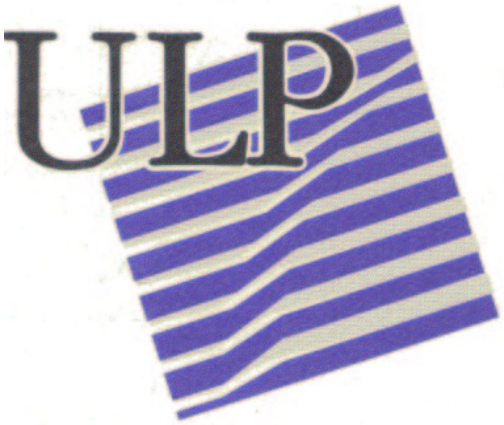

ICPS

Informatique et
Calcul Parallèle
de Strasbourg

Publication 94-12

A mathematical theory and its environment for parallel programming

E. Violard

Published in Dagstuhl Seminar on Parallelization Techniques, to appear in Parallel Processing Letters,1994.

ICPS - Université Louis Pasteur
Pôle API, Boulevard Sébastien Brant, F-67400 Illkirch

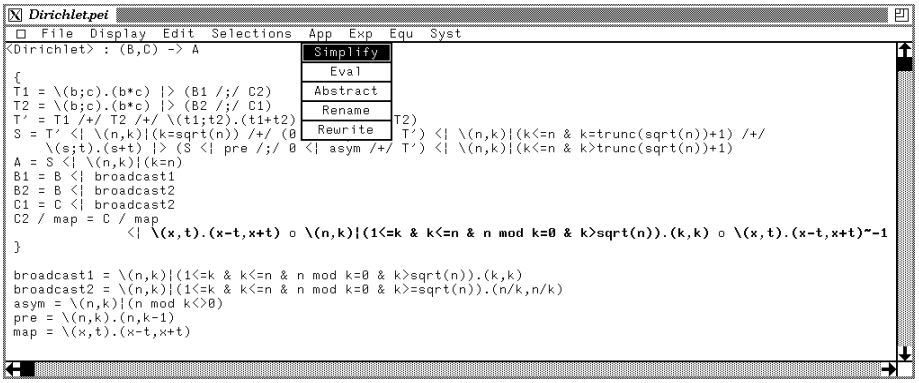

Fig. 10. The PEI environment

1. M. Chen and Y. Choo. Synthesis of a systolic Dirichlet product using non-linear domain contraction. M. Cosnard, Y. Robert, P. Quinton, and M. Raynal, editors, In *Parallel and Distributed Algorithms*, pages 281–295. North-Holland, Oct. 1988.
2. P. Clauss. *Synthèse d'algorithmes systoliques et implantation optimale en place sur réseaux de processeurs synchrones.* PhD thesis, Univ. of Franche-Comté, May 1990.
3. C. Creveuil. *Techniques d'analyse et de mise en oeuvre des programmes* GAMMA. PhD thesis, Univ. of Rennes I, December 1991.
4. INRIA - Rocquencourt. *Maple Reference Manual, 4th Edition*, March 1989.
5. *High Performance Fortran version 1.0*, January 1993.
6. D. Gelernter. Generative communication in LINDA. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.
7. INRIA Sophia-Antipolis, Valbonne. *Centaur 1.1*, 1991.
8. R.M. Karp, R.E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of ACM*, 14(3):563–590, July 1967.
9. C. Mauras. ALPHA *: un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones.* PhD thesis, Univ. of Rennes I, December 1989.
10. P. Quinton. The mapping of linear recurrence equations on regular arrays. *Journal of VLSI Signal Processing*, 1, 1989.
11. S. Rajopadhye. LACS : A Language for Affine Communication Structures. Technical report, IRISA Rennes, 1993.
12. E. Violard and G.-R. Perrin. PEI: A language and its refinement calculus for parallel programming. *Parallel Computing*, 18:1167–1184, February 1992.
13. E. Violard and G.-R. Perrin. PEI : a Single Unifying Model to Design Parallel Programs. A. Bode and M. Reeve, editors, In *LNCS 694, PARLE'93*, pages 500–516, Berlin, June 1993. Springler-Verlag.

```
broadcast1 = \(x,t)|(t>=x+2 & x=0 & (t+x)/2 mod (t-x)/2=0 &
                     (t-x)/2>=sqrt((t+x)/2)) +
             \(x,t)|(t>=x+2 & x>0 & (t+x)/2 mod (t-x)/2=0 &
                     (t-x)/2>=sqrt((t+x)/2))
                    .broadcast1 o right (x,t)
broadcast2 = \(x,t)|(t>=x+2 & x>=0 & (t+x)/2 mod (t-x)/2=0 &
                     (t-x)/2=sqrt((t+x)/2) & t+x=2) +
             \(x,t)|(t>=x+2 & x>=0 & (t+x)/2 mod (t-x)/2=0 &
                     (t-x)/2=sqrt((t+x)/2) & t+x>2)
                    .broadcast1 o right (x,t) +
             \(x,t)|(t>=x+2 & x>=0 & (t+x)/2 mod (t-x)/2=0 &
                     (t-x)/2>sqrt((t+x)/2) &
                     ((t+x)/2 mod ((t-x)/2-1)=0 &
                      (t+x)<>(t-x)*((t-x)/2-1)))
                    .broadcast2 o left (x,t) +
             \(x,t)|(t>=x+2 & x>=0 & (t+x)/2 mod (t-x)/2=0 &
                     (t-x)/2>sqrt((t+x)/2) &
                     ((t+x)/2 mod ((t-x)/2-1)<>0 !!
                      (t+x)=(t-x)*((t-x)/2-1)))
                    .broadcast2 o right (x,t)
asym = \(x,t)|((t+x)/2 mod (t-x)/2<>0)
right = \(x,t).(x-1,t-1)
left = \(x,t).(x+1,t-1)
```

The refinement process is now completed. The whole description of this linear systolic array is given in [1].

6. Conclusion

The theory we have presented in this paper is founded on the simple mathematical concepts of multiset and of an equivalence between their representations as data fields. Program transformations are founded on this equivalence and defined from a refinement relation. Due to the unifying aspect of this theory, solutions that can be reached by these transformations are relevant to various parallel programming models, as systolic processing or data-parallelism.

In this last case, global operations or alignments are modelled through routings or change of basis operations in PEI, while microscopic operations are the functional ones. PEI can then be considered as the mathematical domain for a semantics of data-parallel languages. Transformations and equivalence warrent correct transformations on data-parallel programs.

The mathematical basis of this theory leads to a nice implementation in CENTAUR [7], using MAPLE [4] for formal definitions, of an environment whose purpose is to transform parallel programs (cf. fig. 10).

Step 3: uniformization This step consists in uniformizing the dependencies `broadcast1` and `broadcast2`. These routings can be rewritten in the following way:

```
broadcast1 =
   \(n,k)|(1<=k & k=n & n mod k=0 & k>sqrt(n)) +
   \(n,k)|(1<=k & k<n & n mod k=0 & k>sqrt(n))
         .broadcast1 o translate1 (n,k)
broadcast2 =
   \(n,k)|(1<=k & k<=n & n mod k=0 & k=sqrt(n) & n=1) +
   \(n,k)|(1<=k & k<=n & n mod k=0 & k=sqrt(n) & n>1)
         .broadcast1 o translate1 (n,k) +
   \(n,k)|(1<=k & k<=n & n mod k=0 & k>sqrt(n) &
          (n mod (k-1)=0 & n<>k*(k-1)))
         .broadcast2 o translate2 (n,k) +
   \(n,k)|(1<=k & k<=n & n mod k=0 & k>sqrt(n) &
          (n mod (k-1)<>0 !! n=k*(k-1)))
         .broadcast2 o translate1 (n,k)
translate1 = \(n,k).(n-1,k)
translate2 = \(n,k).(n,k-1)
```

Details of this uniformization are given in [1].

Step 4: scheduling and mapping This last step consists in applying of a change of basis to express the scheduling and the mapping. Classical technics allow to determine the following change of basis:

```
map = \(n,k).(n-k,n+k)            map^-1 = \(x,t).((t+x)/2,(t-x)/2)
```

We obtain the following program:

```
Dirichlet: (B,C) -> A
  {
  B1 = B <| broadcast1
  B2 = B <| broadcast2
  C1 = C <| broadcast2
  C2 = C <| broadcast1

  T1 = \(b;c).b*c |> (B1 /;/ C2)
  T2 = \(b;c).b*c |> (B2 /;/ C1)

  T' = (T1 /+/ T2) /+/ \(t1;t2).t1+t2 |> (T1 /;/ T2)

  S  = T' <| \(x,t)|((t-x)/2=sqrt((t+x)/2)) /+/
       (0 <| asym /+/ T')
          <| \(x,t)|(x>=0 & (t-x)/2=trunc(sqrt((t+x)/2))+1) /+/
       \(s;t).s+t |> (S <| left /;/ (0 <| asym /+/ T'))
          <| \(x,t)|(x>=0 & (t-x)/2>trunc(sqrt((t+x)/2))+1)

  A = S <| \(x,t)|(x=0)
  }
```

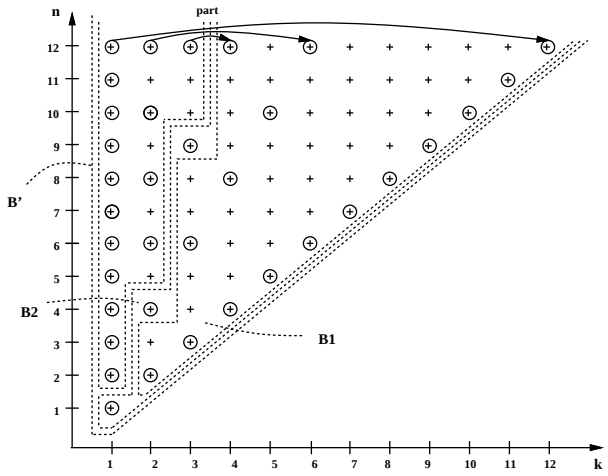

Fig. 9. The fields B1 and B2 and the bijection part

Step 2: sum reduction At this step, we complete the previous program by computing the sum of the data field T values. The idea is to accumulate symmetrical pairs of values along the symmetry defined in step 1. We obtain the following program:

```
Dirichlet: (B,C) -> A
  {
  B1 = B <| broadcast1
  B2 = B <| broadcast2
  C1 = C <| broadcast2
  C2 = C <| broadcast1

  T1 = \(b;c).b*c |> (B1 /;/ C2)
  T2 = \(b;c).b*c |> (B2 /;/ C1)

  T' = (T1 /+/ T2) /+/ \(t1;t2).t1+t2 |> (T1 /;/ T2)

  S  = T' <| \(n,k)|(k=sqrt(n)) /+/
       (0 <| asym /+/ T')
          <| \(n,k)|(k<=n & k=trunc(sqrt(n))+1) /+/
       \(s;t).s+t |> (S <| pre /;/ (0 <| asym /+/ T'))
          <| \(n,k)|(k<=n & k>trunc(sqrt(n))+1)

  A = S <| \(n,k)|(k=n)
  }

asym = \(n,k)|(n mod k<>0)
pre = \(n,k).(n,k-1)
...
```

```
  T = \(b;c).b*c |> (B' /;/ C' <| div)
  }

broadcast = \(n,k)|(1<=k & k<=n & n mod k=0).(k,k)
div       = \(n,k).(n,n/k)
```

We present herebelow the steps to reach a systolic solution:

Step 1: contraction This first step lies on the symmetry of k and n/k along the curve $k = \sqrt{n}$ [1]. It consists in associating the points (n, k) where $k > \sqrt{n}$ with the points (n, k') where $k' < \sqrt{n}$ and $k' = n/k$ when k divides n. This operation is called *contraction* in the theory CRYSTAL [1]. It is defined by a change of basis from Z^q to $\{0, 1\} \times Z^q$. In the present case, this change of basis is defined by the bijection `part` as follows:

```
part = \(n,k)|(1<=k & k<=n & n mod k=0 & k>sqrt(n)).(0,n,k) +
       \(n,k)|(1<=k & k<=n & n mod k=0 & k<=sqrt(n)).(1,n,n/k)
```

The inverse of this bijection is:

```
part^-1 = \(p,n,k)|(1<=k & k<=n & n mod k=0 & p=0 & k>sqrt(n)).(n,k) +
          \(p,n,k)|(1<=k & k<=n & n mod k=0 & p=1 & k>=sqrt(n)).(n,n/k)
```

This change of basis induces a partition of fields `B'`, `C'` and `T`. For example, the field `B'` is partitioned as follows:

```
B' / part = B' / part <| \(p,n,k)|(p=0)  /+/
            B' / part <| \(p,n,k)|(p=1)
```

We define the fields `B1` and `B2` associated with two parts of `B'` as:

```
B1 / \(n,k).(0,n,k) = B' / part <| \(p,n,k)|(p=0)
B2 / \(n,k).(1,n,k) = B' / part <| \(p,n,k)|(p=1)
```

The first step leads to the following program, equivalent to the previous one:

```
Product: (B,C) -> T
  {
  B1 = B <| broadcast1
  B2 = B <| broadcast2
  C1 = C <| broadcast2
  C2 = C <| broadcast1

  T1 = \(b;c).b*c |> (B1 /;/ C2)
  T2 = \(b;c).b*c |> (B2 /;/ C1)
  T  = T1 /+/ T2 <| div
  }

broadcast1 = \(n,k)|(1<=k & k<=n &
                n mod k=0 & k>sqrt(n)).(k,k)
broadcast2 = \(n,k)|(1<=k & k<=n &
                n mod k=0 & k>=sqrt(n)).(n/k,n/k)
...
```

This change of basis defines a part of σ that shows the scheduling of the solution.

At this step we consider that the refinement process is completed. The scheduling and a regular mapping are determined, which describe the following systolic array (drawn for $p = 5$). The routing operation `left` and `right` define the processor links. The routing operation `init` defines a memory cell. The systolic array topology is defined by the set $\{x,\ 1 \leq x \leq p\}$ which is deduced from the data field drawing (cf. fig. 8).

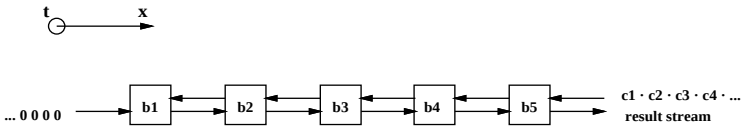

Fig. 8. A processor array for the convolution sum

5. A further example

In the previous section we have considered the convolution sum problem. It is a very classical example in the area of systolic algorithms synthesis for its affine data dependencies. A more complicated example is the Dirichlet product which shows non-affine dependencies.

> Example: The Dirichlet product. Let b and c be two functions defined on N*. Let a be the Dirichlet product of b and c. It is the function defined on N*, for all $n > 0$ by:
>
> $$a_n = \sum_{k|n} b_k \times c_{n/k}$$
>
> where $k|n$ means k divides n.

Thus, theories of parallel program transformation relying on convex manipulations fail on such examples. In [1], the authors emphasize this point and develop a metalanguage to transform CRYSTAL statements. In our theory, such transformations are realized inside the mathematical model itself.

In the following, we present the derivation of a systolic architecture for the Dirichlet product. This derivation of a solution is organized in two steps. First, we consider the computation of the products $b(k) \times c(n/k)$. Then, we deduce an appropriate reduction for the sum of these terms. The field `T` whose values are the terms of the sum can be defined by the following program:

```
Product: (B,C) -> T
  {
  B' = B <| broadcast
  C' = C <| broadcast
```

Step 1: uniformization From this statement, in order to determine a regular mapping of the solution, we rewrite routings `broadcast1` and `broadcast2` as follows:

```
broadcast1 = \(n,k)|(n=1) +
             \(n,k)|(n>1).broadcast1 o translate1 (n,k)
translate1 = \(n,k).(n-1,k)
broadcast2 = \(n,k)|(n=1) +
             \(n,k)|(n>1).broadcast2 o translate2 (n,k)
translate2 = \(n,k).(n-1,k+1)
```

Step 2: scheduling Considering the change of basis operation `map` defined as `\(n,k).(k,2*n+k-1)` (its inverse is `map^-1 = \(x,t).((t-x+1)/2,x)`) and the routing ones in the previous program, the application of rule (6) and of the substitution rule leads to the following equivalent program:

```
Convolution: (B,C) -> A
   {
   B' = B <| broadcast1
   C' = C <| broadcast2

   S  = 0 <| \(x,t)|(x=0)     /+/
        \(s;b;c).s+b*c |> (S <| right /;/ B' /;/ C') <| \(x,t)|(x>0)

   A  = S <| \(x,t)|(x=p)
   }

broadcast1 = \(x,t)|(t=x+1) +
             \(x,t)|(t>x+1).broadcast1 o init (x,t)
broadcast2 = \(x,t)|(t=x+1) +
             \(x,t)|(t>x+1).broadcast2 o left (x,t)
init  = \(x,t).(x,t-2)
left  = \(x,t).(x+1,t-1)
right = \(x,t).(x-1,t-1)
```

that is equivalent to

```
Convolution: (B,C) -> A
   {
   B' = B <| \(x,t)|(t=x+1) /+/  B' <| init <| \(x,t)|(t>x+1)
   C' = C <| \(x,t)|(t=x+1) /+/  C' <| left <| \(x,t)|(t>x+1)

   S  = 0 <| \(x,t)|(x=0)     /+/
        \(s;b;c).s+b*c |> (S <| right /;/ B' /;/ C') <| \(x,t)|(x>0)

   A  = S <| \(x,t)|(x=p)
   }

init  = \(x,t).(x,t-2)
left  = \(x,t).(x+1,t-1)
right = \(x,t).(x-1,t-1)
```

The choice of the order relation $<$ on Z^m predetermines the operational definition of a program. In fact, the aim of the transformations is to make explicit or to build a "nice" bijection σ which introduces the "convenient" order to define a "nice" operational behaviour of the program. These transformations lie on the change of basis operation. It is shown in the following sections devoted to program derivations.

Examples: Let us consider a bijection σ from Z^n to Z^m such as $\sigma(z) = (p(z), t(z))$, where p is a function from Z^n to Z^{m-1} and t a function from Z^n to N. Note that such a definition is a classical way to define a scheduling and a mapping of the computations on a processor set.

- Let $<$ be an order on Z^m such that $\sigma(z) < \sigma(z')$ iff $t(z) < t(z')$ on N. The induced operational definition only defines computations scheduling.
- Let $<$ be an order on Z^m such that $\sigma(z) < \sigma(z')$ iff $p(z) = p(z') \wedge t(z) < t(z')$. The induced operational definition defines computations mapping and the scheduling of the processors.

4. A complete example

The design of a systolic solution for a given problem requires a bijection σ such as $\sigma(z) = (p(z), t(z))$ by determining a convenient change of basis in order to define the computation scheduling t. It may happen that the existence of such a scheduling requires a transformation of the initial equation set to re-arrange the data dependencies. In PEI, these preliminary refinement steps consist in introducing sub-data fields and possibly in changing some basis by applying an equivalence rule. In order to determine a nice mapping function p, next steps may consist in uniformizing the dependencies to reach local routings: this is achieved in PEI by decomposing functional and routing operations. These points are illustrated below with the convolution sum.

Step 0: simplification By applying the classical equivalence rule to change the representation of inputs and outputs, we can simplify the previous program as follows:

```
Convolution: (B,C) -> A
   {
   B' = B <| broadcast1
   C' = C <| broadcast2

   S  = 0 <| \(n,k)|(k=0)     /+/
        \(s;b;c).s+b*c |> (S <| pre /;/ B' /;/ C') <| \(n,k)|(k>0)

   A  = S <| \(n,k)|(k=p)
   }

broadcast1 = \(n,k)|(n>=1).(1,k)
broadcast2 = \(n,k)|(n>=1).(1,k+n-1)
pre        = \(n,k).(n,k-1)
```

Intuitively, we will say that two programs are equivalent if the input data fields and the output data fields are equivalent. This notion of equivalence is formally defined from a refinement relation [12,13].

In practice, one can manipulate structured data as arrays. For example, data field `B` in the convolution could precise its structure by showing the indexes. To make the presentation simple, we have ignored this technical point.

3.5. Transformation rules

Transformation rules come in three types: the first rules are derived from operation properties, the following ones are derived from equations systems and the last ones are equivalence rules.

3.5.1. Operation properties rules

These rules consist in substituting a data field expression for an other one, that can be proved equal from some operation property or from the mathematical structure of data fields set. These rules maintain the equality of programs.

$$f \ \text{|>}\ (f' \ \text{|>}\ \mathtt{X}) = f \circ f' \ \text{|>}\ \mathtt{X} \quad (1)$$

$$(\mathtt{X} \ \text{<|}\ g') \ \text{<|}\ g = \mathtt{X} \ \text{<|}\ g' \circ g \quad (2)$$

$$(\mathtt{X} \ /\ h) \ /\ h' = \mathtt{X} \ /\ h' \circ h \quad (3)$$

$$(f \ \text{|>}\ \mathtt{X}) \ \text{<|}\ g = f \ \text{|>}\ (\mathtt{X} \ \text{<|}\ g) \quad (4)$$

$$(f \ \text{|>}\ \mathtt{X}) \ /\ h = f \ \text{|>}\ (\mathtt{X} \ /\ h) \quad (5)$$

$$(\mathtt{X} \ \text{<|}\ g) \ /\ h = (\mathtt{X} \ /\ h) \ \text{<|}\ h \circ g \circ h\mathtt{\hat{}-1} \quad (6)$$

$$f \ \text{|>}\ (\mathtt{X} \ /\text{+}/\ \mathtt{X'}) = f \ \text{|>}\ \mathtt{X} \ /\text{+}/\ f \ \text{|>}\ \mathtt{X'} \quad (7)$$

$$(\mathtt{X} \ /\text{;}/\ \mathtt{X'}) \ \text{<|}\ g = \mathtt{X} \ \text{<|}\ g \ /\text{;}/\ \mathtt{X'} \ \text{<|}\ g \quad (8)$$

3.5.2. Equations systems rules

These rules are more general than the preceeding ones. They modify not only expressions, but also equations of the program. The transformed system is a new system which has the same solutions: these rules maintain the equality of programs. Substitution or internal operation application are examples of such rules.

3.5.3. Equivalence rules

These rules are more general than the preceeding ones. They transform a program into an equivalent but not forcefully equal one. For example, a classical equivalence rule allows to change the representation of inputs and outputs.

3.6. Operational aspects

Operational aspects define the set of computations associated with some data field definition. This means the definition of an order on the data field elements. This order is a partial order for parallel computations. We define this order, denoted as $\vdash$, for some given partial order $<$ on Z^m, by considering the bijection σ from Z^n to Z^m which characterizes a data field.

Let $\mathtt{X} = (v : \sigma)$ a data field where σ is a bijection from Z^n to Z^m,

$$\forall z, z' \in D_{\mathtt{X}},\ v(z) \vdash v(z') \Leftrightarrow \sigma(z) < \sigma(z')$$

Example: Figure 6 represents a data field `X` and the data field
`X <| \(i,j)|(i=j-1).(i+1,j-1)`

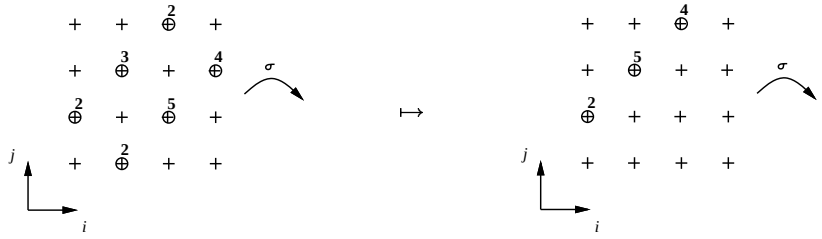

Fig. 6. An example of geometrical operation

Points above the diagonal receive values following vector $(-1, 1)$ while other ones are cleared for they don't occur in the function domain.

3.3.3. Change of basis operation

Let `E` be a data field expression where `E` $= (v : \sigma)$ and $D_\sigma \subset \mathrm{Z}^n$. Let h be a bijection from D_h to Z^m ($D_h \subset \mathrm{Z}^n$). The change of basis operation defines the data field `E / h` as $(v \circ h^{-1} : \sigma \circ h^{-1})$.

Example: Figure 7 represents a data field `X` and the data field
`X / \(i,j).(5-i,j)`

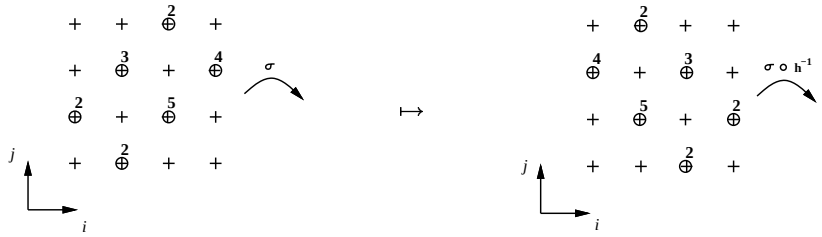

Fig. 7. An example of change of basis operation

3.4. Programs and equivalence

In PEI, a program is a function which applies data fields onto other data fields. Transformations of program in PEI are based on an equivalence of data fields and programs. It is the foundation for our transformational approach to derive solutions from a program statement.

We will say that two data fields are equivalent if they represent the same multiset.

$$\mathtt{X} \equiv \mathtt{X}' \Leftrightarrow M_{\mathtt{X}} = M_{\mathtt{X}'}$$

intersection of the argument drawings, are ordered sequences of values. In PEI, sequences are built using an associative operation denoted as ;. The functions `tail` and `head` are classical functions on sequences. Restrictions of this superimposition operation to disjunction and intersection of drawings are called *sum* and *product* operations. We denote them respectively as `/+/` and `/;/` .

3.3. External operations

External operations either define the computations of the values of a data field, or express data dependencies, or else redraw a data field. These operations apply a partial function onto a data field. According to the way the function is applied, the operation is called a *functional* operation, or a *geometrical* one, or else a *change of basis* operation. The notation PEI for functions is derived from lambda-calculus. When no ambiguity results, any function f of domain $D_f = \{x, P(x)\}$ can be denoted as $\backslash x \,|\, P(x) \,.\, f(x)$. Moreover, we introduce the following simplifications:

$\backslash x \,.\, f(x)$ is equal to $\backslash x \,|\, \texttt{true} \,.\, f(x)$ $\quad$ $\backslash x \,|\, P(x)$ is equal to $\backslash x \,|\, P(x) \,.\, x$

A function f can be defined as a partition f_1 + f_2 + ... + f_n of functions where the f_i are defined on disjunctive sub-domains.

3.3.1. Functional operation (or calculus)

Let `E` be a data field expression whose values are in V and where `E` $= (v : \sigma)$. Let f be a function defined on a subset of V. The functional operation defines the data field f `|>` `E` as $(f \circ v : \sigma)$.

Example: Figure 5 represents a data field `X` and the data field
`\x|(x mod 2=0).x/2 |> X`

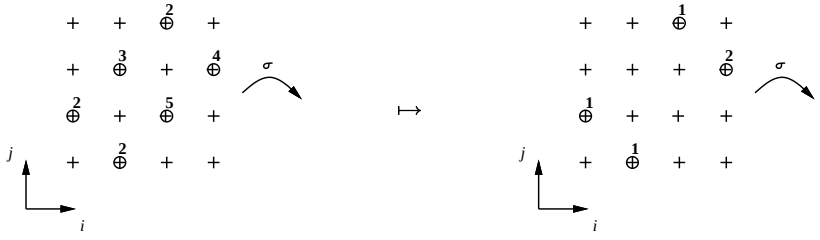

Fig. 5. An example of functional operation

Even values are divided by two while other ones are cleared because they don't occur in the function domain.

3.3.2. Geometrical operation (or routing)

Let `E` be a data field expression where `E` $= (v : \sigma)$ and $D_\sigma \subset \mathrm{Z}^n$. Let g be a function from D_g to Z^n $(D_g \subset D_\sigma)$. The geometrical operation defines the data field `E <|` g as $(v \circ g : \sigma)$.

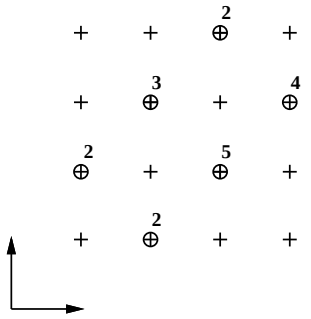

Fig. 3. A geometrical representation of a multiset

is associated with a value of the multiset by this representation. There are infinitely many geometrical representations for a given multiset. They differ from one another by a bijection. This is expressed by the concept of *data field.*

Definition 1 *A data field* X *is the association of a geometrical representation* v *of a multiset and a bijection* σ *such that* $v \circ \sigma$ *is an other representation for the same multiset.*

A data field will be denoted as $(v : \sigma)$. Its representation v is said to be defined within σ. The *drawing* of X, denoted as $D_{\mathtt{X}}$, is the drawing of v. The multiset $\prec v(z), z \in D_v \succ$ associated with X is denoted as $M_{\mathtt{X}}$.

Figure 4 represents an example of data field associated with the previous multiset.

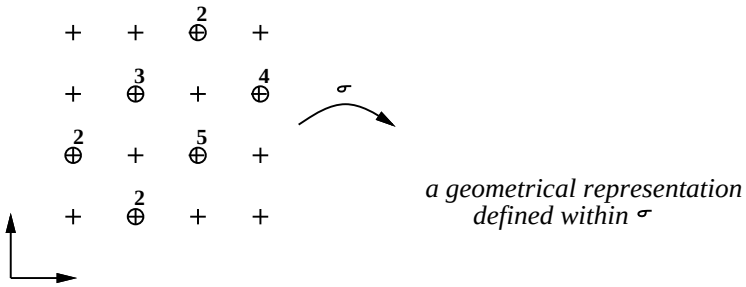

Fig. 4. An example of data field

3.2. Internal operations

Internal operations define the way data fields are built. They are based on a single associative binary *superimposition* operation: the drawing of a data field obtained by applying the superimposition operation is the union of its arguments drawings. The values of the resulting data field, which are associated with the

the change of basis operation `\k.(1,k)` on `C`. In the context, the function `broadcast2` is defined as `\(n,k)|(n>=1).(1,n+k-1)`. It defines a geometrical operation which broadcasts the values of the data field `C / \k.(1,k)` in the direction $(1, -1)$ (cf. fig. 2).

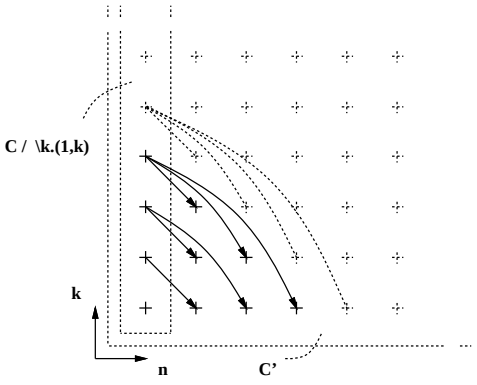

Fig. 2. Broadcast of the field `C` values

The data field `S` is the *sum* of two sub-data fields. The first sub-data field of `S` whose drawing is $\{(n,k),\ k = 0 \ \wedge \ n \geq 1\}$, is a constant data field whose values are 0. The complementary sub-data field of `S` whose drawing is $\{(n,k),\ k > 0 \ \wedge \ n \geq 1\}$, is equal to the application of the functional operation `\(s;b;c).s+b*c` on a *product* of three data fields. The first one is the result of the geometrical operation `pre` equal to `\(n,k).(n,k-1)` on `S` which expresses the dependency $(0, 1)$. The second is the field `B'` and the third is the field `C'`.

Last, within the change of basis `\n.(n,p)`, the data field `A` is the sub-data field of `S` whose drawing is $\{(n,k),\ k = p \ \wedge \ n \geq 1\}$.

3. The theory PEI

3.1. Data fields and equations

Data fields are mathematical structures based on geometrical representations of multisets. Intuitively, a multiset is a set in which an element can appear more than once. A multiset can be denoted between $\prec$ and $\succ$.

A geometrical representation of a multiset consists in associating a geometrical coordinate in Z^n with any value. A *geometrical representation* v of a multiset M, whose values are in V, is a function from Z^n to V, $n \in \mathrm{N}$ such that:

$$M = \prec v(z), z \in D_v \succ$$

where D_v, the domain of v, is called the *drawing* of the representation.

Fig. 3 shows a geometrical representation in Z^2 of the multiset $\prec 5, 2, 2, 4, 3, 2 \succ$. The circled points form the drawing of the representation. Any point of the drawing

Example: The convolution sum (continued). In PEI, the previous definition can be expressed in the following way. Series a and c are represented by data fields `A` and `C` whose drawings are $N^\star$. Series b is represented by data field `B` whose drawing is $\{k,\ 1 \leq k \leq p\}$.

```
Convolution: (B,C) -> A
   {
   B' = B / \k.(1,k) <| broadcast1
   C' = C / \k.(1,k) <| broadcast2

   S  = 0 <| \(n,k)|(k=0) /+/
        \(s;b;c).s+b*c |> (S <| pre /;/ B' /;/ C') <| \(n,k)|(k>0)

   A / \n.(n,p) = S <| \(n,k)|(k=p)
   }

broadcast1 = \(n,k)|(n>=1).(1,k)
broadcast2 = \(n,k)|(n>=1).(1,k+n-1)
pre        = \(n,k).(n,k-1)
```

The data fields `B'`, `C'` and `S` are intermediate data fields. The values of data fields `B` and `C`, drawn in Z^2, are broadcasted to form data fields `B'` and `C'`. The data field `S` values are the intermediate results for the convolution sum.

The data field `B'` is equal to the result of the application of the geometrical operation `broadcast1` on the result of the application of the change of basis operation `\k.(1,k)` on `B`. In the context, the function `broadcast1` is defined as `\(n,k)|(n>=1).(1,k)`. It defines a geometrical operation which broadcasts the values of the data field `B / \k.(1,k)` in the direction $(1, 0)$ (cf. fig. 1).

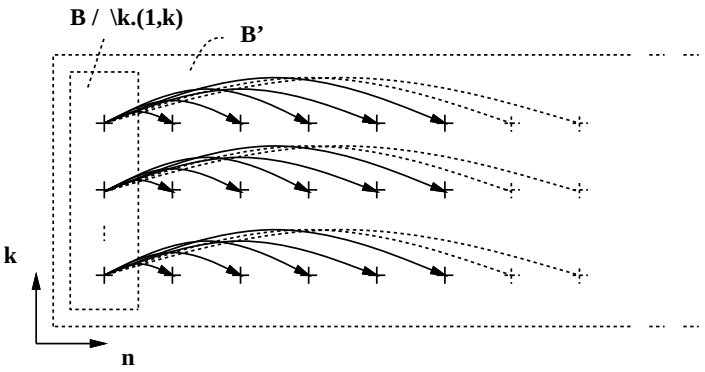

Fig. 1. Broadcast of the field B values

Similarly, the data field `C'` is equal to the result of the application of the geometrical operation `broadcast2` on the result of the application of

of computations can be associated with some data field definition, by ordering its elements: a partial order means parallel computations.

The design of such solutions consists in transforming statements and maintaining the global semantics. These transformations are expressed in PEI too. They are founded on an equivalence relation on data fields.

This paper presents the main concepts of PEI and emphasizes its derivation power. The mathematical basis of this theory leads to an elegant implementation in CENTAUR [7] of an environment whose purpose is to assist in transforming parallel programs. It is illustrated by two related examples: the convolution sum and the Dirichlet product. The second one uses non-affine dependencies that can be easily taken into account using PEI.

2. A short presentation

Some languages such as GAMMA [3] or LINDA [6] are founded on non-deterministic transitions on a multiset of values. The main problem for these formalisms is to define an efficient execution model. Nevertheless they express in a nice abstract way a very large class of problems.

Other languages founded on the recurrence equations concept (ALPHA [9] or CRYSTAL [1] for example), define a denotational semantics of expressions as functions which map index sets to sets of values. Function domains are then built from integral convex polyhedra in Z^n. Domains and values are deduced from equations which define data dependencies.

> Example: The convolution sum. Let b be a series defined on $\{k,\ 1 \leq k \leq p\}$ and c a series defined on $N^\star$. Let a be the convolution sum of b and c. It is the series defined for all $n > 0$ by:
>
> $$a_n = \sum_{k=1}^{p} b_k \times c_{k+n-1}$$
>
> In a recurrence equation form, any $a_n, n > 0$, can be calculated on the domain $\{k,\ 1 \leq k \leq p\}$ of Z by:
>
> $$\begin{cases} s_{n,0} = 0 & (1) \\ s_{n,k} = s_{n,k-1} + b_k \times c_{k+n-1} \quad 1 \leq k \leq p & (2) \\ a_n = s_{n,p} & (3) \end{cases}$$
>
> where $s_{n,k}$ are intermediate results for the convolution sum. The recurrence equation (2) emphasizes uniform dependencies $(0,1)$ for the calculation of $s_{n,k}$.

On the contrary, PEI is a notation for abstract domains, which are geometrical *drawings* of multisets. These mathematical objects are called *data fields.*

A program is a set of equations describing a function whose unknowns are outputs and parameters are inputs. In PEI, each equation `E1 = E2` of a program, specifies that the two data fields `E1` and `E2` are the same mathematical object. The two sides of any equation are expressions composed of data field names and operations on data fields.

A mathematical theory and its environment for parallel programming

Eric Violard

University of Franche-Comté
Laboratoire d'Informatique
F-25030 Besançon cedex
e.mail: violard@comte.univ-fcomte.fr

ABSTRACT

This paper presents the main concepts of the mathematical theory PEI for parallel programming and emphasizes its derivation power. The mathematical basis of this theory leads to a nice implementation in CENTAUR [7] of an environment whose purpose is to transform parallel programs. It is illustrated by two similar examples: the convolution sum and the Dirichlet product. The second one uses non-affine dependencies that can be easily taken into account using PEI.

Keywords: parallel programming, transformation, domains, environment

1. Introduction

Karp, Miller and Winograd [8] presented a convenient framework for programs as a set of *recurrence equations*. The introduction of uniform recurrence equations provided a formal framework for dealing with the classical notions of data dependencies, potential parallelism and computation scheduling. Since their work, many generalizations have been proposed. Affine recurrences on integral convex domains are mainly studied for systolic synthesis [10,2].

More recent developments in data-parallel languages [5] emphazise virtual array structures and global operations such as data alignment and communications. These operations also consider affine transformations and meet more general geometrical models as in LACS [11].

The class of problems that can be adressed by such a modeling suffers from these drastic geometrical constraints. This point can be overcome by defining an abstract structure of domains which are only considered as geometrical drawings of multisets. PEI is a mathematical notation for these domains, called *data fields*. In PEI, names refer to mathematical objects and operations are mathematical operations. A set