RENPAR/CFSE/SYMPA
Paris, cité des sciences et de l'industrie, 24 – 27 avril 2001

Placement automatique d'applications de traitement du signal sur machines parallèles

Nicolas Museux * †, François Irigoin *, Michel Barreteau †, Juliette Mattioli †

* ENSMP / CRI,
13, rue St Honoré, 77305 Fontainebleau, France,
e-mail : <*Nom*>@ cri.ensmp.fr

† THALES / LCR,
Domaine de Corbeville, 91404 Orsay, France,
e-mail : <*Prénom*>.<*Nom*>@ thalesgroup.com

Résumé

Compte tenu de l'accroissement des besoins de calcul des applications temps réel et de l'évolution incessante de l'intégration des processeurs, le coût du placement de ces applications temps réel embarquées devient de plus en plus prohibitif. Cet article présente les progrès depuis [4] de la méthodologie de placement automatique d'applications de traitement du signal sur architectures multi-processeurs (PLC_2) étudiée par THALES[1]. Cette méthodologie de placement utilise la modélisation concurrente dans le cadre de la *programmation par contraintes*.

Mots-clés : Prototypage rapide, Parallélisation automatique, Traitement du signal systématique, Programmation par contraintes.

---

1. Introduction

Les puissances de calcul toujours croissantes donnent de nouvelles possibilités aux différents algorithmes de calculs scientifiques. Cependant le programmeur voulant profiter au maximum de ces avancées technologiques doit faire preuve d'agilité intellectuelle. La recherche d'une adéquation parfaite entre une application et une architecture cible est loin d'être simplifiée lorsqu'il s'agit de systèmes embarqués, exigeant généralement des temps de réponse limités, et contraints par les ressources machine. De ce fait s'accroît la nécessité de bénéficier d'un outil d'aide au développement dès la conception des algorithmes.

Dans cette optique THALES[2] a proposé dès 1995 une méthode de placement automatique reposant sur la modélisation concurrente et la programmation par contraintes (PPC) [4][2][5] : PLC_2. Le contexte applicatif et architectural était alors idéalisé puisqu'il s'agissait d'en étudier la faisabilité. Nous avons cherché à étendre ce contexte en prenant en compte des algorithmes et des architectures réels plus complexes.

Cet article présente nos avancées sur l'aspect applicatif. La section 2 fera une brêve analyse de l'état de l'art des différents outils de placement disponibles aujourd'hui. Elle présentera aussi rapidement la PPC. La section 3 décrira les applications de traitement de signal systématique ciblées. Avant de conclure, la section 4 fera état des extensions du contexte applicatif et de leurs conséquences sur nos modèles.

2. Cadre et contexte

2.1. Regard sur l'état de l'art des divers outils proposés

Certains outils permettent d'obtenir un code parallèle ou des directives de placement assimilables par un compilateur dédié (HPF par exemple).

---

[1] Anciennement Thomson-CSF.
[2] Avec le concours du Centre de Recherche en Informatique de l'École Nationale Supérieure des Mines de Paris.

Les compilateurs parallèles ont la difficile charge d'exhiber le parallélisme du code. Ils peuvent cependant être aidés par le programmeur au travers de directives explicitant ce parallélisme (qu'il sont libres de suivre ou non). Ces compilateurs, possèdant chacun leurs spécificités, apparaissent comme des boîtes noires sur lesquelles le programmeur n'a aucun contrôle. La famille de ces compilateurs est trop vaste pour être développée plus avant ici.

Parmi les autres environnements de développement ([8] [6] [10] [9]), il apparait que la plupart laissent au programmeur la tâche non négligeable et difficile d'extraire le parallélisme d'un algorithme. En revanche, ils permettent de simuler le code produit, donnant ainsi certaines garanties sur le résultat final.

PLC_2 a pour objectif de palier cette carence pour les applications de traitement du signal systématique (TSS) s'exécutant sur des machines parallèles homogènes. Elle génère des directives de placement garantissant les contraintes opérationnelles, à partir d'une description fonctionnelle de l'algorithme de TSS et de l'architecture cible. Notre méthode repose sur la PPC et la modélisation concurrente.

2.2. La modélisation concurrente et la programmation par contraintes

En modélisation concurrente, un modèle est vu sémantiquement comme l'ensemble des spécifications du comportement du constituant qu'il modélise. Ainsi il est caractérisé par un ensemble de contraintes qui lui sont propres, et mis en relation avec d'autres modèles par le biais d'autres contraintes. La figure 1 montre la représentation concurrente que nous avons du placement. Ces modèles sont issus de l'état de l'art de la parallélisation automatique, mais ont fait l'objet d'une adaptation au contexte applicatif choisi et au mode de résolution utilisé. Ces différents modèles (représentés par les bulles) sous-jacents à la

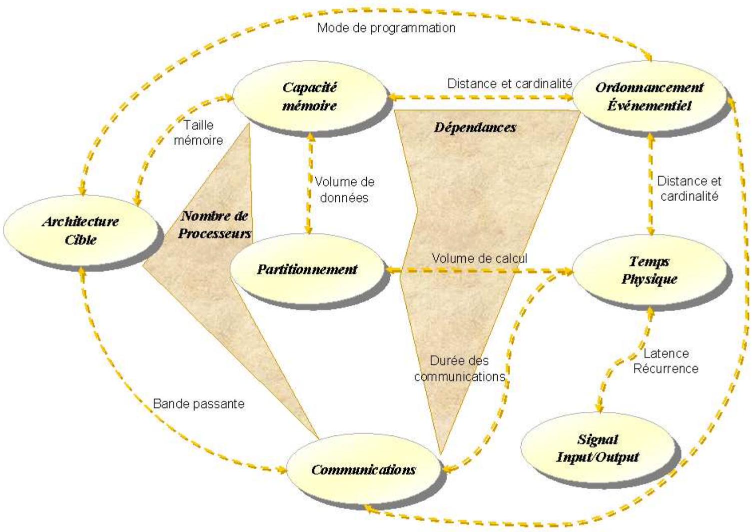

Figure 1: La problématique du placement vue par PLC_2

fonction placement, ainsi que les différentes relations (les flèches et les polygones) qui les relient, sont mathématiquement formalisés. Ainsi, de part l'aspect déclaratif de la PPC, une variable mathématique devient directement une variable « informatique » que l'on appelle *variable de domaine*. Elle est définie par son domaine de valeurs (un intervalle de $\mathbb{Z}$ dans notre cas) et les contraintes dans lesquelles elle est impliquée. Une solution est alors une instanciation cohérente de toutes ces variables.

Tout le raisonnement mis en œuvre par les contraintes est basé sur le paradigme de *manipulation d'informations partielles* [5]. L'avantage est que le système peut prendre des décisions (opérateur *ask*) sans attendre que l'ensemble soit déterminé. À chaque prise de décision, l'ensemble {variables + contraintes} cherche à rester cohérent (opérateur *tell*). Ce mécanisme de maintien de la cohérence, qui représente l'échange d'informations partielles, est appelé *propagation*.

Le mécanisme de résolution est générique. De ce fait la modélisation et la résolution sont indépendantes alors que bien souvent ces deux aspects sont « mélangés » dans des approches plus algorithmiques, telles que celles de la recherche opérationnelle. De plus la compositionalité est maximale puisque cela permet la coexistence et la coordination de modèles ayant différents niveaux de granularité.

L'intégration d'un nouveau modèle, ou la modification d'un modèle existant, n'affecte pas l'algorithme de résolution utilisé. D'où le choix de cette technologie pour la résolution globale du placement d'applications de traitement du signal systématique.

3. Le traitement de signal systématique

Notre étude se limite aux applications de traitement du signal dit systématique. Il s'agit principalement d'algorithmes de filtrage dont les données d'entrées proviennent de radar ou de sonar. L'objectif de tels opérateurs est d'éliminer le bruit d'un signal reçu pour ne garder que les informations pertinentes (détection d'une cible, caractérisation de sa position et/ou sa vitesse). Ces applications[3] sont du type traitement de données intensif. C'est-à-dire que leur complexité réside dans le nombre de données à traiter et du temps imparti pour le faire, plutôt que dans les calculs eux-mêmes. Notre préoccupation est donc de paralléliser les données plutôt que les calculs.

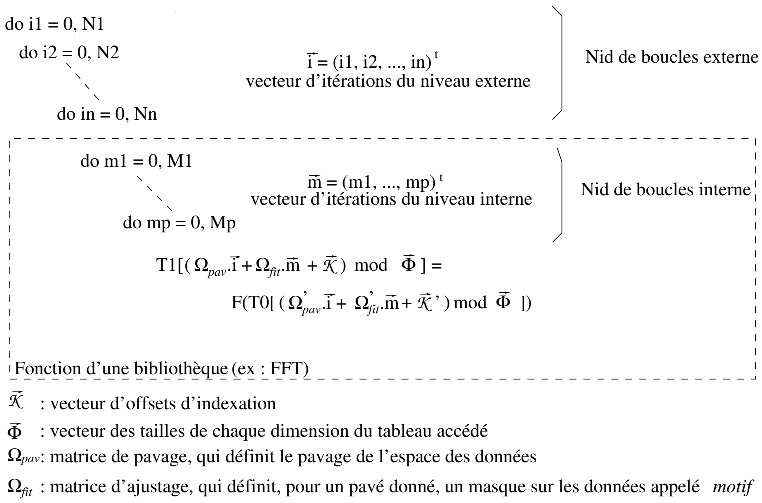

Figure 2: Structure d'un nid de boucles

Les algorithmes TSS sont définis par un ensemble de nids de boucles séquentielles, dont la structure est montrée par la figure 2. Le nid de boucles *interne* constitue une fonction de la bibliothèque TSS utilisée (une transformation de Fourier, par exemple). C'est le plus petit calcul que l'on puisse faire. Il définit également le *motif* qui est le plus petit ensemble de données que l'on puisse traiter en une fois. Ce nid de boucles n'est donc pas parallélisable[4]. Par opposition au nid de boucles *externe* totalement parallèlisable et qui définit le nombre de fois que la fonction TSS est appelée. Chacun des itérateurs de boucles, externe (pour l'accès principal) ou interne (pour le motif), porte une sémantique physique (temps, fréquence, case distance...). Ce qui explique pourquoi la boucle la plus externe est souvent infinie puisqu'elle correspond au caractère temporel du signal.

Ces fonctions TSS agissent sur des tableaux multidimensionnels (les traitements étant réguliers et s'effectuant sur des signaux multidimensionnels). Nous avons ainsi un quasi-isomorphisme entre les formules mathématiques du traitement du signal et leur représentation informatique. L'application est en assignation unique.

De ce fait la profondeur d'un nid de boucles externe est directement liée au nombre de dimensions du tableau produit par ce nid. Cette profondeur peut aller jusqu'à 7 dans certaines applications de traitement radar ([7]).

Les fonctions d'accès aux données d'un tableau **T** sont des fonctions affines des itérateurs de boucles définissant $\vec{\varphi}_{(\vec{\imath},\vec{m})}$ le vecteur d'indice, telles que $\vec{\varphi}_{(\vec{\imath},\vec{m})} = \left(\Omega_{pav}\vec{\imath} + \Omega_{fit}\vec{m} + \vec{\mathcal{K}}\right) \mod \vec{\Phi}$ (fig. 2).

Ω_{pav} et Ω_{fit} sont issues du formalisme Array-OL [1] utilisé initialement (fig. 2). Ces matrices sont diagonales à une permutation près. C'est-à-dire que les boucles d'un même niveau (externe/interne) sont

[3]Nous utiliserons indifféremment les termes *application* et *algorithme* pour désigner cet ensemble de calcul.
[4]Non pas par nature, mais parce qu'il n'est pas accessible.

indépendantes les unes des autres. Cela signifie également que l'ordre des boucles n'est pas nécessairement l'ordre des dimensions du tableau accédé.

Il n'existe pas de dépendance autre que lecture/lecture au sein d'un nid de boucles. Les seules dépendances écriture/lecture existantes définissent le flot de données (dont la représentation en graphe est quelconque mais sans circuit).

4. Extensions du contexte applicatif

La description donnée dans la section 3 donne le cadre des hypothèses applicatives considérées actuellement. La plupart des modèles utilisés ont été exposés dans [2] ou [4]. Nous ne donnerons ici que les contraintes correspondant aux changements d'hypothèses.

4.1. D'une chaîne à un graphe

L'ensemble des graphes de flots de données des applications considérées dans [4] était réduit aux chaînes. Nous avons traité le cas où le graphe est quelconque (mais sans circuit).

Cette prise en compte influence directement notre modèle de latence[5] car il repose sur la structure même du graphe. En effet, son calcul est fonction de la longueur du plus long chemin de dépendance allant d'un nid de boucles de l'ensemble d'entrée[6] (IN) jusqu'à un nid de boucles de l'ensemble de sortie[7] (OUT).

Dans un graphe valué orienté sans circuit, ce calcul correspond à la recherche d'un chemin de coût maximum. Il s'agit là d'un algorithme classique de parcours de graphe. Nous associons à chaque nœud nb du graphe flot de données un poids n_{nb} correspondant au coût maximum du sous-graphe issu de ce nœud. Chaque arc a un coût v défini comme étant la plus grande distance événementielle de dépendance séparant les nœuds liés par cet arc. Mais ces poids et ces coûts ne sont pas connues puisqu'ils dépendent du partitionnement et de l'ordonnancement (principe de modélisation concurrente). L'algorithme devient alors un ensemble récursif de contraintes qui permet d'obtenir le coût Δ du chemin critique du graphe complet : $\Delta = \max_{nb \in IN}(\beta_{nb} + n_{nb}) + 1$ avec

$$\begin{array}{rcl} \forall nb \in OUT, \quad n_{nb} & = & 0 \\ \forall nb \notin OUT, \quad n_{nb} & = & \max_{\{nb_j \mid \mathcal{D}(nb,nb_j)\}} \left(v_{nb,nb_j} + n_{nb_j}\right) \\ v_{nb,nb_j} & = & \max_{\{\text{blocs de calculs dependants}\}} \left(d^{nb_j}(\vec{c}_k^{\,nb_j}) - d^{nb}(\vec{c}_k^{\,nb})\right) \end{array}$$

$d^{nb}(\vec{c}^{\,nb})$ désigne la fonction d'ordonnancement du nid de boucles *nb*. Elle est de la forme $\vec{\alpha}^{nb}\vec{c}^{\,nb} + \beta^{nb}$, où β^{nb} est le temps de start-up, et $\vec{\alpha}^{nb}$ définit la période de calcul. $\vec{c}^{\,nb}$ est la composante temporelle du partitionnement 3D du nid de boucles externe ([4, Chap. 4]). $\mathcal{D}(nb, nb_j)$ est le prédicat prenant la valeur vrai s'il existe une dépendance écriture/lecture entre les nids de boucles nb et nb_j. Bien sûr, lorsque le graphe est une chaîne, nous retrouvons la contrainte de [4].

Si $nbc^{nb}(0,\Delta)$ désigne le nombre de fois qu'un bloc de calcul du nid de boucles *nb* est exécuté sur la longueur du chemin critique à partir de l'instant 0, et $ct(nb)$ la durée physique du calcul intégrant sa communication, alors le temps de latence, en SIMD, de l'application placée est tel que :

$$latence \leq \sum_{nb=1}^{NBMax} \left(nbc^{nb}(0,\Delta).ct(nb)\right) \qquad NBMax = \text{Nombre de nids de boucles}$$

4.2. Fonctions d'accès affines

Les fonctions d'accès aux données sont à la base du modèle des dépendances. Soit **T** un tableau et nb^w et nb^r deux nids de boucles tels que nb^w écrit dans **T** et nb^r lit dans **T**. Alors nb^w et nb^r sont dépendants si $\exists(\vec{i}^w, \vec{m}^w), (\vec{i}^r, \vec{m}^r) \,\Big|\, \vec{\varphi}^w_{(\vec{i}^w,\vec{m}^w)} = \vec{\varphi}^r_{(\vec{i}^r,\vec{m}^r)}$ [3]. L'introduction du partitionnement 3D[8] dans cette équation

[5] Nous appelons latence la différence entre la date de sortie au plus tard d'un signal avec la date d'entrée au plus tôt des signaux ayant contribué à son élaboration.

[6] Ensemble des nid de boucles dont tous les tableaux lus ne sont produit par aucun des nids de boucles de l'application.

[7] Ensemble des nids de boucles dont le tableau produit n'est lu par aucun des nids de boucles de l'application.

[8] $\forall \vec{i} \quad \exists(\vec{c}, \vec{p}, \vec{l}) \quad \vec{i} = LP\vec{c} + L\vec{p} + \vec{l}$, détails dans [4]

donne la relation suivante : $\exists(\vec{c}^w, \vec{p}^w, \vec{l}^w, \vec{m}^w)$, $(\vec{c}^r, \vec{p}^r, \vec{l}^r, \vec{m}^r)$

$$\left[\Omega^w_{pav}\left(L^w P^w \vec{c}^w + L^w \vec{p}^w + \vec{l}^w\right) + \Omega^w_{fit}\vec{m}^w + \vec{\mathcal{K}}^w\right] \bmod \vec{\Phi} = \left[\Omega^r_{pav}\left(L^r P^r \vec{c}^r + L^r \vec{p}^r + \vec{l}^r\right) + \Omega^r_{fit}\vec{m}^r + \vec{\mathcal{K}}^r\right] \bmod \vec{\Phi}$$

Les matrices étant diagonales, nous pouvons projeter l'équation dimension de tableau par dimension de tableau. De plus les fonctions d'accès croissantes nous permettent de définir l'espace des dépendances sur l'axe temporel c du partitionnement. Ainsi sont déduites les équations des droites D_{inf} et D_{sup} délimitant, dans l'espace d'itérations, l'espace des dépendances (fig. 3).

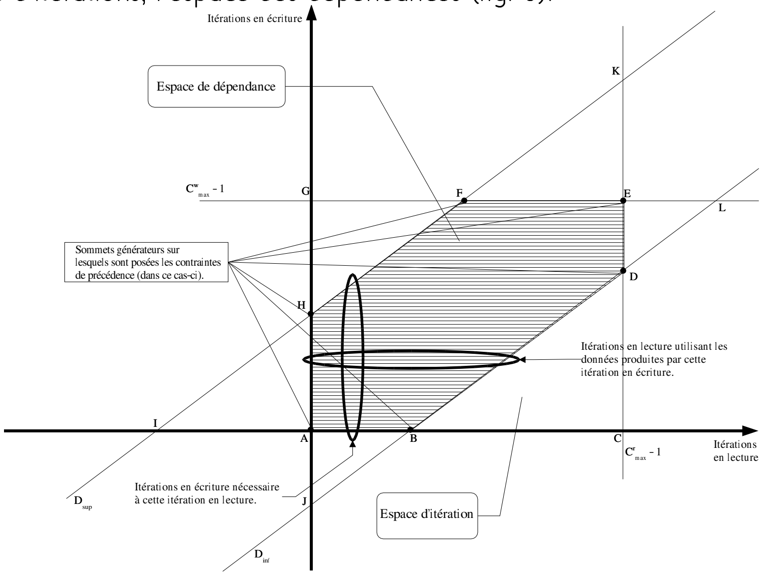

Figure 3: Le polytope de dépendances sur un exemple monodimensionnel

Une représentation polyédrique des dépendances a été choisie : les dépendances inter nids de boucles écriture-lecture sont approximées par l'enveloppe convexe de leur représentation. Les fonctions d'ordonnancement étant des fonctions affines et le polygone étant convexe, on exprime les contraintes de dépendances sur tous ses sommets générateurs (fig. 3). Ainsi ces contraintes sont respectées sur l'ensemble des points intérieurs à l'enveloppe convexe. Par exemple, la contrainte de précédence sur le point F de la figure 3, assurant que l'écriture a lieu avant la lecture, s'écrit ($d^{\{w,r\}}$ sont les fonctions d'ordonnancement des nids de boucles dépendants, l'un écrit, l'autre lit) : $\mathcal{D}\left(y_F, x_F\right) \Rightarrow \left(d^w(y_F) < d^r(x_F)\right)$

Ces sommets sont les points d'intersection des différentes droites délimitant l'espace de dépendances (par définition). La prise en compte d'un paramètre constant dans les fonctions d'accès a pour effet de donner plus de liberté aux deux droites D_{inf} et D_{sup}. Alors que dans un cadre linéaire, le point A est toujours dans l'espace de dépendance, dans un cadre affine ce n'est plus le cas. C'est-à-dire que selon le partitionnement (ou l'ordonnancement) choisi, le point I ou J peut appartenir à l'espace de dépendance ou non.

4.3. Amélioration du modèle de dépendances

Notre modélisation des dépendances conserve le modèle polyédrique décrit ci-dessus. Nous approximons l'espace de dépendances par la plus petite enveloppe convexe le recouvrant dont les sommets générateurs sont à coordonnées entières. Ce sont sur ces sommets que nous posons nos contraintes de précédence.

Les sommets générateurs de l'enveloppe convexe sont obtenus par arrondi des coordonnées des sommets du polygone exact. De ce fait, l'espace de dépendances sur-approximé contient plus de points, et par conséquent définit plus de dépendances qu'il n'y en a réellement. L'inconvénient est que cela élimine des solutions d'ordonnancement. Nous avons donc cherché à éliminer cette sur-approximation.

La description exacte du polygone n'est pas possible pour la principale raison que tous les paramètres intervenant dans sa définition sont des variables de domaine (i.e., sont inconnus initialement). Le nombre de points nécessaire à une description exacte est alors un paramètre variable. Il n'est pas possible de poser les contraintes de précédence sur les sommets générateurs.

La simplification des équations des droites D_{inf} et D_{sup} supprime dans certains cas la sur-approximation. Cette simplification consiste à obliger ces droites à passer par des points entiers. Ainsi, si D_{sup} est de la forme $ax + by = c$ alors sa forme simplifiée est

$$a'x + b'y = c' \quad \text{avec} \quad \left\{ \; a' = \frac{a}{pgcd(a,b)}; \quad b' = \frac{b}{pgcd(a,b)}; \quad c' = \left\lfloor \frac{c}{pgcd(a,b)} \right\rfloor \right.$$

Cette amélioration du modèle se traduit par la pose de quelques contraintes supplémentaires relatives à ces simplification d'équations dans les définitions des sommets générateurs. La difficulté ici réside dans la définition d'une contrainte pgcd qui n'a pas un comportement monotone. Néanmoins, grâce au caractère déclaratif de la PPC, nous avons pu l'introduire en exprimant l'ensemble de ses propriétés en contraintes.

5. Conclusion

Nous avons présenté dans cet article quelques évolutions du contexte applicatif traité par PLC_2. Cependant ce ne sont pas les seules puisque nous avons également repoussé les limites architecturales (modélisations fines des entrées/sorties, instauration d'une hiérarchie mémoire). Les machines cibles sont de type SIMD/SPMD avec un réseau de communication restant idéal. Une hiérarchie mémoire à deux niveaux est implémentée, mais notre modèle prend en compte un nombre quelconque de niveaux hiérarchiques.

À l'heure actuelle, avec les modèles présentés ici, nous traitons des applications comprenant une trentaine de tâches dont la profondeur des nids de boucles externes va de trois à sept [7]. Un problème de ce type représente une complexité de résolution de quelques 30 000 variables et un nombre de contraintes du même ordre de grandeur. Sachant que la taille des domaines de la plupart de ces variables est de l'ordre de 10^{15}, il est aisé de comprendre que le parcours de l'espace de solution de façon naïve et exhaustive est loin d'être efficace.

L'algorithme d'optimisation utilisé est standard et sans stratégie particulière. Ses limites commencent à être atteintes du fait de la complexité évoquée ci-dessus, et l'élaboration d'algorithmes plus efficaces (prenant en compte des heuristiques du domaine du placement) devient une forte préoccupation.

Nous nous intéressons aujourd'hui à la modélisation d'architecture de type multi-SPMD, redéfinissant ainsi un nouveau contexte pour les différentes notions présentées ici, et permettant d'en introduire de nouvelles.

Grâce au point fort de la PPC et de la modélisation concurrente, l'intégration de nouveaux modèles ou le changement d'algorithme de recherche de solutions est simplifié. Ce qui permet d'élargir le domaine couvert à des cas réels, aussi bien du point de vue des applications traitées que des architectures ciblées, ou encore des fonctionnalités de l'outil (comme la génération d'un code efficace).

Bibliographie

1. A.Demeure and Y. Del Gallo. An approach for signal processing design. In *SAME*, System on Chip Session, 1998.
2. C. Ancourt and al. Automatic data mapping of signal processing applicationsá. *IEEE International Conference on Application Specific Systems, Architectures, and Processors*, pages 350–362, 1997.
3. A. J. Bernstein. Analysis of programs for parallel processing. *IEEE Transactions on El. Computers, EC-15*, 1966.
4. C. Guettier. *Optimisation globale et placement d'applications de traitement du signal sur architectures parallèles utilisant la programmation par contraintes.* PhD thesis, École Nationale Supérieure des Mines de Paris, November 1997.
5. 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 D. Diderot - Paris VII, February 1995.
6. Lockheed Martin. *GEDAE Users' Manual / GEDAE Training Course Lectures.*
7. N. Museux. Radar aéroporté à forme d'onde mfr - transcription. Technical Report E/226/CRI - TSI/99/906, Thomson-CSF / LCR - ENSMP/CRI, 2000.
8. 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.
9. Y. Sorel and C. Lavarenne. http://www-rocq.inria.fr/Syndex/pub.htm.
10. E. A. Lee team. http://ptolemy.eecs.berkeley.edu/papers.