.

The mapping $t(U,z)$ is interpreted as the time at which the calculation of the equation instance defining $U(z)$ is done, and $a(z)$ is interpreted as the processor number where all the calculations having index z are done.

Let $t(U,z) = \sum_{k=1}^{n} \lambda_k z_k + \alpha_U$ be the timing function, assuming that λ_k and α_U all belong to $\mathbf{Z}$. In the particular case when the index functions I are translations, it is easy to show that the λ_k and the α_U coefficient can be found by solving a linear integer program. More precisely, let $U(z)$ and $V(I(z))$ be a pair of result and argument of an equation. We must have:

$$t(U,z) \geq t(V,I(z)) + 1$$

or, equivalently, denoting $\lambda^T = (\lambda_1, ...\lambda_n)$,:

$$\lambda^T(z - I(z)) + \alpha_U - \alpha_V \leq 1 \tag{7}$$

As I is a translation, $z - I(z)$ is independent on z, and we have a finite number of such inequalities.

Among the variants of this result, let us cite:

- the extension to rational timing functions, and to non bounded convex polyhedral domains ([39]),
- another form of the result which makes it possible to handle the case when the functions can be implemented using combinatorial logic ([42, 9]),
- the extension to multi-dimensional timing functions ([35]).

A similar result can be given when the index function I is an affine mapping. In this case, equation (7) may generates infinitely many inequalities, when z ranges over D_P, and P ranges over N. However, noticing that D_P is a convex domain, it can be proven ([40]) that equation (7) can be replaced by a finite number of inequalities, that are obtained using the vertices, rays, and lines of D_P[1] This result makes it possible to handle the case of linear recurrence equations in a very similar way.

The simplest way to define a mapping a, that we call an *allocation function*, is to use a projection of the space $\mathbf{Z}^n$ on $\mathbf{Z}^{n-1}$ along a direction u of $\mathbf{Z}^n$ which is not parallel to the hyperplane defined by the timing function[2]. Other allocation methods, requiring usually a smaller number of processors have been studied by Louka and Tchuente ([32]).

Let us illustrate the time-space reindexing on the convolution algorithm. The timing-function parameters can be chosen as:

$$\lambda_1 = \lambda_2 = 1$$
$$\alpha_Y = P - 2$$
$$\alpha_W = \alpha_X = P - 3$$

so that the input of the first coefficient $x(1)$ be done at time 0. On the other hand, in order to obtain a finite projection for the domain, the only solution is to use the direction $u = (1, 0)$. Therefore, the architecture has P processors, each one of which associated with a point of the segment $[(0, 1), (0, P)]$. This is the solution presented in figure 3.

It should be noticed that a system which can be linearly scheduled is necessarily computable. At least, we know that such a method, when successful, produce correct results.

3.4 Other transformations

In general, a system of recurrence equation cannot be directly time-space reindexed. It can be necessary to modify the system using more complex transformations that concern subdomains of the systems. This amounts to using piecewise linear reindexing. Another way of looking at the same problem is to consider that a system is made of several steps which are indexed separately and assembled together afterwards. Results on this subject have been given by Delosme and Ipsen ([7]). In order to identify the steps of a system of equations, they analyze the *reduced dependence graph* of the system (see also [42]), that is the graph whose vertices are variable names, and edge link dependent variables. Steps are defined using strongly connected components of the dependence graph. Then they show that scheduling the whole system amounts to apply the time-space reindexing independently each step, then assemble together the steps. Again, this can be done by solving an integer linear program.

There are some cases when a system of recurrence equation whose reduced dependence graph is strongly connected cannot be directly reindexed, for example in the Gauss-Jordan elimination algorithm ([31, 38]). It is still an open problem whether there exist method for identifying such a situation and find out how to decompose the domains.

[1] A vertex of a convex polyhedron is an extreme point of the domain. A ray (resp. a line) is the direction of a half-line (resp. of a line) included in the domain (see [44]).

[2] When the timing function is multi-dimensional, say of dimension q, then a is from $\mathbf{Z}^n$ to $\mathbf{Z}^{n-q}$.

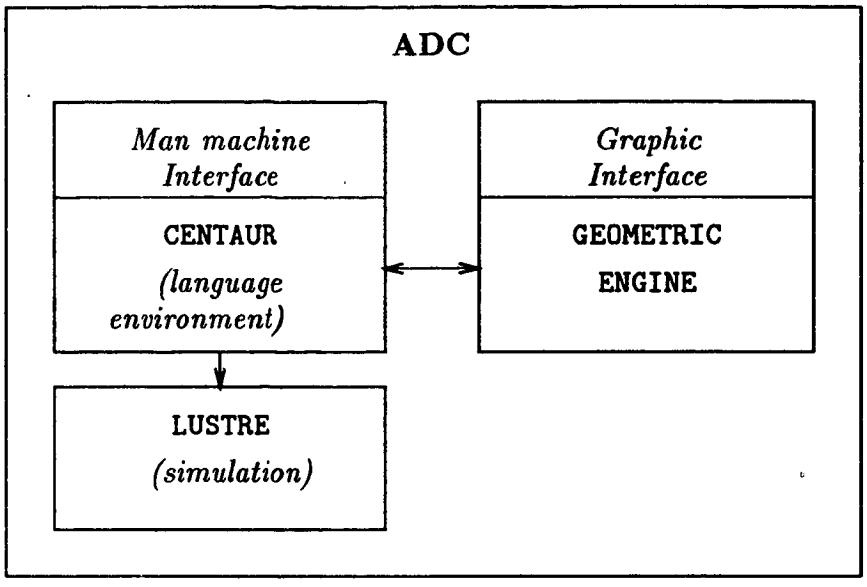

Figure 4: Architecture of Alpha du CENTAUR

Finally, work has been done on the *partitioning problem*, which is very important for the design of parallel algorithms for fixed size architectures ([36, 19]). But in general, optimality results on partitioning are missing.

4 Architecture of the system

The goal of ADC is mainly to help understanding some of the problems we have mentioned on the preceding section. Our experience designing the DIASTOL software package has shown us that it is important to have tools in order to study in a systematic way the transformations. Indeed, some of the transformation are very difficult to apply by hand, even if they are conceptually quite simple.

The logical architecture of ADC is shown in figure 4. ADC comprises three parts : a language environment, which is built using the CENTAUR system, a geometric engine, and a simulation environment based on the LUSTRE language ([4]).

4.1 The language environment

The language environment of ADC is based on the use of the CENTAUR system. CENTAUR ([2]) is a generic interactive environment designed by INRIA as part of an ESPRIT project. When given the formal specification of a particular programming language - including its syntax and semantics - it produces a language specific environment. This environment comprises a structure editor, an interpreter/debugger and other tools, all of which have graphic man-machine interfaces.

The CENTAUR architecture is based on three elements. The *kernel* supports symbolic manipulations of structured documents, that is, manipulations used while editing documents, and also evaluations, more common in semantic processing. A *specification level* supports the syntactic and semantic aspects of languages. Finally, a *user interface* takes care of all interactive communication between the system and its user.

The kernel is the central part of CENTAUR. It is made of two specialized components, intended to be used as co-routines: the *Virtual Tree Processor* (VTP), which handles syntactic aspects, and the *logical machine*, i.e., a Prolog interpreter, whose function is to handle all semantic aspects.

Designing a language environment consists in providing CENTAUR with the abstract syntax, a concrete syntax, and a semantics of the language. Once these inputs are given, one obtain a scanner, a multi-entry parser, a pretty printer, and abstract syntax tables. Therefore, CENTAUR is ready to process a textual input (using the scanner and the parser), to edit the text, and to check the validity of editing operation. The semantics of the language, described using the language TYPOL, is compiled into Prolog. TYPOL is based on the so called "Natural Semantics" ([21])

The man-machine interface makes it possible to have several views, each one of which related to one particular point of view on the program being edited, or on the specifications of the language itself. Therefore, modifying these specifications is easy, and greatly simplifies the task of defining a language.

The language for describing algorithms in ADC is called ALPHA. It is designed to fulfill the following goals:

- the language must be able to describe system of linear recurrence equations in a global and functional way, at all the levels of transformation, from the initial specification to the final parallel implementation,
- ALPHA follows an *equational* approach, first of all, because it is the "natural" way to express algorithms we want to consider (mainly, numerical and signal processing algorithms), and also because it is recognized that such an approach is a good framework for the study of algorithm transformation and proof (see [1]).

The ALPHA language is inspired from LUCID (unique assignment equational data flow language). However, we generalize the notion of *flow* of LUCID to multi-dimensional functions: in ALPHA, a variable is a function on a convex polyhedral domain of $\mathbf{Z}^n$, instead of a simple sequence. As a consequence, the causality analysis in ALPHA is more complex than in LUCID. In addition, ALPHA is also inspired from LUSTRE ([4]) which can be seen as a synchronous real-time LUCID. LUSTRE (as other synchronous languages like SIGNAL [28]) is very well suited to the description of synchronous parallel algorithms or architectures. In particular, it is possible to prove the correctness of circuits ([17]), and express some well-known transformations such as the retiming of Leiserson et al. ([29] [18]). ALPHA departs from LUSTRE in that it does not convey implicitly the notion of time, which actually results from the transformations that are applied on the initial specifications. Therefore, it is possible in ALPHA to specify an algorithm without referring to the notion of execution.

In the following, we present the main notions in ALPHA, starting from the example of the convolution. In ALPHA, a variable is a mapping from a spatial domain to a domain of values. Basic domains of values are the usual elementary types (integer, boolean, real, etc.). The notion of spatial domain extends the classical notion of array in usual languages. A spatial domain is specified by a set of integral linear inequalities.

A program is a function that maps a set of ALPHA variables (input variables) to a set of (output) ALPHA variables. An ALPHA program consists in the declaration of input, output and intermediate variables, and in a system of equations defining the intermediate and output variables. The semantics of an ALPHA program is thus simply the least fixed point of its equation system, in the sense of LUCID, that is the usual order on functions. The goal of the transformations we apply to the initial form of an algorithm is to seek an equivalent form where the notion of time is explicitly defined, and for which the causality property (no reference to the future, and no instantaneous cycle) can be checked. Therefore, when these transformations are successful, what we obtain is a LUSTRE program.

In order to illustrate these notions, consider the example of the convolution:

```
system convolution (w : {i | 1<=i<=8} of integer;
                    x : {i | i>=1} of integer)
returns (y : {i | i>=1} of integer)
var
  Y : {(i,k) | 1<=i, 0<=k<=8} of integer;
let
  Y(i,k)= case
      k=0  : 0;
      k>=1 : Y(i,k-1) + w(k) * x(i-k+1);
     esac;
  y(i)= Y(i,8)
tel
```

This ALPHA program correspond directly to equations (1) to (3): w and x are declared as input variables, y as an output variable, and Y as an intermediate variable. The part between the keywords `let` and `tel` describe the equations. For the sake of simplicity, the notion of parameter is not included in this example. Some simple syntactic constraints on the language and semantics calculus make it possible to check that the system is well-formed, i.e., that the unique assignment rule is satisfied, and the use of the variables conforms their declaration. In ADC, this is done using TYPOL.

After pipelining x and w, we obtain:

```
system convolution (w : {i | 1<=i<=8} of integer;
                    x : {i | i>=1} of integer)
returns (y : {i | i>=1} of integer)
var
  Y : {(i,k) | 1<=i, 0<=k<=8} of integer;
  W : {(i,k) | 0<=i, 1<=k<=8} of integer;
```

```
  X : {(i,k) | i-k>=-7, 0<=k<=8} of integer;
let
  Y(i,k)= case
      k=0  : 0;
      k>=1 : Y(i,k-1) + W(i,k) * X(i,k);
     esac;
  W(i,k)= case
      i>=1 : W(i-1,k);
      i=0  : w(k);
     esac
  X(i,k)= case
      k>=1 : X(i-1,k-1);
      k=0  : x(i+1);
     esac
  y(i)= Y(i,8)
tel
```

After the space-time reindexing, the final version is:

```
system convolution (w : {i | 1<=i<=8} of integer;
                    x : {i | i>=1} of integer)
returns (y : {i | i>=1} of integer)
var
  Y : {(t,p) | t-p>=6, 0<=p<=8} of integer;
  W : {(t,p) | t-p>=5, 1<=p<=8} of integer;
  X : {(t,p) | t-2p>=-2, 0<=p<=8} of integer;
let
  Y(t,p)= case
      p=0  : 0;
      p>=1 : Y(t-1,p-1) + W(t,p) * X(t,p);
     esac;
  W(t,p)= case
      t-p>=6 : W(t-1,p);
      t-p=5  : w(p);
     esac
  X(t,p)= case
      p>=1 : X(t-2,p-1);
      p=0  : x(t+3);
     esac
  y(i)= Y(i+13,8)
tel
```

At this level, this system as a semantic strictly equivalent to that of the initial specification. Notice that in these equations, the names of the indexes denote only the position of the index

in the left-hand side variable. In that sense, writing `y(i)=Y(i+13,8)` is strictly equivalent to `y(u)=Y(u+13,8)`. However, in the last form of the algorithm, we can interpret the first index as the time at which a variable instance is calculated, and the second one as the processor number where it is calculated.

This last version can be translated into LUSTRE and simulated using the LUSTRE interpreter. In addition to the simulation, it will be easy with CENTAUR to generate code for several parallel architectures, at least, when no partitioning is necessary. Indeed, from the final version, is only necessary to write the code of the process which is executed on each processor. This can be done using the TYPOL processor of CENTAUR.

4.2 The geometric engine

The geometric engine is a program which is being developed using the earlier version of DIASTOL ([41]). Its main role is to compute the geometric part of the transformations which have to be applied to the indexes. The kernel of the geometric engine is an algorithm which transform the description of a convex polyhedron using integral linear inequalities, and computes the set of its vertices, rays and lines, and conversely. This algorithm is based on an extension of Chernikova's algorithm ([12]) which runs extremely fast for small problems such as those we have here (a few tens of inequalities at most). As shown in figure 5, finding the vertices, rays and lines of a convex polyhedron amounts to finding the rays of a convex polyhedral cone, when homogeneous coordinate are used: a vertex (such as s_1 and s_2 in figure 5) is a ray of the cone whose Z coordinate is non zero, and a ray (or a line) is a ray of the cone whose Z coordinate is 0. Notice that the dual problem, that is finding the linear inequalities defining a convex, given the vertices, rays and lines, is solved by the same algorithms, as in the case of a cone, the inequalities are the rays of the polar cone.

The algorithm, together with a few simple linear algebra routines, makes it possible to solve most of the calculations we have to do in order to transform the equations, for example computing the image of a convex by a linear transformation, the intersection of two convexes, the projection of a convex, etc. Moreover, this algorithm is used as the basis for solving linear programs. As the set of constraints is small, it is simpler to compute all the vertices and rays of a domain defined by a set of linear constraints, then to evaluate the linear function to be optimized at the vertices, than to use a pivoting method such as the simplex.

The geometric engine is provided with a graphic window which makes it possible to visualize domains used in the algorithm.

5 Conclusion

We have presented Alpha Du Centaur, an environment for the design of parallel regular algorithms. ADC comprises a language environment (based on CENTAUR), a geometric engine, and a simulation environment (based on the LUSTRE interpreter). The model underlying ADC is the notion of system of parameterized linear recurrence equations. ALPHA, the language of ADC, makes it possible to describe an algorithm, from the initial specification to the final version. Transformations are performed using calculations done by the geometric engine. ADC

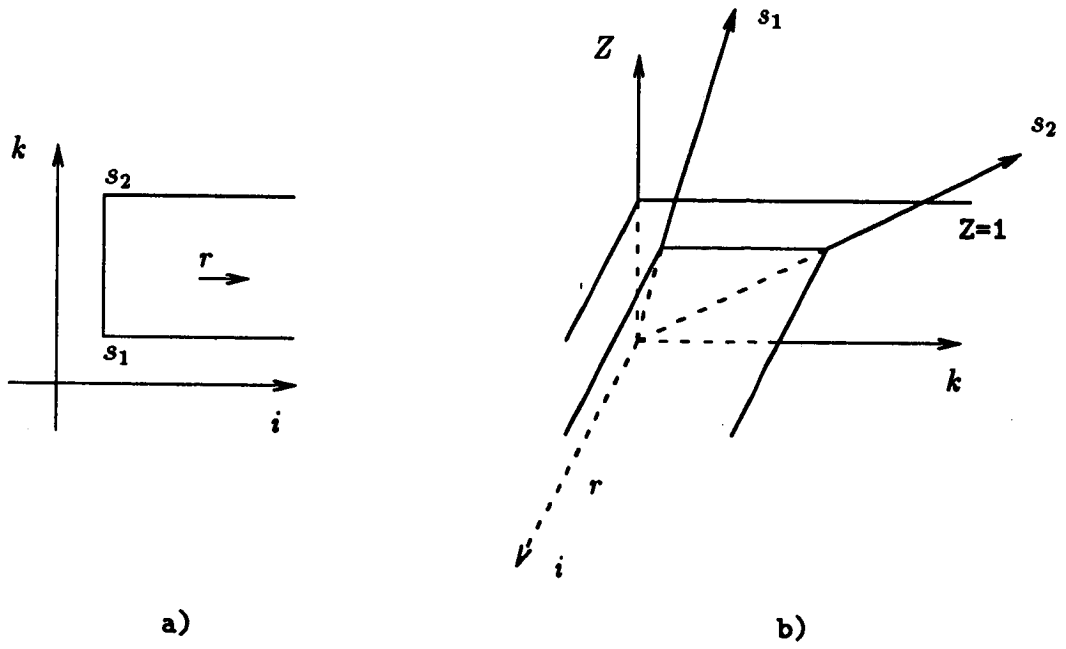

Figure 5: Convex polyhedral domain associated with a domain

is still under development. The ALPHA language is being extended in order to include block structuring. We plan to generate code for parallel architectures such as the Intel iPSC, or Transputer based parallel architectures.

References

[1] E.A. Ashcroft and W.W. Wadge. LUCID : a non procedural language with iteration. *CACM*, 20(7), July 1977.

[2] P. Borras, D. Clément, Th. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. *CENTAUR : the System.* Technical Report 777, INRIA, December 1987.

[3] P.R. Cappello and K. Steiglitz. Unifying VLSI array design with linear transformations of space-time. In F.P. Preparata, editor, *Advances in Computing Research*, JAI Press Inc., Greenwich, U.K., 1984.

[4] P. Caspi, D. Pilaud, N. Halbwachs, and J.A. Plaice. LUSTRE : a declarative language for programming synchronous systems. In *Fourteenth Annual ACM Symp. on Principles of Programming Languages*, pages 178–188, ACM, Munich (West Germany), January 1987.

[5] M.C. Chen. *Space-Time Algorithms Semantics and Methodology.* PhD thesis, California Institute of Technology, Pasadena, California, May 1983.

[6] M.C. Chen. *Transformations of Parallel Programs in Crystal.* Technical Report YALEU/DCS/RR-469, Yale University, April 1986.

[7] J.M. Delosme and I.C.F. Ipsen. "An illustration of a methodology for the construction of efficient systolic architectures in VLSI," *Proc. Second Int. Symposium on VLSI technology, systems and applications*, Taipei, Taiwan, Mai 1985, 268-273.

[8] J.M. Delosme and I.C.F. Ipsen. Efficient systolic arrays for the solution of Toeplitz systems: an illustration of a methodology for the construction of systolic architectures in VLSI, Rapport de recherche YALEU/DCS/RR-341, Department of Computer Science, Yale University, U.S.A., Juin 1985.

[9] J.M. Delosme and I.C.F. Ipsen. Systolic array synthesis : computability and time cones. In *Parallel Algorithms and Architectures*, pages 295–312, North-Holland, 1986.

[10] V. Van Dongen. PRESAGE, a tool for the design of low-cost systolic circuits. In *IEEE International Symposium on Circuits and Systems*, Espoo, Finland, June 7-9 1988.

[11] V. Van Dongen and P. Quinton. Uniformization of linear recurrence equations : a step towards the automatic synthesis of systolic arrays. In *International Conference on Systolic Arrays*, San Diego (EU), Mai 1988.

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

[13] J.A.B. Fortes and D.I. Moldovan. Data broadcasting in linearly scheduled array processors. In *Proc. 11th Annual Symp. on Computer Architecture*, pages 224–231, 1984.

[14] M.J. Foster and H.T. Kung. "The design of special-purpose VLSI chips," *Computer* 13, 1 (1980), 26-40.

[15] P. Frison, P. Gachet, and P. Quinton. Designing systolic arrays with DIASTOL. In S.Y. Kung, R.E. Owen, and J.G. Nash, editors, *VLSI Signal Processing II*, pages 93–105, IEEE Press, November 1986.

[16] P. Gachet, B. Joinnault, and P. Quinton. Synthesizing systolic arrays using DIASTOL. In W. Moore, A. McCabe, and R. Urquhart, editors, *International Workshop on Systolic Arrays*, pages 25–36, Adam Hilger, University of Oxford, UK, July 2-4 1986.

[17] N. Halbwachs, A. Longchamp, and D. Pilaud. Describing and designing circuits by means of a synchronous declarative language. In *From HDL Description to Guaranteed Correct Circuit Designs, Proceedings of IFIP Working Conference, Grenoble (France)*, North-Holland, 1986.

[18] N. Halbwachs and D. Pilaud. Use of real-time declarative language for systolic array design and simulation. In W. Moore, A. McCabe, and R. Urquhart, editors, *International Workshop on Systolic Arrays*, pages 81–90, Adam Hilger, University of Oxford, UK, July 2-4 1986.

[19] K. Jainandunsing. Optimal partitioning schemes for wavefront/systolic array processors. *IEEE Trans. on Computers*, XX(Y):940–943, 1986.

[20] B. Joinnault. Conception d'algorithmes et d'architectures systoliques. Thèse de l'Université de Rennes I, Sept 1987.

[21] G. Kahn. Natural semantics. In *STACS 1987*, Springer-Verlag, March 1987.

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

[23] H.T. Kung. Why systolic architectures? *Computer*, 15(1):37–46, January 1982.

[24] H.T. Kung and C.E. Leiserson. "Systolic arrays for (VLSI)", in [33] chapitre 8.3.

[25] H.T. Kung and C.E. Leiserson. Systolic arrays (for VLSI). In *Sparse Matrix Proc. 1978*, pages 256–282, Society for Industrial and Applied Mathematics, 1978.

[26] S.Y. Kung. *VLSI Array Processors.* Prentice Hall, 1988.

[27] L. Lamport. The parallel execution of Do-Loops. *Comm. ACM*, 83–93, Feb. 1983.

[28] P. LeGuernic, A. Benveniste, P. Bournai, and T. Gautier. SIGNAL: a data flow oriented language for signal processing. In *IEEE Workshop on VLSI*, 1984.

[29] C.E. Leiserson, F.M. Rose, and J.B. Saxe. Optimizing synchronous circuitry by retiming (preliminary version). In R. Bryant, editor, *Proc. 3d Caltech Conf. on VLSI*, pages 87–116, Computer Science Press, 1983.

[30] C. Lengauer. *Toward Systolizing Compilation : an Overview.* Technical Report TR-88-37, Department of Computer Science, The University of Texas at Austin, October 1988.

[31] P.S. Lewis and S.Y. Kung. Dependence graph based design of systolic arrays for the algebraic path problem, *Proc. 20-th Asilomar Conference*, 10-12 Novembre 1986.

[32] B. Louka and M. Tchuente. Dynamic programming on two-dimensional systolic arrays. *Information Processing Letters*, 29, September 1988.

[33] C.A. Mead and L.A. Conway. *Introduction to VLSI systems*, Addison-Wesley, Reading, Mass., USA, 1980.

[34] W.L. Miranker and A. Winkler. "Space-time representations of systolic computational structures," *Computing* 32 (1984), 93-114.

[35] D.I. Moldovan. On the analysis and synthesis of VLSI algorithms. *IEEE Transactions on Computers*, C-31(11), November 1982.

[36] D.I. Moldovan and J.A.B. Fortes. Partitioning and mapping algorithms into fixed size systolic arrays. *IEEE Transaction on Computers*, C-35(1):1–12, January 1986.

[37] C. Mongenet. Une méthode de conception d'algorithmes systoliques, résultats théoriques et réalisation. Thèse de 3ieme cycle, 1985.

[38] P. Quinton. Mapping recurrences on parallel architectures. In *Third International Conference on Supercomputing*, Boston (U.S.A), May 1988.

[39] P. Quinton. *The Systematic Design of Systolic Arrays.* Technical Report 193, Publication Interne IRISA, Avril 1983.

[40] P. Quinton and V. Van Dongen. The Mapping of Linear Recurrence Equations on Regular Arrays, Submitted to *The Journal of VLSI Signal Processing*, October 1988

[41] P. Quinton and P. Gachet. *DIASTOL User's Manual- Preliminary Version.* Technical Report 233, Publication interne IRISA, Rennes, France, Aout 1984.

[42] S.K. Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays.* PhD thesis, Standford University, U.S.A., October 1985.

[43] Y. Saouter and P. Quinton. *Computability of Conditional Recurrence Equations.* Technical Report, IRISA - Rennes (France), 1988.

[44] A. Schrijver. *Theory of Linear and Integer Programming. Wiley-Interscience series in Discrete Mathematics*, John Wiley and Sons, 1986.

[45] S.K.Rao. Regular iterative algorithms and their implementations on processor arrays, PhD Thesis, Information Systems laboratory, Standford University, U.S.A., Octobre 1985.

[46] Y. Wong and J.M. Delosme. *Transformation of Broadcasting into Pipelining.* Technical Report, Yale University, Department of Computer Science, June 1987.

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 433 - SPEAKER INDEPENDENT ACOUSTIC-PHONETIC RECOGNITION USING ADAPTATIVE VECTOR QUANTIZATION
Huan-Yu SU
16 Pages, Octobre 1988.

PI 434 - SIMULATING STRUCTURED PLANS FOR TELEROBOTICS USING TEMPORAL LOGICS
Eric RUTTEN, Lionel MARCE
16 Pages, Octobre 1988.

PI 435 - PERFORMANCE MODELLING OF QUEUES WITH RENDEZ VOUS SERVICE
Louis-Marie LE NY, C. Murray WOODSIDE
24 Pages, Octobre 1988.

PI 436 - DIAGNOSING MECHANICAL CHANGES IN VIBRATING SYSTEMS
George MOUSTAKIDES, Michèle BASSEVILLE, Albert BENVENISTE, Georges LE VEY
12 Pages, Octobre 1988.

PI 437 EXTENSION OF CHERNIKOVA'S ALGORITHM FOR SOLVING GENERAL MIXED LINEAR PROGRAMMING PROBLEMS
Felipe FERNANDEZ, Patrice QUINTON,
38 Pages, Octobre 1988.

PI 438 A PROPOS DE LA RESOLUTION D'UN SYSTEME LINEAIRE DANS UN CORPS FINI : ALGORITHMES ET MACHINES PARALLELES
Hervé LE VERGE, Patrice QUINTON, Yves ROBERT, Gilles VILLARD
22 Pages, Novembre 1988.

PI 439 ALPHA DU CENTAUR : A PROTOTYPE ENVIRONMENT FOR THE DESIGN OF PARALLEL REGULAR ALGORITHMS
Pierrick GACHET, Patrice QUINTON, Christophe MAURAS
Yannick SAOUTER
20 Pages, Novembre 1988.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique