SEGMENTATION SPATIO-TEMPORELLE STATISTIQUE DES FOUILLIS RADAR
IMPLANTATION TEMPS REEL SUR MACHINE PARALLELE : CAMARO

Frédéric BARBARESCO, François GIERSH

THOMSON-CSF , Division Systèmes Défense et Contrôle
7, rue des Mathurins BP 10
92223 BAGNEUX

RESUME

Pour permettre la classification des fouillis radar, il serait souhaitable de pouvoir différencier préalablement les fouillis dynamiques des fouillis statiques, à partir de la composante homogène de leur mouvement. Pour ce faire, nous avons adapté un algorithme de l'INRIA [1][2] de segmentation spatio-temporelle statistique multiéchelle sur des séquences d'images de données de l'environnement fouillis provenant d'un radar 3D de veille longue portée. Cet algorithme a été parallèlisé et implanté sur une machine parallèle MIMD développée en interne par THOMSON-CSF SDC : CAMARO. L'algorithme, la machine CAMARO et son environnement de développement sont décrits dans cet article. Les résultats sur des données tests sont fournies, ainsi que ceux obtenus sur des données réelles de l'environnement fouillis radar.

ABSTRACT

In order to classify radar clutters, we differenciate dynamic from static according to their homogenous spatial motion. We have adapted a multigrid spatio-temporal segmentation algorithm, developed by INRIA, to images sequences of radar clutters environment from a 3D long range air surveillance radar. This algorithm has been parallelized and implemented in an MIMD (multiple instruction multiple data) architecture of a THOMSON-CSF SDC parallel machine : CAMARO. The multigrid spatio-temporal segmentation algorithm, parallel machine CAMARO and its software development environment are explained in detail in this following paper. We provide results from synthetic and real radar clutters data.

1. PREAMBULE

Les radar 3D de veille longue portée ont à leur disposition une bibliothèque de modes d'émission (forme d'onde, séquencement des pinceaux électromagnétiques en site, traitement du signal associé, paramètres de réglages, ...) adaptés à des situations synthétiques prédéfinies en laboratoires. L'objectif de la veille consiste à sélectionner le mode le plus adapté à la situation réelle de l'environnement radar, pour réduire les pertes en détection. Il est donc essentiel que le système puisse disposer d'une synthèse automatique de son environnement,ce qui passe par une analyse préalable des perturbations (fouillis, contre mesures électroniques, etc ...). Pour analyser cet environnement, une première étape consiste à utiliser l'information de mouvement des fouillis pour différencier les fouillis dynamiques (fouillis atmosphériques, chaff, ...) des fouillis statiques (sol, mer, ...).

Pour extraire cette information de mouvement des fouillis, nous avons appliqué un algorithme de relaxation multiéchelle utilisant des modèles énergétiques décrits dans [1]. Cet algorithme utilise un modèle énergétique markovien et permet une segmentation des fouillis radar suivant leur composante homogène de mouvement. Pour l'application radar, nous avons implanté sur machine parallèle (machine parallèle THOMSON-CSF SDC : CAMARO) la version multiéchelle par champs markovien, la version multirésolution, ainsi qu'une variante rapide HCF (High Confidence First) de la version multiéchelle.

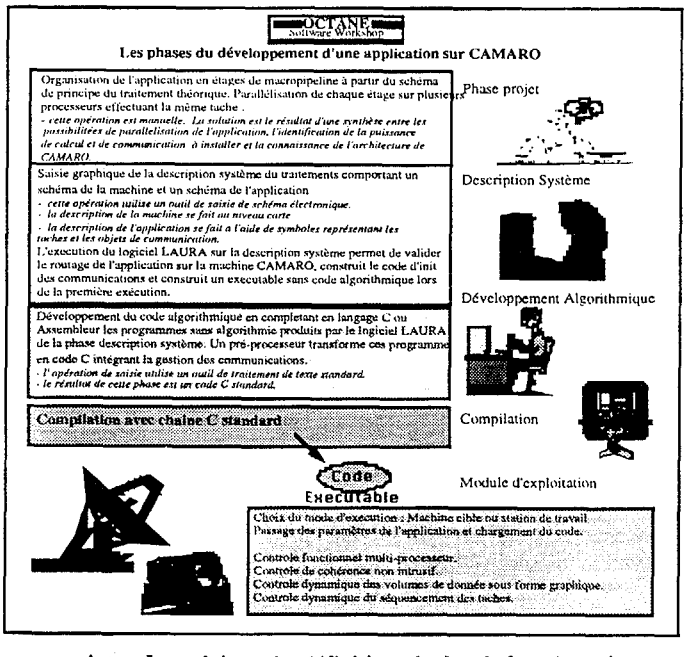
Dans une première partie, nous allons décrire la machine CAMARO, ainsi que son environnement logiciel. Dans une seconde, seront décrits les différentes versions de l'algorithme de segmentation spatio-temporelle des fouillis radar. Nous donnerons, en dernier lieu des exemples obtenus.

2. LA MACHINE PARALLELE CAMARO

C'est pour répondre aux besoins croissants de puissance de calcul des adars que la division Système Défense et Contrôle de Thomson-CSF a té amenée à développer une architecture parallèle suffisamment nodulaire pour s'adapter à l'ensemble des besoins des radars existants u en développement (du radar de poursuite au radar FFC en passant ar le radar météo ou de contrôle de trafic aérien). Basée sur une carte ère qui associe deux cartes mezzanine de 7 DSP (40Mips) du ommerce (format double Europe), cette architecture permet de faire availler ensemble jusqu'à 2000 DSP (80 Gflops). Les liens de ommunication (6 liaisons à 40Mbits/s par carte) offrent aujourd'hui es débits moyens garantis proches des valeurs crêtes : les flots sont ontinus (pilotage sans interruptions) avec des possibilités, le cas chéant, de routage dynamique. Ils pourraient supporter à l'avenir la norme HIC (Hétérogeneous InterConnect) à des débits plus élévés (jusqu'à 3 Gbits/s). L'architecture de communication est en outre indépendante des algorithmes de traitement implémentés dans les DSP. Cet avantage, associé à l'atelier logiciel OCTANE utilisant un langage de haut niveau (C et non assembleur), permet déjà à cette architecture de viser, avec des coûts de moyenne série, d'autres applications que celles du radar.

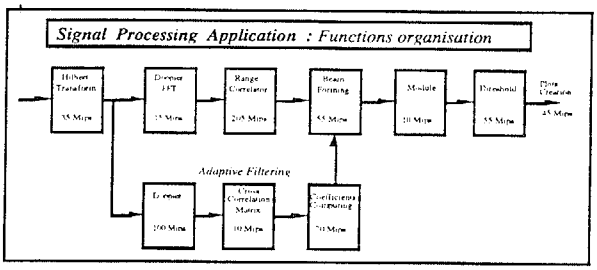
L'atelier logiciel OCTANE est destiné au développement d'applicatifs multi-processeurs. La productivité et la fiabilité du développement des codes applicatifs est obtenue en réduisant le problème multi-processeurs en n problèmes mono-processeur et en générant automatiquement le code des communications inter-processeur. Cette opération est réalisée à partir d'une représentation graphique définissant le système applicatif. La mise en oeuvre de cette technique permet un gain d'environ 85% en lignes de code d'une application radar.

Méthodologie de développement :



Phase projet : Le schéma de définition de la chaîne de traitement théorique de l'application à mettre en oeuvre est le point de départ de

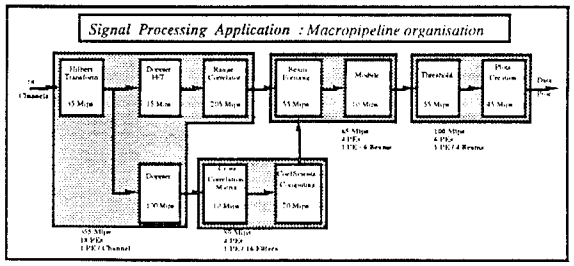
tout développement. Ce schéma définit les fonctions de transfert et leur ordre de séquencement suivant un modèle de type mono-processeur. Le schéma ci-dessous donne un exemple d'une telle description d'application.



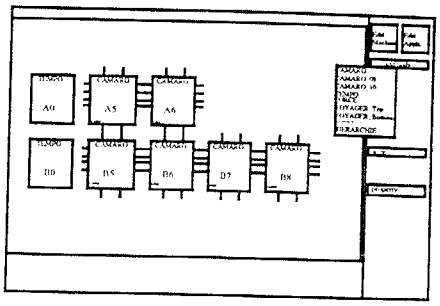
Le concepteur identifie à partir de ce schéma l'ensemble des informations nécessaires pour la définition de la configuration matérielle et logicielle de cette application développée sur CAMARO. Les points ci-dessous sont donnés à titre indicatif.

-la nature des traitements à effectuer.
-la puissance de calcul à mettre en oeuvre pour chaque fonction.
-la taille mémoire requise.
-les débits de données en entrée et en sortie de chaque fonction.
-les types de parallélisme potentiels de chaque fonction.
-les modes d'échanges globaux inter-fonctions.

Ces informations brutes de définition de l'application sont ensuite synthétisées par le concepteur pour aboutir à la reconstruction d'une nouvelle chaîne de traitement par regroupement et décomposition des différentes fonctions de traitement élémentaires en étages de macro-pipeline. Un étage de macro-pipeline correspondant à un seul type de logiciel exécuté en // sur plusieurs processeurs élémentaires, on cherchera à en diminuer le nombre pour limiter le nombre de logiciels à développer et diminuer le nombre et le volume d'échanges de données entre tâches de traitement. Dans le cas d'applications utilisant les ressource de CAMARO aux limites, cette opération demande une connaissance fine du modèle de programmation et des possibilités de configuration machine pour en assurer un bon usage. Le schéma ci-dessous représente une décomposition possible du traitement représenté ci-dessus.

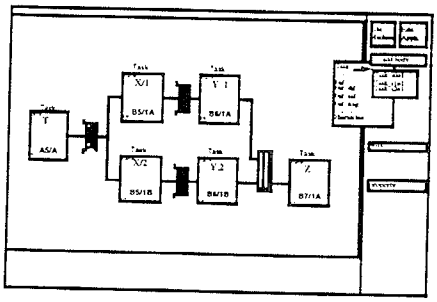


Définition système : L'atelier logiciel OCTANE intervient après cette première phase d'étude projet pour assurer la saisie graphique de la définition système de l'applicatif. Cette définition système comprend un schéma de la configuration matérielle et un schéma de l'organisation de l'application. Ces deux schémas définissent l'ensemble des informations sur l'organisation multiprocesseur de l'application. La liste des tâches à développer et la définition des flux de données entre ces tâches permettent de créer un code exécutable qui validera l'ensemble des chemins de données de l'application.
Saisie graphique de la machine : On représente la machine CAMARO à l'aide de symboles pré-définis représentant les cartes. On nomme chaque carte d'après la convention suivante:<ref-baie{W-X}><ref-panier{A-E}><ref-carte{0-20}>. Le schéma ci-dessous donne un exemple de définition de configuration machine.



Saisie graphique de l'application : On représente l'application à l'aide de symboles définissant les tâches algorithmiques et les objets de communication (simple buffer, diffusion, store & forward, anneau). On précise sur le schéma de l'application le nom des tâches et le nom du processeur hardware qui exécutera chacune des tâches. Le principe de désignation d'une tâche est le suivant: <nom de la tâche>[numéro];

Deux tâches de même nom ne doivent pas avoir même numéro de référence mais ces numéros peuvent être absolument quelconques. Deux tâches de même nom ont un même code algorithmique mais ces codes peuvent tester à l'exécution le numéro de référence et gérer des particularités ou effets de bord. Le schéma ci-dessous représente un exemple de saisie d'application.



Après avoir saisi ces deux schémas, on lance le logiciel LAURA/Routage qui vérifie que les schémas sont conformes aux règles de spécification, et qui valide la capacité des algorithmes de routage et circuits de routage à réaliser l'application projetée. Ce logiciel crée des fichiers de commande qui permettent de visualiser le routage. Dans l'hypothèse d'une impossibilité de routage, on peut faire un "état des lieux" et contraindre éventuellement manuellement le routage de certains chemins. Par ailleurs, le logiciel de routage construit un fichier prototype (si celui-ci n'existe pas déjà) pour chaque tâche programme définie dans le schéma de l'application. Ces fichiers existent pour chaque étage de macro-pipeline et héritent du nom racine de la tâche de macro-pipeline plus l'extension ".pgm" qui permet de les distinguer des fichiers ".c".
Après validation du routage et création automatique des nouveaux fichiers prototypes, deux actions complémentaires sont possibles sans analyse supplémentaire :
- Lancement de la chaîne de compilation sur station de travail pour création d'un exécutable permettant la validation des fonctions de communication et du séquencement.
- Si l'on dispose par ailleurs d'une machine cible, il sera possible, dès cette première phase, de générer un code exécutable conforme à la spécification et comprenant un logiciel de test des communications qui est systématiquement intégré à toute application.

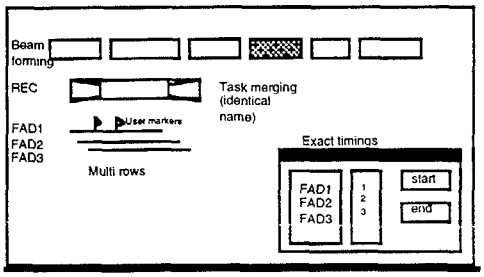
Développement Algorithmique: Après la définition système de l'application et la validation du routage, on pourra intégrer les codes applicatifs dans la spécification système. Le modèle de base du développement de l'algorithmique est basé sur un principe d'interface entre le code textuel écrit en C et le schéma graphique. Ce principe d'interface repose sur un passage à l'applicatif des adresses des buffers d'entrée et de sortie des tâches. Par ailleurs, un fichier optionnel *.prj

permet de préciser le regroupement de différents étages de macro-pipeline au sein d'un même exécutable. Le lancement du module développement passe par la saisie textuelle des fonctions algorithmes.

Un fichier optionnel *.grp permet de déterminer le regroupement de plusieurs applications au sein d'un même exécutable. Après toutes ces définitions, on peut lancer une compilation permettant de créer un exécutable de simulation sur station de travail ou l'exécutable de l'application sur la machine cible.

L'exécutable crée sera utilisé par le module d'exploitation. Ce module permettra de passer à l'exécutable en simulation sur station de travail ou sur machine cible l'ensemble des paramètres permettant de faire varier ses conditions d'exécution sans avoir à le recompiler.

Exploitation machine : Le module exploitation assure la gestion du téléchargement et de l'exécution du code applicatif sur station de travail ou machine cible avec contrôle possible par debugger. Après execution du code sur machine cible, un module d'exploitation machine permet la visualisation graphique de volumes de données, par interface avec le logiciel PV-WAVE, la visualisation de l'activité des différents processus de la machine, tâches de calcul, modules algorithmiques. Le schéma ci-dessous synthétise cette visualisation d'activité.



3. LA SEGMENTATION SPATIO-TEMPORELLE

Les observations (valeurs des pixels des images d'environnement) et les étiquettes (informations que l'on cherche à extraire des observations : 1 fouilllis en mouvement, 0 fouillis statique) sont supposées former des champs aléatoires définis sur de s grilles rectangulaires. Le critère d'optimisation est celui du *Maximum A Posteriori (MAP)*, c'est à dire, étant donné le champ d'observation O, on cherche le champ d'étiquettes ê le plus probable au sens de la distribution a posteriori, soit :

$$\hat{e} = \arg \max_{e} p(E=e/O=o)$$

La règle de Bayes et le théorème de Hammersley et Clifford (hypothèse que (E,O) forment un champ de Markov) nous permettent d'affirmer que cela revient à chercher le minimum global, soit :

$$\hat{e} = \arg \min_{e} U(e,o) = \arg \min_{e} [U_1(e,o) + U_2(e)]$$

Pour notre application, le détection de mouvement repose sur l'analyse des variations de la fonction de luminance entre les instants t-dt et t. Les observations seront donc les différences inter-images :

$$o_t(s) = |I_t(s) - I_{t-dt}(s)|$$

A ces observations quantitatives sont ajoutées des observations symboliques binaire :

$$\bar{o}_t(s) \in \{0,1\}$$

Celles-ci indiquent la présence ou l'absence de variations significatives de luminance au site s entre les instant t-dt et t. Elles sont obtenues par un détecteur de changement, simple seuillage de la différence de luminance sur une fenêtre A de taille n donnée. Dans [1], les auteurs décompose en trois termes l'énergie globale :

$$U(e_t, o_t, o_{t+dt}, \bar{o}_t, \bar{o}_{t+dt}) = U_{11}(e_t, o_t, o_{t+dt}) + U_{12}(e_t, \bar{o}_t, \bar{o}_{t+dt}) + U_2(e_t)$$

L'énergie U_2 ,correspondant à une énergie de voisinage, se décompose en somme de potentiels sur les cliques C engendrée par un système de 8-voisinages.

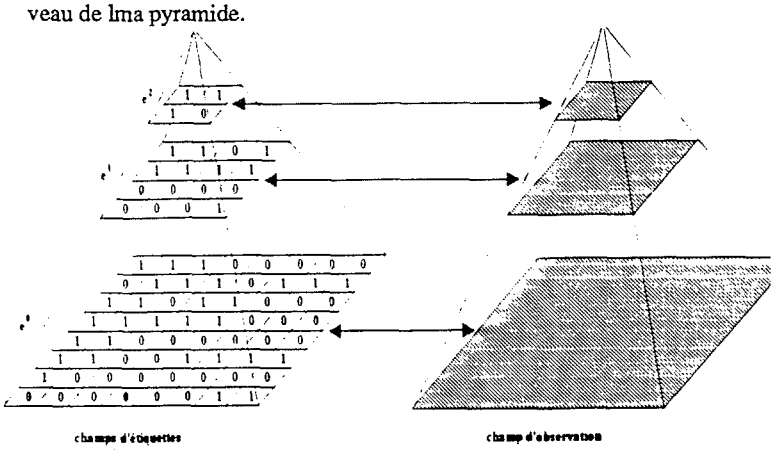
$$U_2(e_t) = \sum_{c \in C} V_c(e_t)$$

Le choix de potentiels permet de conférer des propriétés a priori sur le champ d'étiquettes. Dans cette application, nous recherchons des masques de mouvement compacts, à l'image des objets mobiles dont ils sont la projection dans le plan image. Nous attribuons pour cela un potentiel non nul aux seules cliques binaires. Il s'agit d'un potentiel à deux niveaux, indépendant de l'orientation des cliques :

$$U_2(e_t) = \sum_{c \in C} V_c(e_t)$$

U_{11} exprime le lien entre les variation,s temporelles de luminance et les étiquettes. U_{12} permet d'éviter la prise en compte des zones d'occlusion aux bords des masques.

Pour minimiser l'énergie décrite précédemment, on utilise l'algorithme de relaxation déterministe Métropolis, à différents niveaux de résolution. En effet, la stratégie multirésolution classique consite à décomposer l'image en une pyramide de sous-images de taille réduite et de travailler successivement sur chaque imagette. On minimise l'énergie à chaque niveau de lma pyramide.

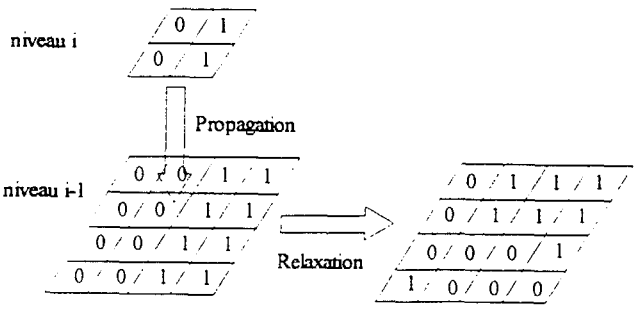


Structuration pyramidale multirésolution.

Une fois la convergence obtenue à un niveau, le niveau inférieur est initialisé par interpolation par répétition du résultat précédemment obtenu. C'est à dire qu'un pixel ou une étiquette à un niveau donné correspond à un bloc de quatre pixels ou étiquettes au niveau supérieur. On peut considérer qu'un niveau donné représente une bonne approximation du niveau juste inférieur, et donc que la convergence sera obtenue plus rapidement en initialisant l'algorithme avec cette approximation. Le niveau de résolution le plus grossier est initialisé, pour la première image, par :

$$\text{Max}(\bar{o}_t, \bar{o}_{t+dt})$$

Par la suite, l'initialisation se fait par le résultat final obtenu à l'image précédente.



Stratégie multiéchelle descendante.

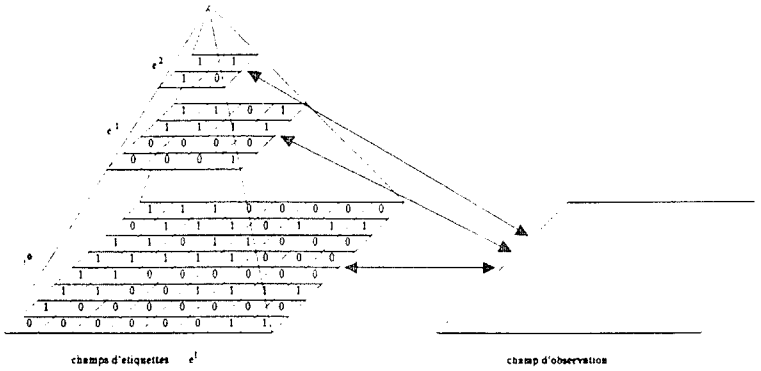
Cette stratégie permet de converger plus vite, mais aussi d'obtenir des champs d'étiquettes beaucoup plus homogènes qu'en ne travaillant à la résolution la plus fine. Cependant cette méthode reste lourde, car elle oblige à faire au préalable la décomposition multirésolution de l'image. Pour palier à cet inconvénient, on choisira une version multiéchelle, dans laquelle seule le champ d'étiquette est décomposé de façon pyramidale sur les observations au niveau le plus fin.. On ne parlera plus de pixels voisins mais plutôt de blocs de pixels voisins, car à une résolution donnée, un pixel représentera un bloc de quatre pixels au niveau inférieur. On introduit des coefficients de pondération de l'énergie de voisinage permettant de prendre en compte le niveau de résolution auquel on se trouve. L'énergie de voisinage à l'échelle i sera donnée par :

$$U_2^i = \sum V_c^i(e_t^i)$$

où un système de 8-voisinage est considéré. Pour les autres énergies :

$$U_{11}^i(e_t^i, o_t, o_{t+dt}) \quad \text{et} \quad U_{12}^i(e_t^i, o_t, o_{t+dt})$$

on profite de toute l'information disponible, c'est à dire au niveau de résolution le plus fin.

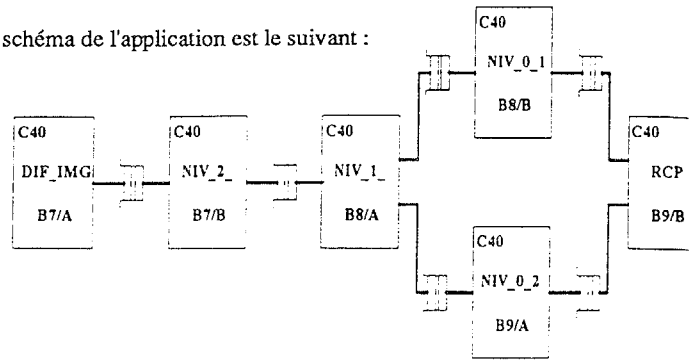


Structuration pyramidale du champ d'étiquettes.

Pour accélerer l'algorithme, on a considéré la version HCF (High Confidence First). Cette méthode se concentre directement sur les points instables, c'est à dire pour lesquels l'énergie n'est pas minimale, contrairement aux deux autres méthodes qui trainent systématiquement tous les points de l'image. Pour créer la pile d'instabilité, on calcule l'énergie locale pour tous les points et on vérifie si elle est minimale. Dans le cas contraire, on calcule un coefficient d'instabilité défini comme la différence entre l'énergie actuelle et l'énergie minimale. A chaque itération, on prend le point au sommet de la pile et on modifie son étiquetage de façon à minimiser l'énergie locale. Dans notre cadre markovien, cela n'entraine que la modification des énergies de ses huit voisins. Par différence, seul le terme U_2 correspondant à l'énergie de voisinage intervient dans le calcul de la nouvelle instabilité. Ce changement d'étiquette peut donc modifier l'instabilité des voisins, qui seront alors intégrés ou non à la pile. Ce procédé est répété jusqu'à ce que la pile soit vide.

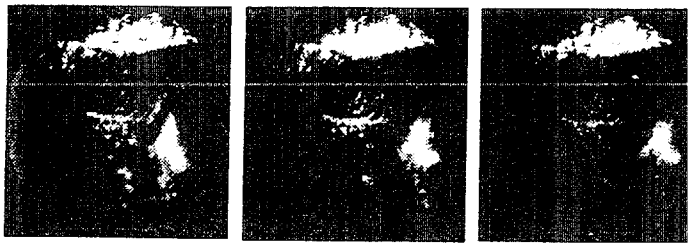
Pour la parallèlisation, deux méthodes s'offrent à nous : découpage de l'algorithme, ou découpage des données. Or le découpage des données est peu adapté au problème, pour lequel les objets en mouvement dans l'image ne sont pas uniformément répartis (mauvaise répartition des charges). Aussi, la seconde méthode de type pipe-line semble plus adapté. L'idée est de se fixer un nombre de niveaux indépendant de la taille de l'image. Nous avons choisi trois étapes (niveau de résolution 2, 1 et 0). Il faut ajouter à cela une première étape pour calculer les observations. Nous obtenons donc quatre étapes. Des séries de tests soulignant l'importance relative de chacune des étapes définies, ont fait apparaître l'utilité de la structure en pipe-line en divisant par deux le temps de traitement au niveau 0 (découpe du traitement sur deux processeurs travaillant en parallèle sur deux moitiés d'image).

Le schéma de l'application est le suivant :

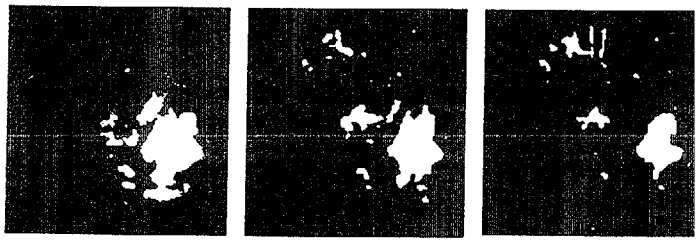


Le sixième processeur fait office de récepteur, qui doit envoyer les données vers l'extérieur (processuis temporaire en attendant une sortie par lien ETHERNET).

Les résultats sont les suivants :



Trois images extraites d'une sequence d'evolution du chaff. On voit que l'objet en mouvement et déformable. De plus les fouillis de proximite au centre de l'écran se modifie beaucoup.



Résultat de la segmentation temporelle. On obtient bien les masques des objets en mouvement. Quelques parasites subsistent dû au probleme de qualite des images radars. ou la mise a jour des différents secteurs peut poser quelques problemes.

5. CONCLUSION ET PERSPECTIVES

En régime continu, nous obtenons un temps de calcul de 42,2 secondes pour 10 images 128*128, et l'estimation donne un temps de calcul de 139 secondes en mono-processeur. L'accélération obtenue est donc de 3,3 pour cinq processeurs. Cependant, il apparaît que le processus chargé de calculer le niveau 1 s'avère être le processus limitatif (au vue du détail des temps d'execution par processeurs). Il serait ainsi envisageable de mettre un processeur pour le niveau 2, deux processeurs pour le niveau 1 et trois pour le niveau 0. L'étude des temps nous indique que cette répartition serait certainement très bonne, c'est à dire que les différents processeurs auraient une charge de travail équivalente, et que peu d'entre eux devraient attendre des données en entrée ou une allocation de buffer de sortie.

6. BIBLIOGRAPHIE

[1] Fabrice HEITZ, Patrice PEREZ et Patrick BOUTHEMY, "Constrained Multiscale Markov Random Fields and the Analysis of Visual Motion", INRIA Rapport de recherche n°1615, Février 1992

[2] Patrick PEREZ et Fabrice HEITZ, "Une approche multiéchelle à l'analyse d'images par champs markoviens", INRIA, rapport de recherche n°1477, Juillet 1991