A Constraint Optimization Framework for Mapping a Digital Signal Processing Application onto a Parallel Architecture

J. Mattioli, N. Museux, J. Jourdan, P. Savéant, and S. de Givry

THALES, Corporate Research Laboratory
Domaine de Corbeville, 91404 Orsay Cedex
juliette.mattioli@thalesgroup.com

Abstract. In this paper, we present a domain specific optimization framework based on a concurrent model-based approach for handling the complete problem of mapping a DSP application on a parallel architecture. The implementation is based on Constraint Programming and the model is described in details. Our concurrent resolution approach undertaking linear and non linear constraints takes advantage of the special features of signal processing applications. Finally, our mapping tool developped with the ECLAIR solver is evaluated and compared to a classical approach.

1 Introduction

In order to reduce development costs, a major trend in Software Engineering is to follow a strategy of capitalization built on the reuse of software components. This strategy has been adopted in Thales for the development of the planning/optimization functions of Defence and Aeronautics systems. The concrete side of this approach is the design of applicative frameworks dedicated to specific domains and built on the expertise of the company. Such a framework provides an abstract model together with a generic resolution procedure. The development of a specific application is then reduced to a simple customization. The objective of the paper is to describe how this approach was applied to the automatic parallelization of Digital Signal Processing (DSP) applications.
Taking advantage of a multi-processor architecture to speed up a processing which has a potential of parallelization is natural but can become a huge challenge. In the context of Digital Signal Processing (DSP) applications running on a parallel machine with distributed memory, the *mapping problem* can be seen as a scheduling problem with multiple resource allocation where typical objective functions aim at the minimization of:

- the memory capacity,
- the number of processors,
- the response time of the application,
- the bandwith used for communication between processes.

Running an application on such an architecture implies both distributing the code and data on the processors, and scheduling computations and communications. Real life DSP applications run in open loop with a time unit in the order of magnitude of the millisecond, a volume of data in the order of magnitude of the mega byte and consist of thousands of elementary tasks.
The mapping problem has been proved to be NP-complete [10, 21] and is usually decomposed into sub-problems which are solved separately by dedicated algorithms [5] making global optimization impossible. Work based on Integer Programming with Boolean variables led to a combinatorial explosion [21]. A lot of work has been done to optimise local criteria such as data and/or computation distribution locality [15, 6, 13], parallelism level, number of communications [2, 24]. In [11], the scheduling is computed w.r.t. a given partitioning.
Since a few years, THALES in collaboration with Ecole des Mines de Paris open a radically new way by bringing up a concurrent model-based approach to handle the problem as a whole [1, 12, 16]. Since then, this model has been implemented with contraints in finite domains using ECLAIR ©, the THALES constraint solver. Today an application framework dedicated to the mapping of a DSP application onto a "parallel" machine is available where the target architecture can be specified as well as the type of outputs for code generation. Our objective has been to provide specialists with an interactive tool for application domains that involve signal processing: radar[27], sonar[12] or telecom[3]. The framework can be specialized for different types of:

- architecture: SIMD, MultiSPMD, MIMD,
- processors network topology: fully connected, ring based,
- memory: simple, multiple level,
- computer: mainframe (100 processors) [12], on-board [27], system-on-chip [3].

All this flexibility supposes a high degree of modularity and we will try to show in this paper how this goal is met with Constraint Programming.
Several tools that aim at mapping applications onto a multi-processor architecture are presently available as research prototypes or commercial off the shelf products. CASCH, FX, GEDAE [23], PTOLEMY [35], SYNDEX [33], TRAPPER [30] are tools of this type. Each tool has its own features but none of them allows to simultaneously take into account all the architecture and applicative constraints in a global optimization process.
Mapping in a deterministic way a DSP application with specific signal requirements [17, 34] have been widely investigated. The representative PTOLEMY framework [22, 32, 25] brings some solutions but at a coarse grain level.

2 Architectural and Application features

A DSP application is decomposed in tasks and computational dependencies are stated by a data flow graph.
The control structure of a task is restricted to a set of perfect nested loops (like russian dolls). Each "loop nest" encapsulates a call to a procedure such as for

instance a Fast Fourier Transform. These procedures work on arrays and are the elementary tasks w.r.t. parallelization and are thus considered as black boxes. The source of parallelism comes from the following properties of the procedures:

- single-assignment form: only one writing operation in an array can occur,
- each loop index is associated to a unique dimension of an array,
- there are no read/write dependencies [4].

Therefore all permutations of loops in a nest are equivalent. Note that parallelization is maximal since any elementary iteration can be done separately. Finally, a DSP application is a system in open loop which is fed periodically. This is captured by introducing an infinite dimension.
A toy example composed of three tasks is given in Figure 1. Tasks are described in a pseudo-language which supposes infinite loops and arrays with infinite dimensions. Tasks precedences can be inferred by the fact that the Sum Task needs TAB23 which is computed by Diff Task. Diff Task needs TAB12 computed by Square Task.

Square Task: C
```
DO I=0,INFINITE
  DO J=0,7
    TAB12[I,J]=
        TAB1[I,J]*TAB1[I,J]
  ENDDO
ENDDO
```

Diff Task: D
```
DO I=0,INFINITE
  DO J=0,3
    TAB23[I,J]=
        TAB12[2*I,J]-TAB12[2*I+1,J]
  ENDDO
ENDDO
```

Sum Task: I
```
DO I=0,INFINITE
  S=0
    DO J=0,3
      S=S+TAB23[I,J]
    ENDDO
  TAB3[I]=S
ENDDO
```

Fig. 1. A simple DSP application defined by a sequence of 3 loop nests

The target architecture considered here is an abstract Single Program Multiple Data (SPMD) distributed memory machine. In such an architecture, all processors are executing the same elementary task on different data at the same time. The architecture is defined by:

- the network topology,
- the number of processors,
- the memory capacity of each processor,
- the type of memory (hierarchical, circular buffering)
- the clock rate of each processor,
- the communication bandwith,
- the type of communication (point to point, pipeline, block by block)

In the following we have chosen a fully connected topology where all processors are connected to each other in order that communication duration depends only on the size of the data, and not on the position of the processors. Under this assumption explicit processor assignment can be ignored. In addition it is assumed that a communication and a computation can be done simultaneously on one processor.

3 The mapping model

The mapping problem is decomposed in a set of concurrent models as shown in Figure 2:

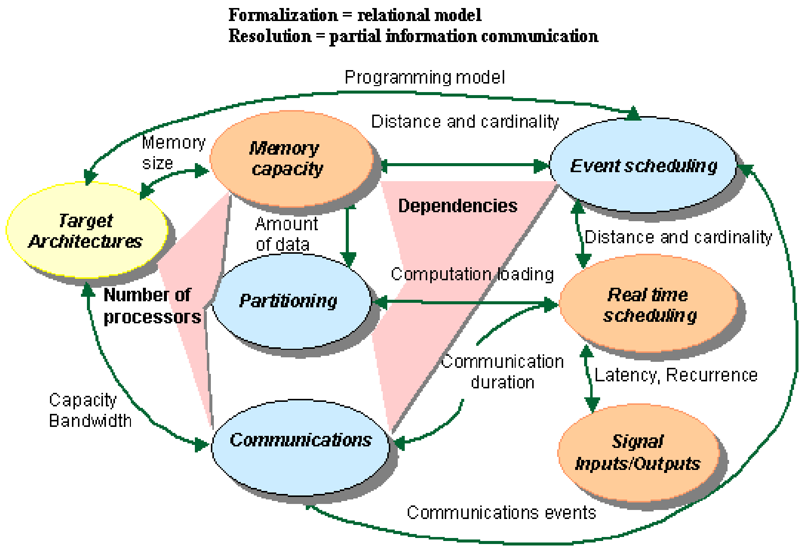


Fig. 2. The concurrent modeling view of the mapping problem

A model has to be viewed semantically as the set of formal specifications [19, 20] of the behaviors of the (functional or physical) sub-problem components. In the mapping context, we have the following models:

memory capacity : ensures the application executability under a memory constraint. A capacitive memory model is used. It evaluates the memory required for each computational block mapped onto a processor.

partitioning : controls the distribution of data onto processors.

communications : schedules the communications between processors.

event scheduling : associates to each computational block a logical execution event on a processor.

real time scheduling : schedules tasks and communications taking into account computation and communication duration and their overlapping.

signal inputs/outputs : the signal is characterized by two values: the input signal *recurrence*, i.e. the time between two consecutive input signals and the *latency*, i.e. an upper time bound for the production of the results on an input signal.

dependencies : express that a piece of data of loop nest cannot be read before being updated by the corresponding written piece of loop nest.
number of processors : defines the available processors.
target architecture : defines the class of which the target architecture belongs: SIMD, MIMD,...

A model, represented in fig. 2 by a bubble, is viewed as a set of variables and a set of constraints over them. All the constraints of each model have been defined separately. The relations between models are either constraints or defined predicates, and are represented by arcs or hyper-arcs in fig. 2.
The modeling phase consists in axiomatizing the behavior using the properties and relations of all the different components. Consequently a model is identified to the set of relations defined on its interface variables. The relations are either predefined constraints or user defined predicates. The variables have to be considered as *access ports* to the model. Thus, model coordinations can be achieved either by unifying some ports of several models together or by involving ports of different models in a relation. Each variable takes part in a global cross-model composite solving, such that only relevant information is exchanged between models. A global resolution mechanism (search) looks for partial solutions in the different concurrent models. For instance, the set of scheduling and target machine variables are partially instantiated by inter-model relations during the resolution. The search relies on the semantic of the different variables involved in each model and their importance according to other different models as well as in regards of the goal to achieve (e.g. resources minimization). Model-specific or more global heuristics are used to improve the resolution. For instance, computing the shortest path in the data-flow graph drives good schedule choices.
The concurrent model based approach matches directly the constraint programming paradigm which provides a concurrent model of solving.
Due to space limitation, only the partitioning, scheduling and memory models[16] are presented in the following sub-sections. But the communication, latency, architectural and applicative models obviously influence the resolution.

3.1 The Data-Partitioning model

The data-partitioning controls the application parallelism level, memory location requirement and event scheduling parameters. Its model is designed to distribute elementary tasks onto the target architecture without resource and real time scheduling considerations. Since DSP applications are sequences of parallel loop nests, the partitioning problem results in a nest by nest partitioning [1].

Due to DSP application features (presented in §2), only the multidimensional iteration domain $\mathcal{I}$ is partitioned. This domain is defined by the Cartesian product of each iteration domain of the loop.
In the example given in fig. 1, the iteration domain of the Square Task, is given by: $\mathcal{I} = dom(I) \times dom(J) = [0..\infty[\times [0..7]$.

[1] Here, we use the word *partitioning* in the mathematical sense.

The iteration domain is projected on a three-dimension space. For that, the iteration domain is decomposed in 3 vector parameters: c, p, l where c represents the cyclic recurrence, p a processor and l a local memory area. This projection gives a hierarchical definition of the partioning model: at one time c, p processors are used and each of them uses a local memory area l. This implies that all vectors of iterations $i \in \mathcal{I}$ is constrained by

$$i = LPc + Lp + l \tag{1}$$

where P and L are variable diagonal square integer matrices involving respectively processor distribution and memory data location. Diagonal matrices L and P (resp. vectors c, l, p) are lists of variables defined in ECLAIR by

```
DMATRIX :: list[Var]
VECTOR  :: list[Var]
VECTOR+ :: (list[Var U {infinity}])
```

Equation (1) induces $i_{up} = L.P.c_{up}$ if the target machine is in a SIMD programming mode and $c_{up} = \left\lceil \frac{i_{up}}{L.P} \right\rceil$ if the target machine is in a SPMD programming mode. Then we have, in the SIMD case, the following constraints:

```
let lb  := (if (lpartition[NbTache].upC[1] = infinity) 2 else 1)
in  (for i in (1 .. lb - 1) LP[i] = L[i] * P[i],
       for i in (lb .. length(UpI))
         (LP[i] = L[i] * P[i],
          LP[i] <= UpI[i],
          UpI[i] = UpC[i] * LP[i])
```

Vector l (resp. p) is constrained by $\forall l, 0 \leq L^{-1}l < 1$ (resp. $\forall p, 0 \leq P^{-1}p < 1$) which means that there is no computation replication on the processors (resp. over the time). One iteration on memory is modeled [2] by $det(L) \neq 0$ and $det(P) \neq 0$ which means that there is at least one processor. At least, since matrices L and P are diagonal, new constraints are induced:

- $\prod_i L_{ii}$ which is the number of local iterations executed by one processor at each cycle c (each local iteration execute a procedure call);
- $\prod_i P_{ii}$ which gives the maximum number of processors and
- the following variable $\max(c)$ represents the maximum number of synchronization (cycles) necessary for the loop nest completion.

For example, a partitionning for application defined on fig.1 could be:

Square : $P = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix}$ and $L = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Which means that the finite dimension J is mapped onto 4 processors, and only one data element is distributed at each cycle c.

[2] The determinant of a m by m matrix A equals the volume of a parallelepiped K in m-dimensional space, provided the edges of K come from the rows of A, or the edges could form the columns of A, giving an entirely different parallelepiped with the same volume.

Diff : $P = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix}$ and $L = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ **Sum :** $P = \begin{pmatrix} 1 \end{pmatrix}$ and $L = \begin{pmatrix} 1 \end{pmatrix}$

3.2 The Scheduling model

The scheduling model is designed to associate each partitioned subset of elementary tasks, a logical execution event on a processor. This schedule is an event schedule, that defines only a total order between tasks.
Since we are in a cyclic applicative context, an affine scheduling approach, used in parallelization techniques [14], has been applied. In terms of constraints, we post:

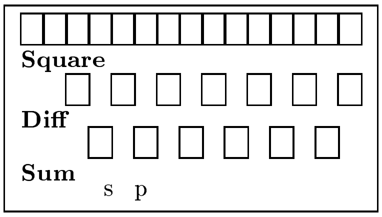
$$d^k(c^k) = \alpha^k \cdot c^k + \beta^k \tag{2}$$

Thanks to the SPMD architecture, all processors work on the same volume of data at each time unit. That is why the link with the partioning model is its cyclic recurrence c.
Variables are indexed by the loop nest number k. d^k is the scheduling event date of the k^{th}-loop nest; α^k and β^k are the scheduling affine variables, where α^k is a line vector, and β^k is scalar. (2) is implemented in Eclair by:

```
[d(alpha:VECTOR, beta:Var, C:VECTOR) : Var
-> scalar(alpha /+ list(beta), C /+ list(ONE))]
```

For example, an event schedule for the example defined on fig. 1 could be the one of fig. 3.2:

Square : $\alpha^1 = (1, 1)$, $\beta^1 = 0$
Diff : $\alpha^2 = (2, 1)$, $\beta^2 = 2$
Sum : $\alpha^3 = (2)$, $\beta^3 = 3$



Symbol s denotes the start-up and p represents the beginning of periodic schedule.

Fig. 3. Bi-dimentionnal Chronogram on the application defined in fig. 2

3.3 The Data Flow dependencies model

The relation (represented by an hyper-arc on fig. 2) that links the partitioning and scheduling models is the *data flow dependencies*. It expresses that a piece of data of the loop nest N^r cannot be read before being updated by written loop nest N^w. These dependencies between two cycles c^w (written cycle) of loop nest N^w and c^r (readen cycle) of N^r imply that:

$$\forall(c^w, c^r)\ Dependencies(c^w, c^r) \Rightarrow d^w(c^w) + 1 \le d^r(c^r) \tag{3}$$

d^w (respectively d^r) is the schedule associated to N^r (resp. N^w). These dependencies enforce a partial order on parallel program instructions for guarantying the same result to the sequential program. Note that these dependencies are computed between iterations of *different* loop nests. All the dependency relationship between blocks of computation can not be stated in the original constraint (3) due to the universal quantifier over the data flow dependency universal predicate: $\forall(c^w, c^r), Dependencies(c^w, c^r)$.
Due to DSP characteristics, data flow dependencies universal predicate is characterized by a set of integer points belonging to a Cartesian product of polygons called the dependency polygon. Furthermore, thanks to the convexity property of this polygon [31], the data flow dependency constraint (3) has been encoded as constraint (4):

$$\forall(c_s^w, c_s^r) \quad d^w(c_s^w) + 1 \leq d^r(c_s^r) \tag{4}$$

where (c_s^w, c_s^r) are the vertices components of the integer convex hull of the dependency polygon, that have been computed symbolically. Hence, the scope of this $\forall$ is *narrower* than in constraint (3). Unfortunately, these vertices are rational and data flow dependencies are approximated by their *integer* convex hull representation. Since the coordinates of the convex hull vertices are given by constraints, we can't use the Harvey convex hull algorithm [18].
This approximation allow us to obtain the same set of valid schedules as with the exact representation but with an impact [3] in the parallel generated code. For reducing the impact, the data flow dependencies are characterized by the smallest convex hull which vertices are integer. This convex hull is defined through a `gcd` constraint [26].

3.4 The target SMPD architecture model

Let N is be the number of loop nests. It is used over the scheduling constraint (2) with the offset $+k$ in order to avoid the execution at the same date of two computations belonging to different loop nests. Then the scheduling model is transformed for taking into account the SPMD architectural feature, and we obtain a SPMD specific schedule:

$$d^k(c^k) = N(\alpha^k \cdot c^k + \beta^k) + k$$

In the same way, two computational blocks of a single loop nest cannot be executed at the same date. Let c_i^k and c_j^k with $i < j$ be two cyclic components of the partitioned loop nest N^k. Then, the execution period of cycle c_i^k must be greater than the execution time of all cycles c_j^k. Hence, constraints: $\alpha_i^k > \sum_{j>i} \alpha_j^k \max(c_j^k)$ with $\alpha_n^k \geq 1$ must be verified.

3.5 The Memory capacity model

The memory model ensures the application executability under a memory constraint. Since the capacity of the memory on each processor is limited, it is

[3] In some case, the generated code is not optimal in sense of number of lines.

necessary to make sure that the memory used by the data partitioning do not exceed its resources [12]. A capacitive memory model is used and this model is based on a kind of producer/consumer constraint closely related to a capacity constraint. It evaluates the memory required for each partitioned elementary tasks block mapped onto a processor by analyzing the data dependencies. The number of data needed to execute an elementary tasks block is computed. Due to the partitioning model all elementary tasks blocks have the same simple structure and the same size. Data dependencies are used to determine the data block lifetime. For each block, the schedule and data dependencies give the maximum lifetime of involving data and the number of data creations during one cycle. This gives the required memory capacity per elementary tasks block and cycle.

4 APOTRES: a mapping framework

In order to assist specialists on parallel machines with distributed and hierarchical memory levels for DSP applications, a mapping framework called APOTRES[4] for rapid DSP prototyping has been developed. Thanks to the concurrent model-based approach, each model defines a modular component of the mapping framework. For example, if an architectural feature is required, a new model will be designed and relations with the other models will be refined.

4.1 Eclair Solver

Eclair © is a finite domain constraint solver over integers written in the Claire functional programming language [9, 8]. The current release includes arithmetic constraints, global constraints and boolean combinations.
Eclair [7, 28] provides a standard labeling procedure for solving problems and a branch-and-bound algorithm for combinatorial optimization problems. Programmers can also design easily their own non-deterministic procedures thanks to the dramatically efficient trailing mechanism available in Claire.
Eclair can be embedded in a real time system. A package has been developed to take into account time management and memory allocation with the introduction of interrupt points.
Eclair has been used mainly in the domain of weapon allocation, weapon/sensor deployment and the parallelization of DSP applications (the topic of this paper). An open source version is available at:

http://www.lcr.thomson-csf.com/projects/openeclair

The non-linear constraints appearing in the partitioning model and in the scheduling model rely on the *type reduction* implementation scheme presented in [29]. This approach makes effective the reduction of complex constraints into simpler ones.

[4] APOTRES is the french acronym of "Aide au Placement Optimisé pour application de Traitement Radar Et Sonar" which means "Computer-assisted mapping framework for Radar and Sonar applications" and is protected by a patent.

4.2 Search procedure

The optimization procedure is a classical Branch-and-Bound algorithm. An enumeration is performed for each model according to its decision variables and for which there are specific strategies and heuristics. In our context, two enumerations are required to find a solution of the whole problem. The first one concerns the partitionning and consists in trying all possible mappings for the data. The second one is related to scheduling where the goal is to order the tasks.

4.3 The user interface

After loading the DSP application, the user specifies through a graphical interface (cf. fig. 4):

- the target machine through the parametrization of the number of processors, the bound of memory capacity, the bandwith and the frequency clock;
- the optimization criteria, if the user wants to use the system in order to get for example the smallest number of processors, the smallest amount of memory, the smallest latency and/or the cheapest architecture.

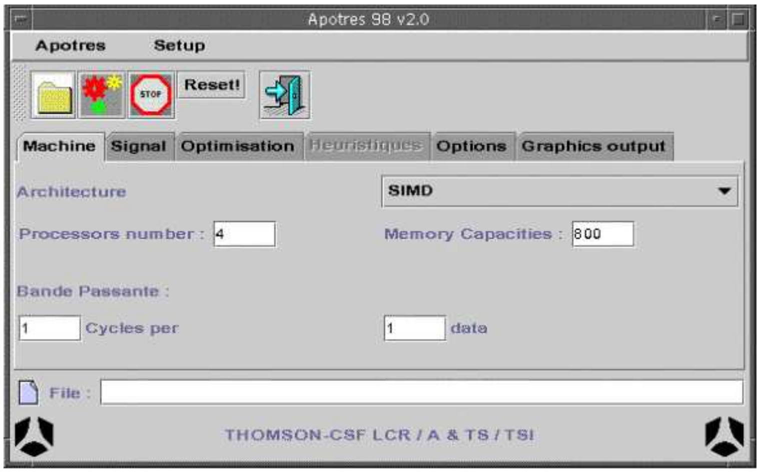


Fig. 4. The user graphical interface

Several use modes are possible:

- The system finds automatically an optimal mapping or a mapping near a percentage of the optimum. (In this case, a complete algorithm is used.)
- Another possibility is to find (if it is possible) a solution of a given partial mapping, that allow to enforce a specific schedule or a specific data partitioning. The search stops after finding the first solution.

– It is also a mapping verification system. The user can instantiate all the mapping variables and the result will be a "yes/no"-answer.

There are graphical user interfaces for visualizing the data partitionning, the schedule, the task/communication overlapping, and finally the tool can generate a LaTeX report (or html report), in order to get all the mapping directives that allow the target machine compiler to generate the parallel code.

5 An industrial validation

Our tool has been evaluated successfully on several THALES DSP benchmarks.

5.1 A simple example of a mapping solution

We present in this section the results on the application described in Fig.1. In this example, the optimizing cost function is the *latency minimization*. The target machine has 4 processors. The capacity memory constraint is set to 200 memory elements (8,16,32 or 64 bits).
The latency optimum is reached (and proved by the completeness of the search algorithm). Its value is 4 cycles for a memory capacity of 64 memory elements. The table of fig.5 gives latency and memory values at each step of the search. The diagram of fig.5 describes partitioning and event scheduling of the optimal solution, and arrrows represent the data flow dependencies.

Optimization criteria latency minimization			
no	# proc.	memory element	Latency cycle
0	4	200	12
1	4	80	8
2	4	64	7
3	4	64	4

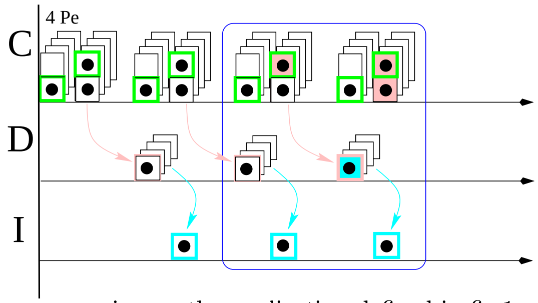


Fig. 5. The optimal latency mapping on the application defined in fig.1

5.2 Validation on real DSP applications

To evaluate the approach, we have compared the solutions found with Apotres to solutions found by experts on real DSP applications [12]. We present in this section the results on the Panoramic Analysis application described in fig. 6 and fig. 7.
In this application, the optimizing cost function is the *memory size minimization*. The target machine has 8 processors. The latency constraint is set to 4.10^8

```
doall r,c
  call FFT(r,c)
enddo
doall r,f,v
  call BeamForming(r,f,v)
enddo
doall r,f,v
  call Energy(r,f,v)
enddo
doall r,v
  call ShortIntegration(r,v)
enddo
doall r,v
  call AzimutStabilization(r,v)
enddo
doall r,v
  call LongIntegration(r,v)
enddo
```

Fig. 6. *Panoramic Analysis* application

```
do r=0,infinity
 do c=0,511

  c   Read Region:
  c        SENSOR(c,512*r:512*r+511)
  c   Write Region:
  c        TABFFT(c,0:255,r)

  call FFTDbl(SENSOR(c,512*r:512*r+511),
              TABFFT(c,0:255,r))

 enddo
enddo
```

Fig. 7. FFT Loop nest

processor clock cycles and the memory is unbounded. Figure 8 describes the partitioning and the schedule found by Apotres.

The partitioning characteristics follow. (1) Only finite dimensions are mapped onto the 8 processors. (2) The write region of the second loop nest is identical to the read region of the third loop nest. So the system fuses these loop nests in order to reduce memory allocation. (3) The access analysis of the second and third loop nests presents read region overlaps between successive iteration execution. This overlap is detected. The system parallelizes according to another dimension to avoid data replication.

According to the different partitions, only the time dimension is globally scheduled. From the α and β scheduling parameters in Figure 8, the schedule can be expressed using the regular expression:

$$(((FFT, [BF, E], BB)^8, SI, SA)^8, LI)^\infty$$

The system provides a fine grain schedule at the procedural level using the dependence graph shortest-path. This enables the use of data as soon as possible, avoids buffer allocation, and produces output results at the earliest. Eight iterations of Tasks `FFT,BF-E,BB` (executed every $\alpha_1^1 = 6$ steps) are performed before one iteration of `SI,SA` (executed every 48 = 6*8 steps). The last task `LongInteg` cannot be executed before 8 iterations of the precedent tasks. So it is executed every 384 (=8*48) steps.

Manual mappings of DSP applications are very difficult because finding an effective mapping imperatively requires to take into account both architectural resources constraints and real time constraints and of course the resulted mapping program must return the same result as the sequential program. We have

Partitionning	FFT	$Beam\ Forming$ $Energy$	$BroadBand$	$Sht\ Integ$	$Azimut$	$LongInteg$
$Parallelism,\ P =$	$\begin{pmatrix} 1 & 0 \\ 0 & 8 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 8 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 8 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 8 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 8 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 8 \end{pmatrix}$
$Locality,\ L =$	$\begin{pmatrix} 1 & 0 \\ 0 & 64 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 128 & 0 \\ 0 & 0 & 25 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 16 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 16 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 16 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 16 \end{pmatrix}$
Scheduling	FFT	$Beam\ Forming$ $Energy$	$Broad\ Band$	$Sht\ Integ$	$Azimut$	$Long\ Integ$
α	$\begin{pmatrix} 6 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 6 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 6 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 48 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 48 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 384 \\ 1 \end{pmatrix}$
β	0	1	2	45	46	383

Fig. 8. Partitioning and Scheduling matrices for Panoramic Analysis

compared our solution to two different manual solutions. The first one is based on loop transformation techniques. The second one uses the maximization of the processor usage as economic function. Our result is equivalent to the one suggested by parallelization techniques. It is better than the second one which requires more memory allocation.

6 Conclusions

This work illustrates the applicability of the concurrent model-based approach to the resolution of problems of multi-function and multi-component systems through a domain specific framework. This approach transforms the difficulty of dealing with the whole system into the advantage of considering several models concurrently. It also allows the design of a mapping framework dedicated to parallel architectures and DSP applications. The relevance of using CP languages for solving the complex problem of automatic application mapping on parallel architectures has been shown.

In this paper, we focused on SPMD architecture, but our system is currently being extended in order to remove different restrictions such as considering more complex mapping functions[27, 3], considering other architectures (Multi-SPMD, MIMD machines).

Moreover, we give a new alternative for the automatic determination of array alignment and task scheduling on parallel machines opening a radically new way to tackle parallelization problems. For some complex DSP application such as Radar applications, a manual mapping which preserves all constraints, costs about 6 months of effort. The major benefit of our system is that it gives a first solution in few minutes and thus reduces the development time cost.

Acknowledgments

We are very grateful to Pr. F. Irigoin and Dr. C. Ancourt for their permanent help on the modeling phase and to P. Gérard for its fruitful comments.

References

1. C. Ancourt, D. Barthou, C. Guettier, F. Irigoin, B. Jeannet, J. Jourdan, and J. Mattioli. Automatic mapping of signal processing applications onto parallel computers. In *Proc. ASAP 97*, Zurich, july 1997.
2. J.M. Anderson and M.S. Lam. Global optimizations for parallelism and locality on scalable parallel machines. In *SIGPLAN Conf. on Programming Language Design and Implementation*, pages 112–125, Albuquerque, NM, June 1993. ACM Press.
3. M. Barreteau, J. Mattioli, T. Granpierre, Y. Sorel C. Lavarenne and, P. Bonnot, P. Kajifasz, F. Irigoin, C. Ancourt, and B. Dion. Prompt: A mapping environnment for telecom applications on *System-On-a-Chip*. In *Compilers, Architecture, and synthesis for embedded systems*, pages 41–48, november 2000.
4. A. J. Bernstein. Analysis of programs for parallel processing. *IEEE Trans. on El. Computers*, EC-15, 1966.
5. S. S. Bhattacharyya, S. Sriram, and E. A. Lee. Latency-Constrained Resynchronisation For Multiprocessor DSP Implementation. In *Proceedings of ASAP'96*, 1996.
6. E. Bixby, K. Kennedy, and U. Kremer. Automatic Data Layout Using 0-1 Integer Programming. In *Proc. of the International Conference on Parallel Architectures and Compilation Techniques*, August 1994.
7. Y. Caseau, F. Josset, F. Laburthe, B. Rottembourg, S. de Givry, J. Jourdan, J. Mattioli, and P. Savéant. Eclair at a glance. Tsi / 99-876, Thomson-CSF/LCR, 1999.
8. Yves Caseau, François-Xavier Josset, and François Laburthe. CLAIRE: Combining Sets, Search and Rules to better express algorithms. In *Proc. of ICLP'99*, pages 245–259, Las Cruces, New Mexico, USA, November 29, December 4 1999.
9. Yves Caseau and François Laburthe. *Introduction to the* CLAIRE *programming language - Version 2.4.0.* Ecole Normale Supérieure - DMI, www.ens.fr/~caseau/claire.html, 1996-1999.
10. A. Darte. On the complexity of loop fusion. *Parallel Coomputing*, 26(9):1175–1193, August 2000.
11. A. Darte, C. Diderich, M. Gengler, and F. Vivien. Scheduling the computations of a loop nest with respect to a given mapping. In *Eighth International Workshop on Compilers for Parallel Computers*, CPC2000, pages 135–150, january 2000.
12. A. Demeure, B. Marchand, J. Jourdan, J. Mattioli, F. Irigoin, C. Ancourt, and all. Placement automatique optimisé d'applications de traitement du signal. Technical report, Rappport Final DRET 913060009, 1996.
13. M. Dion. *Alignement et distribution en parallélisation automatique.* Thèse informatique, ENS,LYON, 1996. 136 P.
14. P. Feautrier. Some efficient solutions to the affine scheduling problem, part ii : mutidimensional time. *International Journal of Parallel Programming*, 21(6):389–420, december 1992.
15. P. Feautrier. Toward Automatic Distribution. *Parallel Processing Letters*, 4(3):233–244, 1994.

16. Ch. Guettier. *Optimisation globale et placement d'applications de traitement de signal sur architectures parallèles utilisant la programmation logique avec contraintes.* PhD thesis, Ecole des Mines de Paris, 1997.
17. C. Han, K.-J. Lin, and C.-J. Hou. Distance Constrained Scheduling and its Applications to Real-Time Systems. *IEEE Transactions On Computers*, 45(7):814–825, Jul 1996.
18. W. Harvey. Computing two-dimensional integer hulls. *Society for Industrial and Applied Mathematics*, 28(6):2285–2299, 1999.
19. J. Jourdan. *Concurrence et coopération de modèles multiples dans les langages de contraintes CLP et CC : Vers une méthodologie de programmation par modélisation.* PhD thesis, Université Denis Diderot, Paris VII, février 1995.
20. J. Jourdan. Concurrent constraint multiple models in clp and cc languages: Toward a programming methodology by modelling. In *Proc. INFORMS conference*, New Orleans, USA, October 1995.
21. U. Kremer. NP–completeness of Dynamic Remapping. In *Workshop on Compilers for Parallel Computers, Delft*, pages 135–141, December 1993.
22. E. A. Lee and D. G. Messerschmitt. Synchronous Dataflow. In *Proceedings of the IEEE*, September 1987.
23. Lockheed Martin. *GEDAE Users' Manual / GEDAE Training Course Lectures.*
24. B. Meister. Localité des données dans les opérations stencil. In *Treizième Rencontres Francophones du Parallélisme des Architectures et des Systèmes*, Compilation et Parallélisation automatique, pages 37–42, avril 2001.
25. P. Murthy, S. S. Bhattacharyya, and E. A. Lee. Minimising Memory Requirements for Chain-Structured Synchronous Dataflow Programs. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing, April 1994.*, 1996.
26. N. Museux. De la sur-approximation des dépendances. Technical Report E/227/CRI, ENSMP/CRI, 2000.
27. N. Museux, F. Irigoin, M. Barreteau, and J. Mattioli. Parallélisation automatique d'applications de traittement du signal sur machines parallèles. In *Treizième Rencontres Francophones du Parallélisme des Architectures et des Systèmes*, Compilation et Parallélisation automatique, pages 55–60, avril 2001.
28. Platon Team. Eclair reference manual. Technical report, THALES/LCR, 2001.
29. P. Saveant. Constraint reduction at the type level. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, Singapore, 2000.
30. L. Schäfers and C. Scheidler. trapper: A graphical programming environment for embedded mimd computers. In IOS Press, editor, *1993 World Transputer Congress*, Transputer Applications and Systems'93, pages 1023–1034, 1993.
31. M. Schmitt and J. Mattioli. Strong and weak convex hulls in non-Euclidean metric: Theory and Application. *Pattern recognition letters*, 15:943–947, 1994.
32. Gilbert C. Sih and Edward A. Lee. Declustering: A New Multiprocessor Scheduling Technique. *IEEE Transaction on Parallel and Distributed Systems*, 4(6):625–637, June 1993.
33. Y. Sorel and C. Lavarenne. http://www-rocq.inria.fr/Syndex/pub.htm.
34. J. Subhlok and G. Vondran. Optimal latency-troughput tradeoffs for data parallel pipelines. In *Proc. SPAA'96*, Padua, Italy, 1996.
35. E. A. Lee team. http://ptolemy.eecs.berkeley.edu/papers.