Amélioration de la scalabilité de l'assimilation variationnelle dans les modèles de Météorologie par l'utilisation d'algorithmes mieux adaptés ou par d'autres approches plus innovantes.

Robin Degrave

Rapport de stage en entreprise

Maitre de stage : Ryad El Khatib
Tuteur universitaire : Natalie Bonneton

Toulouse, le 09 Septembre 2011

Table des matières

Chapitre 1

Introduction

Météo-France est l'organisme français de météorologie, un établissement public administratif, chargé de la prévision et de l'étude des phénomènes météorologiques. Depuis 2008, les prévisionnistes de Météo-France dispose du modèle numérique de prévision du temps à échelle fine *AROME*. C'est un modèle à aire limité qui permet des prévisions numériques avec une précision de 2.5km. Il peut être exploité de eux manières différentes : la prévision et l'assimilation.Le mode d'utilisation quotidienne de ce modèle est l'assimilation, c'est-à-dire que l'état initial est un mélange entre une prévision faite quelques heures avant (le *GUESS*) et des observations recueillies par différents instruments météoroligiques. Il existe l'assimilation pour l'altitude qui nécessite une analyse variationnelle et une analyse de surface qui demande une interpolation optimale connue sous le nom de *CANARI*. L'exécution d'*AROME* peut se diviser en plusieurs étapes dont notamment le *screening* ou encore la *minimisation*.
Chaque partie monopolise un certain nombre de processeurs. Il est primordial de tirer le meilleur des ressources de calcul tout en améliorant le temps d'exécution, autrement optimiser à la fois la scalabilité et la performance.
On s'intéressera notamment au logiciel *CANARI* qui nécessite un état des lieux. En effet, l'évolution des modèles et des technologies modifient l'utilisation faite du logiciel. Par exemple, les types d'instrument météorologiques se sont perfectionnés tout comme les ordinateurs un nombre de processeurs considérables. La contrainte pour l'opérationnel est d'avoir un code capable de resituer l'application choisie en un temps limité.
On fera alors une analyse et une révision des algorithmes codés en *FORTRAN* sur 2 types de calculateurs : scalaire et vectoriel. Grâce à l'aide de plusieurs spécialistes, l'objectif sera de trouver les solutions les plus judicieuses.
Le but défini de ce stage est d'obtenir une amélioration algorithmique sur différents types de calculateurs par des approches originales, à travers une collaboration avec des chercheurs expérimentés au sein d'une entreprise de renommée internationale.

Chapitre 2

Optimisation du code Canari sur machine scalaire

2.1 Etat initial du code

Lors d'une exécution sur 1 processeur du code de référence Canari, on dresse un profil le profil suivant :

```
Name of the executable : ./MASTERODB
Number of MPI-tasks : 1
Number of OpenMP-threads : 1
Wall-times over all MPI-tasks (secs) : Min=426.010, Max=426.010, Avg=426.010, StDev=0.000
Routines whose total time (i.e. sum) > 1.000 secs will be included in the listing
  Avg-%   Avg.time   Min.time   Max.time   St.dev  Imbal-%   # of calls : Name of the routine
 40.92%    174.285    174.285    174.285    0.000    0.00%      1051806 : CASGQA
 13.20%     56.216     56.216     56.216    0.000    0.00%      1051806 : CASGRA
  8.21%     34.986     34.986     34.986    0.000    0.00%      1577772 : CASPIA
  3.86%     16.455     16.455     16.455    0.000    0.00%      1512170 : MINV:GECO
  3.55%     15.117     15.117     15.117    0.000    0.00%      1512165 : CATRMA
  2.98%     12.696     12.696     12.696    0.000    0.00%      1563395 : CACOVA
  1.80%      7.647      7.647      7.647    0.000    0.00%      1563395 : CAMERA
  1.78%      7.601      7.601      7.601    0.000    0.00%      1512169 : MINV:GEDI
  1.57%      6.688      6.688      6.688    0.000    0.00%        50411 : CANADA
  1.34%      5.725      5.725      5.725    0.000    0.00%           11 : LFIFER_MT
```

On constate alors que les routines CASGQA, CASGRA et CASPIA sont, très largement, les plus coûteuses. On s'attachera alors à décrypter leur fonctionnement et ainsi diminuer le temps passé dans ces routines afin d'optimiser le code tout en gardant les mêmes résultats.

Pour se donner une idée de la structure du code, on peut se référer à un extrait de l'arbre d'appel des routines disponible en annexe (voir Fig(7.2)).

2.2 Abaissement de la boucle sur les points horizontaux pour CASGQA et CASGRA

Quand on observe en détails le profil de référence, on remarque que les routines CASGQA et CASGRA sont appelés un très grand nombre de fois (1051806 fois chacune). Cela vient du fait qu'elles sont appelés au coeur d'une boucle ayant un très grand nombre d'itérations. En remontant dans l'arbre d'appel et en examinant les routines appelantes, on observe que CASGQA et CASGRA sont appelées par CASGVA dans une boucle de 7 itérations mais CASGVA étant appelée, elle-même, par CANADA dans une boucle sur les points de grille. Un premier objectif va donc être faire rentrer la boucle sur les points de grille dans CASGVA puis dans CASGRA et CASGQA, afin de limiter le nombre d'appel à ces routines. Le but étant de restreindre l' accumulation du coût temporel d'appel à ces sous-programmes.

Schéma caricatural de la structure d'appel :
Dans **CANADA**
Boucle sur les points de grille
CALL CASGVA
Boucle de 7 itérations sur les types d'observations
CALL CASGRA
CALL CASGQA

On veut alors transformer ce schéma en :
Dans **CANADA**
CALL CASGVA
Boucle de 7 itérations sur les types
CALL CASGRA
Boucle sur les points de grille
CALL CASGQA
Boucle sur les points de grille

2.2.1 Premier degré d'abaissement de la boucle sur les points de grille

Pour abaisser la boucle sur les points de grille dans la routine CASGVA, il suffit de modifier les arguments lors de l'appel à la routine. En effet, au lieu de passer les variables dépendantes du point de grille en argument, on passe maintenant des tableaux en entier. Désormais, lors de l'appel à CASGVA :

- On prend le tableau de latitude de tous les points au lieu de la latitude d'un seul point
- On prend le tableau de longitude de tous les points au lieu de la longitude d'un seul point
- On prend le nombre de points de grille au lieu de prendre le numéro du point de grille concerné

2.2.2 Deuxième degré d'abaissement de la boucle sur les points de grille

Dans CASGVA, on veut donc maintenant abaisser la boucle sur les points au sein des 2 routines les plus coûteuses, c'est-à-dire CASGRA et CASGQA. Il faut faire tomber la dépendance des variables de CASGVA au point de grille. Hormis, les variables entrant en argument des routines appelés, il existe une seule variable NINSELG dépendant du point. Cette variable intervient au coeur d'un test .OR. mettant en jeu 3 conditions (IF (condition1 .OR. condition2 .OR. condition3)). On a, tout simplement descendu, ces tests dans CASGRA qui est la première routine appelé. On peut noter qu'on contrôle notamment les types d'observations et qu'avec la progression technologique, on utilise plus du tout les mêmes types d'observations qu'autrefois. Ce test perd donc un peu de son utilité. Néanmoins, il a été décidé de le garder mais il sera facile à supprimer dans l'avenir.
La descente de la boucle a donc engendré les mêmes modifications que précédemment pour l'appel a CASGRA et CASGQA. On passe en argument maintenant des tableaux pour les latitudes et pour les longitudes, ainsi que le nombre total de point.

La descente des boucles sur les points génère un gain en temps de 2.5%, ce qui est négligeable. On va donc maintenant analyser le fonctionnement de CASGRA et CASGQA afin d'améliorer l'optimiser du logiciel.

2.3 Travail dans CASGRA et CASGQA

Les routines CASGRA et CASGQA sont les deux routines les plus coûteuses lors de l'exécution du logiciel CANARI. Elles s'occupent toutes les deux de la sélection géographique des observations. En effet, pour un point de grille, c'est-à-dire un endroit de la terre, CASGRA sélectionne toutes les observations de données météorologiques dans un rayon donné. La routine CASGQA s'occupe de conserver une isotropie dans cette sélection.

Le schéma suivant montre la manière dont est fait la sélection. On prend les observations les plus proches (dans CASGRA). On veut qu'elles soient réparties tout autour du point de grille. Pour cela, on définit 4 quadrants, où on limitera le nombre d'observations dans chacun d'entre eux (dans CASGQA).

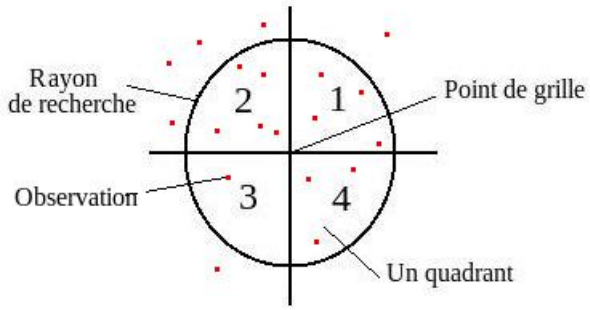

FIG. 2.1 – Schéma de la sélection géographique faite par CASGRA et CASGQA

2.3.1 Modification de CASGRA

Tout d'abord la première modification émane des manoeuvres précédentes. La boucle sur les points a, en effet, été poussée. Donc, on se retrouve avec cette boucle qui englobe tout le coeur de la routine.
En analysant cette routine, il apparaît que les modifications à effectuer sur cette routine, pour qu'elle soit plus efficace, concernent des tests. En particulier des tests qui force une sortie de la routine ce qui casse la performance.
Le premier test à analyser concernent un dépassement de dimension dans les tableaux d'adresses et de distances des observations. L'alternative pour contourner ce test a été d'allouer ces tableaux a des dimensions plus grande afin que la condition soit toujours vraie. On alloue maintenant des tableaux de données sur les observations avec le nombre total d'observations. Le code avait été écrit à la base pour faire une analyse en 3 dimensions mais maintenant le logiciel CANARI ne fait que de l'analyse de surface. Ce nombre total d'observation est considérable quand on travaille dans l'espace alors qu'en 2 dimensions, il est plus raisonnable. On a donc pu se permettre de dimensionner ces tableaux de données avec ce paramètre, sans utiliser trop de mémoire, grâce au changement de configuration du logiciel.

2.3.2 Modification de CASGQA

Comme pour CASGRA, la première modification est la présence de la boucle sur les points dans cette routine.
Le rôle du sous-programme CASGQA est de répertorier, pour chaque observation le quadrant auquel elle appartient. Ensuite, on compte les observations par quadrants et on a un nombre limite fixé (qui s'appelle KMXGQ) d'observations à garder dans chacun d'eux. Ainsi, si le nombre d'observations par quadrant n'atteint jamais le nombre limite, on garde toutes les observations. Sinon, on conserve les KMXGQ plus proches.
On mémorise alors l'adresse et la distance des observations sélectionnés.
L'analyse de cette routine a montré que les points à améliorer étaient principalement un algorithme de tri ainsi que la répartition par quadrant.

Prinicipe du tri des observations dans le code de référence Dans le code de référence, le principe est de compter les observations dans chaque quadrant. On mémorise alors dans des tableaux l'adresse et la distance des observations.
Si, dans un quadrant, le nombre d'observation limite est atteint, on calcule la distance maximale dans le tableau qu'on a mémorisé. On connait donc l'observation la plus éloignée du point de grille. L'observation suivante sera donc sélectionnée si elle est plus proche du point de grille que l'observation limite. Dans ce cas, les valeurs d'adresse et de distance correspondant à cette observation limite sont écrasée par celle de la nouvelle observation.
On réitère ce procédé pour toutes les observations n'ayant pas été traitée dans ce quadrant.

Modification de l'algorithme de tri La première modification de l'algorithme était de faire un tri des observations dans les quadrants plutôt que de le faire consécutivement dans chaque quadrant. Lors de l'application de cette méthode, on a été confronté à des problèmes de scores et on a vite réalisé que le gain ne serait pas important.
L'algorithme de référence a clairement une complexité en $O(n^2)$, il est connu que les algorithmes de tri les plus efficaces possèdent une complexité en $O(nlog(n))$.
Le deuxième essai a été de reprendre un algorithme connu en $O(nlog(n))$: l'algorithme *heapsort*. Le gain a été de 8%, ce qui est non négligeable mais en dessous de nos espérances.
On s'est donc inspiré de l'algorithme *quicksort*, lui aussi en $O(nlog(n))$,qui utilise un système de pivot pour coder un nouvel algorithme.

Si on se place dans un cas, où on a beaucoup d'observations, on atteindra le nombre limite d'observations et ensuite on cherchera à chaque fois l'observation la plus lointaine. Cette recherche de l'observation la plus lointaine correspond au calcul du maximum dans le tableau contenant les distances des observations. On peut conjecturer que cette détermination de maxiumum peut s'avérer très coûteuse si elle est faite à chaque itération après la limite. On peut alors penser que beaucoup de calcul sont fait inutilement.

L'idée de modification est de limiter le nombre de fois où l'on va chercher le maximum une fois qu'on sera à l'itération limite. La démarche va être de conserver en mémoire la distance de l'observation la plus lointaine. On écrasera les données de cette observation que si on rencontre une observation plus proche et seulement, dans ce cas là, on recalculera la distance maximale.

Le gain se retrouvera dans le fait qu'on effectue la recherche de l'extremum seulement si l'observation traitée est plus proche, contrairement l'algorithme de base où cela était fait à chaque itération.

Modification de la répartition des observation par quadrant Pour attribuer un quadrant à une observation, il suffit de savoir la position de cette observation par rapport au point de grille analysé. On connaît la latitude et la longitude de chacun des deux acteurs.
Dans la routine de référence, la détermination des quadrants se faisait en 2 étapes. La première permettait si l'observation se situait à l'est ou à l'ouest du point de grille et la deuxième si elle était au nord ou au sud. Ces opérations faisaient intervenir des fonctions internes au code qui mélaient des multiplications et additions de sinus et de cosinus entre les latitudes et les longitudes.

- 1ère étape : on calcule 2 sinus et 2 cosinus et on fait 3 multiplications et un addition
- 2ème : on multiplie un cosinus et un sinus

L'analyse des équations montre qu'il s'agit d'une erreur scientifique dans la formulation trigonométrique impliquant notamment une erreur sur la définition des angles à utiliser. Ces équations ont, par ailleurs, induit trop de calculs.
On a donc ramené ce calcul au plus simple :

- On compare juste les latitude de l'observation et du point pour savoir si on est dans la partie est ou ouest.
- On détermine le signe du sinus de la différence des longitudes pour savoir si on est au nord ou au sud.

2.4 Résultats et commentaires

2.4.1 Résulats

Le profil résultant des modifications de code sur un processeur est le suivant :

```
Name of the executable : ./MASTERODB
Number of MPI-tasks : 1
Number of OpenMP-threads : 1
Wall-times over all MPI-tasks (secs) : Min=313.290, Max=313.290, Avg=313.290, StDev=0.000
Routines whose total time (i.e. sum) > 1.000 secs will be included in the listing
  Avg-%   Avg.time   Min.time   Max.time   St.dev  Imbal-%   # of calls : Name of the routine
 18.52%     58.022     58.022     58.022    0.000    0.00%        25174 : CASGQA
 18.11%     56.725     56.725     56.725    0.000    0.00%        25174 : CASGRA
 10.83%     33.917     33.917     33.917    0.000    0.00%        37824 : CASPIA
  4.84%     15.169     15.169     15.169    0.000    0.00%      1512613 : MINV:GECO
  4.65%     14.553     14.553     14.553    0.000    0.00%      1512613 : CATRMA
  3.86%     12.097     12.097     12.097    0.000    0.00%      1563711 : CACOVA
  3.13%      9.812      9.812      9.812    0.000    0.00%            2 : SUSPECA
  2.37%      7.437      7.437      7.437    0.000    0.00%      1563711 : CAMERA
  2.30%      7.202      7.202      7.202    0.000    0.00%        50411 : CANADA
  2.27%      7.113      7.113      7.113    0.000    0.00%      1512613 : MINV:GEDI
```

Le premier bilan est de constater le gain au niveau du temps d'exécution de CANARI. Le code de référence mettait 426 secondes à tourner contre 313 secondes pour le code modifié, soit un gain de 26% en travaillant essentiellement sur deux routines.

En regardant le profil détaillé, on remarque que :
- CASGQA passe de 40.92% à 18.52%
- CASGRA passe de 13.20% à 18.11%
- Les nombre d'appels à ces deux routines passe de 1051806 à 25174

2.4.2 Commentaires

Tout d'abord, les expériences menées ont montrées que le gain, résultant de la modification de l'algorithme de tri était de 20%.
L'action de pousser les boucles réduits l'overhead d'appel, ce qui fait baisser sensiblement le temps d'exécution (environ 2.5%). De plus, au passage, on a pu gagné quelque secondes en améliorant l'adressage mémoire.
Le principe des vases communicants fait remonter, au niveau du pourcentage, la routine CASGRA et à moindre échelle la routine CASPIA. Les changements sur CASGRA sont une amélioration de la vectorisation, on peut donc penser que ces modifications seront plus visibles sur une machine vectorielle.
On note que ces modifications de codes ne modifient pas les résultats scientifiques.
Par ailleurs, il est apparu un bugg dans la méthode de référence pour répartir les observations par quadrants. En effet, après vérifications, on a remarqué que certaines observations n'étaient mises dans le bon quadrant avec la méthode de référence.

La modification de code a donc permi d'optimiser le code et, par la même occasion de corriger un bogue existant dans le code. Ceci entraîne donc une modification des résultats scientifiques.
La Figure(Fig(7.1)) en annexe est une carte de différence, pour l'humidité à 2 mètres, entre le code de référence et celui modifié. Toutes la surface blanche est une zone où il n'y aucune divergence dans les résultats. Les points colorés sont des zones de disparité, ces régions correspondent à des lieux où il n'y a pas beaucoup d'observations.
En effet, dans un lieu où on n'a pas beaucoup d'observations, l'isotropie du pannel sera privilégiée tandis que dans le cas contraire, on avantagera plutôt la proximité des observations.
On observe donc, que la quasi-intégralité du domaine ne subit pas de différence. Les changements produits sont en moyenne de l'ordre de 0.02%, ce qui peut être considéré comme négligeable pour l'impact sur les scores du modèle.

Remarque Une idée pour faire baisser le temps d'exécution a été de faire baisser le rayon de recherche des observations. Par défaut, le zone de recherche est de 1000km. Cette modification change les résultats scientifiques car il y a des zones où on ne prendra pas les mêmes observations. Pour des points de mer, on ne prend que les observations marines. Dans le cadre de l'exécution d'AROME pour la france métropolitaine, pour un endroit de la manche on considèrera par exemple une observation situé dans la mer du nord quand on garde un rayon de 1000km.

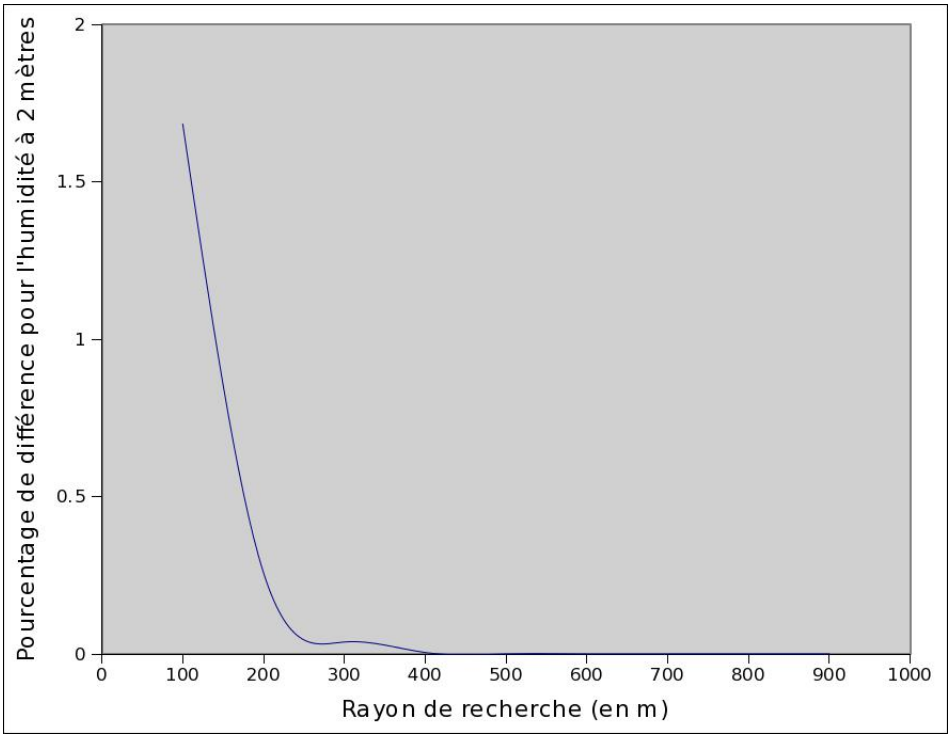

FIG. 2.2 – Evolution du pourcentage de différence pour l'humidité à 2 mètres en fonction du rayon de recherche

La courbe ci-dessus représente l'évolution du pourcentage de différence pour l'humidité à 2 mètres en fonction du rayon de recherche. Il apparaît que jusqu'à environ 400km, la différence de résultat est minime. Le profil résultant de l'exécution du code sur un processeur pour un rayon de

400km est le suivant :

```
Wall-times over all MPI-tasks (secs) : Min=141.390, Max=141.390, Avg=141.390, StDev=0.000
Routines whose total time (i.e. sum) > 1.000 secs will be included in the listing
  Avg-%   Avg.time   Min.time   Max.time   St.dev  Imbal-%   # of calls : Name of the routine
 16.03%     22.660     22.660     22.660    0.000    0.00%        25174 : CASGRA
 13.80%     19.507     19.507     19.507    0.000    0.00%        37824 : CASPIA
  9.59%     13.557     13.557     13.557    0.000    0.00%      1467580 : MINV:GECO
  9.36%     13.230     13.230     13.230    0.000    0.00%      1467580 : CATRMA
  6.24%      8.814      8.814      8.814    0.000    0.00%        25174 : CASGQA
```

A noter que le gain en temps est 55% avec une chute considérable de CASGQA qui est compréhensible car la routine n'a plus besoin d'analyser des observations lointaines. CASGRA ne bouge pas car son action est indépendante du rayon de recherche.
Cette modification ne sera pas validé car il aurait fallu faire une étude des scores sur une simulation de 15 jours. Par manque de temps, on ne se lancera pas dans ces manoeuvres.
La solution future sera d'introduire un indice terre-mer où on adaptera un rayon en fonction du lieu où on effectue la prévision et du modèle. En effet pour le modèle globale prenant en compte tout le globe terrestre, on pourra garder la zone de 1000km pour un point en plein milieu de l'atlantique par exemple. Tandis que dans les points de terre, il est clair que ce paramètre va devoir être diminuer afin d'augmenter la performance.

Chapitre 3

Réduction des niveaux verticaux

L'entête du profil ci-dessous résulte des modifications pour CANARI sur un processeur scalaire.

```
Profiling information for program='./MASTERODB', proc#1:
No. of instrumented routines called : 1604
Instrumentation started : 20110804 095258
Instrumentation   ended : 20110804 095823
Instrumentation overhead: 3.90%
Memory usage : 6596 MBytes (heap), 6221 MBytes (rss)
Wall-time is 313.29 sec on proc#1 (1 procs, 1 threads)
Thread#1:      313.29 sec (100.00%)
```

Le temps d'éxécution est bien sûr le même que précédemment. Une nouvelle information est la consommation mémoire du logiciel. Il utilise plus de 10Go pour un processeur, ce qui 10 fois trop pour le calculateur scalaire mis à notre disposition. On va donc essayer de réduire ce coût mémoire, tout en gardant les résultats scientifiques et en espérant, par ailleurs, un gain sur le temps d'exécution.

3.1 Analyse du contexte

Le logiciel CANARI a été codé, initialement, pour faire une analyse de toute l'atmosphère. Or, l'évolution du code a réduit le domaine d'action de CANARI. En effet, ce logiciel est utilisé, maintenant, uniquement pour analyser les champs de surface.

La figure ci-dessus montre comment l'atmosphère est modélisé. En effet, elle est découpé en plusieurs niveaux. Pour CANARI, on lit 60 niveaux dans le code de référence. On traite donc des données concernant le sommet de l'atmosphère alors qu'on analyse uniquement les champs de surface. Il en résulte une consommation superflu de calcul et de mémoire. En effet, on utilise dans le code référence :

- 6596 MBytes pour la mémoire heap, qui sert notamment aux allocations dynamique de mémoire
- 6220 MBytes pour la mémoire rss, qui donne une idée de la taille totale du programme

Il apparait donc que le logiciel, utilisé dans une lecture des champs de surface, n'a besoin que des niveaux les plus proches du sol.

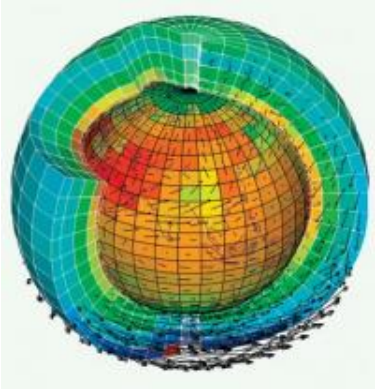

FIG. 3.1 – Découpage en niveaux de l'atmosphère

Il existe une contrainte qui impose que les niveaux lus dans le modèle doivent coïncider avec tous ceux du fichier de conditions initiales. On veut modifier le code afin que les niveaux lus dans le modèle correspondent uniquement avec les niveaux les plus proches du sol dans le ficiher de condition initiale.
On veillera, tout de même, à ce que le fichier issu de la réduction des niveaux verticaux reste lisible par un programme n'ayant pas connaissance de cette modification. Ce fichier doit aussi contenir les niveaux n'ayant pas été lu par le modèle afin de ne pas perturber le chaîne d'assimilation.

3.2 Modifications mises en place

Les modifications du code CANARI repose principalement sur l'introduction de 2 nouvelles variables. La première est une variable logique LIOLEVG qui permet de choisir si on veut, ou non, que les niveaux du modèle coïncide avec ceux du fichier de conditions initiales. Dans le cas où, LIOLEVG = .TRUE. on fait coïncider les niveaux.
L'autre variable est NIOLEVG qui correspond au nombre de niveaux dans les fichier entrées/sorties du modèle. NFLEVG, déja présente dans le code, est le nombre de niveaux lus dans le modèle.
On a alors : LIOLEVG = .TRUE. implique NFLEVG=NIOLEVG.
LIOLEVG=.FALSE. lève cette contrainte. Les tests de cohérence avec le fichier de conditions intiales se font désormais avec la variable NIOLEVG qui possède tous les niveaux. L'en-tête du fichier de sortie est écrit en fonction de NIOLEVG.
L'utilisation de ces variables reste optionelle. Le logiciel lit par défaut tous les niveaux du fichier. Dans le cas de l'écriture de la ligne de commande qui permet de lire un nombre réduit de niveaux, l'utilisateur choisira désormais le nombre NFLEVG qu'il souhaite.

La réduction des niveaux verticaux lus a engendrée la modification d'une dizaine de routines. Les changements sont :

- La définition et l'initialisation des nouvelles variables
- Des changements d'indices pour ne traiter que le sous ensemble de niveaux
- Des opérations à effectuer sur NIOLEVG au lieu de NFLEVG
- Des tests sur le fichier de sortie et sur la date

3.3 Résultats et commentaires

Les expériences menées ont montré qu'on pouvait descendre jusqu'à 2 niveaux verticaux analysées sans modifier les résultats scientifiques. Avec un unique niveau analysé, les champs divergent. Théoriquement, on doit pouvoir descendre à 0 niveau lû mais en pratique la réduction ne fonctionne pas en-dessous de 2 niveaux. En effet, CANARI analyse par exemple la température de surface mais aussi la température à 2 mètres ce qui nécessite d'avoir des information une petite couche de l'atmosphère. Dans l'avenir, l'amélioration dans CANARI nécessitera un affinage de ces quelques niveaux proches du sol.

Le profil résultant des modifications pour CANARI cumulées à celle pour 2 niveaux verticaux sur un processeur sur un calculateur scalaire est le suivant :

```
Profiling information for program='./MASTERODB', proc#1:
No. of instrumented routines called : 1604
Instrumentation started : 20110804 094735
Instrumentation   ended : 20110804 095151
Instrumentation overhead: 5.22%
Memory usage : 902 MBytes (heap), 502 MBytes (rss), 0 MBytes (stack), 0 (paging)
Wall-time is 243.83 sec on proc#1 (1 procs, 1 threads)
Thread#1:      243.83 sec (100.00%)

    (self)        (sec)        (sec)        (sec)       # of calls    ms/call
1    23.78       57.988       57.988       58.014          25174        2.30    CASGQA@1
2    23.24      114.662       56.674       59.475          25174        2.36    CASGRA@1
3    13.63      147.885       33.224       33.268          37824        0.88    CASPIA@1
4     6.15      162.871       14.985       16.139        1512613        0.01    MINV:GECO@1
5     5.97      177.416       14.545       15.697        1512613        0.01    CATRMA@1
6     4.94      189.459       12.043       13.205        1563711        0.01    CACOVA@1
7     3.04      196.875        7.416        8.506        1563711        0.01    CAMERA@1
8     2.91      203.967        7.092        8.145        1512613        0.01    MINV:GEDI@1
9     2.90      211.043        7.076      238.858          50411        4.74    CANADA@1
10     1.56      214.839        3.796       29.031        1512613        0.02     MINV@1
```

Tout d'abord, on remarque que le temps d'exécution est de 243 secondes contre 313 secondes pour le jeu de modifications pour CANARI et 426 secondes pour le code référence.
On a donc gagné 22% par rapport aux modifications précédentes, ce qui nous amène a un gain de 43% par rapport au code de référence. On perdait donc beaucoup de temps à lire des niveaux en haut de l'atmosphère qui n'allait en rien nous aider à faire des prévisions pour la surface du globe. Par ailleurs, on peut voir qu'on utilise maintenant 902MBytes pour la mémoire heap et 502MBytes pour la rss. Soit un gain mémoire de 86% pour heap et 92% pour rss.
Cette modification pour les niveaux verticaux permet donc de faire une économie considérable en mémoire qui se conjugue à un gain de temps non négligeable tout en conservant les résultats scientifiques. On note aussi que le trio de tête des routines les plus coûteuses reste inchangé avec CASGQA, CASGRA et CASPIA.

Chapitre 4

Travail pour CASPIA sur machine vectorielle et scalaire

La différence entre un processeur vectoriel et un processeur scalaire réside dans la capacité à traiter une instruction plus ou moins en parallèle. Le processeur scalaire va manipuler un élément de données tandis que le vectoriel peut accéder à de multiples éléments de données.
L'exemple suivant nous montre comment un code FORTRAN va être interpreter par les différents types de processeurs. Le code est :
DO I = 1, N
A(I) = B(I) + C(I)
ENDDO
traité de la manière suivante par un processeur :

scalaire	vectoriel
INITIALISER I = 1	
10 LIRE B(I)	INITIALISER V = (1,...,N)
LIRE C(I)	LIRE B(V)
ADDITIONNER B(I) + C(I)	LIRE C(V)
STOCKER B(I) + C(I) dans A(I)	ADDITIONNER B(V) + C(V)
INCREMENTER I en I + 1	STOCKER B(V) + C(V) dans A(V)
SI I <= N ALLER À 10	STOP
STOP	

Actuellement, le code opérationnel utilise CANARI sur une machine munie de processeurs vectoriels qui monopolise 8 processeurs pendant son exécution. Les optimisations faites précédemment permettent un gain de 10% du temps d'exécution avec un CASPIA occupant 66% de ce temps.
Il apparait à ce moment de l'optimisation du code qu'il sera difficile de gagner plus dans CASGRA et CASGQA sur un calculateur scalaire et le temps d'exécution de ces sous-programmes est moindre devant celui de CASPIA sur calculteur vectoriel. La routine CASPIA est, quant à elle, à 14% sur machine scalaire et 66% sur machine vectorielle. Le profil sur machine vectorielle pour 8 processeurs, après les modifications, est le suivant :

```
*----------------------*
  FTRACE ANALYSIS LIST
*----------------------*
Execution Date : Thu Aug 11 14:23:26 2011
Total CPU Time : 0:45'40"287 (2740.287 sec.)

PROC.NAME  FREQUENCY  EXCLUSIVE         MFLOPS AVER.  VECTOR    I-CACHE  CONFLICT
                      TIME[sec](  % )          V.LEN   TIME     MISS    NETWORK

caspia       1577772  1818.087( 66.3)   3.7    33.8   1379.106  2.791   1113.399
minv.geco    1512667   232.571(  8.5)   64.3   27.9   110.584   23.984  93.593
casgqa          4208   156.139(  5.7)   969.4  203.7  108.984   0.015   67.363
minv.gedi    1512667    94.592(  3.5)   67.9   22.0   50.222    10.630  43.268
casgra          4208    89.643(  3.3)   967.6  120.2  24.205    0.011   7.967
```

On va dorénavant se concentrer sur l'optimisation de ce sous-programme.
A noter que le total CPU Time ne représente pas le temps d'exécution réel de CANARI. Ici CANARI met 8 minutes 36 (soit 516 secondes) à se dérouler.

4.1 Ecremage des observations

La routine CASPIA permet de sélectionner les mesures les plus judicieuses dans une couche de pression, parmi les observations choisies par CASGQA et CASGRA.
On part donc d'un ensemble d'observations dont on doit extraire les meilleures. La structure du code de référence est une série de petites boucles. Chaque boucle prend en entrée un tableau, on effectue alors un test, s'il est vrai on incrémente un compteur et on remplit alors un nouveau tableau avec ce nouvelle adressage.
Sur une machine vectorielle, l'organisation de cette routine n'est pas du tout appropriée. En effet, on peut voir la longueur de vectorisation du sous-programme dans la colonne AVER.V.LEN. CASPIA a donc un longueur de vectorisation de 33.8 ce qui est très court comparée à celle de CASGQA par exemple. L'addition de petites boucles où interviennent de nombreux tableaux intermédiaires qui entrainent, par ailleurs, un adressage mémoire indirect : toutes ces choses cumulées n'encouragent pas la vectorisation au sein de cette routine ce qui ralentit considérablement l'éxécution sur calculateur vectoriel.

Pour adapter le code à l'architecture du calculateur, on va donc modifier la manière de sélectionner ces observations. Pour ce faire on va tout d'abord regrouper plusieurs boucles. Au lieu d'utiliser le système du compteur incrémenté après chaque test vrai. On va utiliser un seul tableau loqique qui sera vrai si le contrôle est satisfait, et faux sinon. On ne remplira, dorénavant, les tableaux contenant les informations sur les observations qu'à la fin de la sélection sous la condition que le tableau logique soit vrai.

De plus, pour améliorer encore cette vectorisation, on va effectuer le principe du 'loop collapsing' qui, dans notre cas, consiste à fusionner deux boucles : l'une sur les observations, l'autre sur les mesures de chaque observations. Ceci aidera à accroitre la longueur de vectorisation.

4.2 Répercutions des modifications sur les deux types de machines

4.2.1 Analyses et commentaires

Le profil résultant des modifications pour CASPIA sur 8 processeurs sur une calculateur vectoriel est le suivant :

```
*----------------------*
  FTRACE ANALYSIS LIST
*----------------------*
Execution Date : Thu Sep  1 10:03:06 2011
Total CPU Time : 0:33'00"371 (1980.371 sec.)

PROC.NAME  FREQUENCY  EXCLUSIVE         MFLOPS  AVER.  VECTOR   I-CACHE  CONFLICT
                      TIME[sec](  % )           V.LEN   TIME    MISS     NETWORK

caspia_vec   1577772  1099.730( 55.5)      1.4  85.9   70.135   5.542    48.672
minv.geco    1512635   243.328( 12.3)     61.5  27.9   107.134  32.564   90.311
casgqa          4208   176.186(  8.9)   1001.0 204.0   122.961  0.009    73.535
minv.gedi    1512635    97.008(  4.9)     66.5  22.0   49.134   12.849   42.096
catrma       1512635    70.363(  3.6)    236.5  10.6   45.004   4.609    35.841
```

Tout d'abord, le temps d'exécution est passé de 516 secondes secondes à 345 secondes, ce qui représente un gain de 33,1%. Par ailleurs, on peut remarquer que la colonne AVER.V.LEN qui traduit la longueur de vectorisation moyenne a varié de 33.8 à 85.9. Les modifications ont donc augmenté le phénomène de vectorisation qui est à l'origine de l'accélération du code sur machine vectorielle.
En revanche, en faisant des tests sur une machine scalaire, on observe que le code est ralenti d'environ 15%. Cette variation nous pousse à écrire 2 versions de la routine CASPIA : une pour machine vectorielle et une pour machine scalaire. On gardera sur machine scalaire la version initiale de CASPIA qui convient aux besoins de l'architecture. Ce problème doit néanmoins être investiguer dans l'avenir car il est clair qu'il est possible d'optimiser ce sous-programme sur calculateur scalaire.

4.2.2 Tests sur la scalabilité

Présentation du problème Théoriquement, lorsqu'on trace un 'speedup', autrement dit l'évolution du quotient du temps de référence sur le temps d'exécution en fonction du nombre de processeurs, on devrait obtenir une droite. En réalité, cette situation n'est jamais de la sorte. La performance a tendance à chuter quand on augmente les processeurs. On mesure la scalabilité au fait que la courbe expérimentale suit plus ou moins la courbe idéale : plus la courbe expérimentale suit la courbe idéale plus la scalabilité est importante.
Vous retrouverez en annexe (Fig(7.3) et Fig(7.4)) les courbes de scalabilité sur un calculateur vectoriel et scalaire.

Commentaires Tout d'abord, on remarque que l'exécution sur une machine vectorielle est plus scalable que celle sur un calculateur scalaire.
Les profils de référence sur un et 8 processeurs sont disponibles en annexe ainsi que ceux après modifications. On peut expliquer le fait que la scalabilité sur le code de référence soit insuffisante sur machine scalaire par la remontée de sous-programme (comme ODBMP ou LFIFER) traduit la difficulté de communication entre les processeurs. Ce problème est atténué après les modifications. On observe malgré tout que la routine TRGTOL remonte quand le nombre de processeurs augmente. Néanmoins, pour un processeur scalaire le logiciel met 243 secondes à tourner tandis que sur 8 processeurs vectoriels le temps d'exécution est de 345 secondes.
Malgré une scalabilité plus forte sur calculateur vectoriel, il est clair qu'utiliser un seul processeur scalaire est un progrès considérable. Car les processeurs laissés libre peuvent être utilisés à bon escient par les autres parties d'AROME. En effet, en introduction, on parlait des étapes du modèle comme la minimisation ou le screening. En utilisant un processeur scalaire pour CANARI, on libère des processeurs pour ces applications et, si elles sont scalables, elles pourraient voir leur temps d'exécution se réduire.
La solution serait donc de mettre en place un système hybride d'ordinateurs possédant à la fois des processeurs vectoriels et scalaires, c'est actuellement un des nombreux projets de Météo-France.

Chapitre 5

Conclusions et perspectives

Conclusions

Les prévisionnistes disposent depuis 2008 du modèle AROME qui possède une maille fine de 2.5km. Pour satisfaire la demande de plus en plus exigente en matière de rapidité des précisions, les chercheurs de Météo-France ne cesse de renouveler les codes de calculs.
Le logiciel CANARI est, aujourd'hui, en charge de l'analyse de surface, il était nécessaire de l'améliorer afin qu'il puisse répondre aux demandes de l'opérationnel. Plusieurs approches ont aidé à optimiser ce logiciel.
Dans un premier temps, des modifications purement algorithmiques qui ont permis un gain de temps d'environ 20% sur machine scalaire, grâce notamment à l'écriture d'un nouvel algorithme de tri. Puis, il était nécessaire de pouvoir adapter le logiciel à son mode d'utilisation. Dans le cas de l'analyse de surface, la réduction des niveaux verticaux a engendré un gain mémoire considérable (environ 90%), ce qui a limité les coûts de calculs et donc accéléré le code d'environ 20%.
On a donc pu optimiser ce code grâce à des techniques algorithmiques traditionnelles mais aussi en comprenant le fonctionnement du logiciel et en l'adaptant à l'utilisation qui en est faite. En effet, avec l'évolution des modèles et des logiciels de Météo-France, CANARI continuait d'effectuer des opérations dont il n'était plus responsable.

Perspectives

Pour continuer dans cette voie d'optimisation, il faudrait notamment continuer à travailler la partie sur la diminution du rayon de recherche. Le gain de temps est flagrant et il est primordial si on veut améliorer ce logiciel de modifier ce paramètre. Une alternative serait la mise en place de l'indicateur terre-mer, où on adapterait le rayon de recherche en fonction du domaine où effectue l'analyse.
Par ailleurs, la possibilité de travailler sur deux types de calculateurs a montré la nécessité d'avoir des systèmes hybrides. Si pour le logiciel CANARI, un processeur scalaire va plus vite que 8 processeurs vectoriels. Il peut se passer le cas inverse pour un autre code. D'où le besoin de pourvoir choisir le type de calculateur sur le quel va tourner un logiciel. S'il est clair, que l'avenir de CANARI est de tourner sur un calculateur scalaire, il faudra cependant améliorer sa scalabilité qui peut être encore optimiser afin de tirer le meilleur des ressources de calculs. On peut aussi utiliser des fonctions propres au calculateur, par exemple pour la machine scalaire il existe des fonctions

trigonométriques optimisés pour la machine. On peut imaginer que ces fonctions seraient utiles dans CASGRA qui fait beaucoup de calculs trigonométriques.

Remerciements

Je tiens à remercier Ryad El Khatib pour la confiance qu'il m'a accordé, pour son encadrement constructif et sa patience envers toutes mes questions. Je remercie aussi Françoise Taillefer, Eric Sevault et Philippe Marguinaud pour leurs différentes explications. Je remercie enfin toute l'équipe du GMAP pour son accueil chaleureux et stimulant.

Chapitre 6

Bibliographie

Taillefer, F., 2002, *Canari Technical Documentation based on ARPEGE cycle **CY25T1** (AL25T1for ALADIN)*

Chapitre 7

Annexes

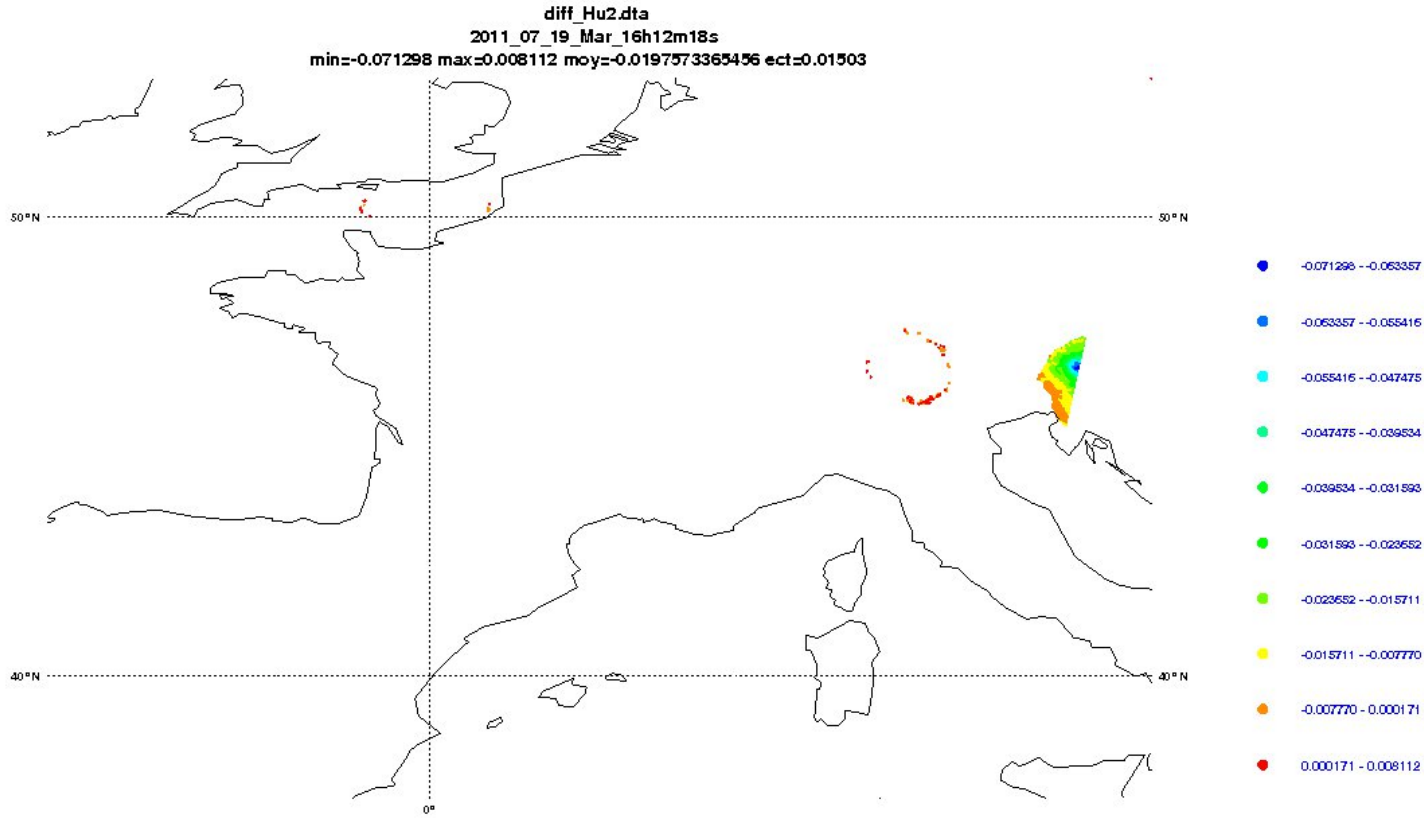

FIG. 7.1 – Carte de différence pour l'humidité à 2 mètres

Le profil de référence pour 1 processeur scalaire est le suivant :

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls	Name of the routine
40.07%	173.276	173.276	173.276	0.000	0.00%	1051806	CASGQA
12.83%	55.494	55.494	55.494	0.000	0.00%	1051806	CASGRA
8.58%	37.103	37.103	37.103	0.000	0.00%	1577772	CASPIA
3.75%	16.207	16.207	16.207	0.000	0.00%	1512170	MINV:GECO
3.42%	14.790	14.790	14.790	0.000	0.00%	1512165	CATRMA

Le profil de référence pour 8 processeurs scalaires est le suivant :

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls	Name of the routine
36.38%	52.471	0.057	60.039	21.178	99.91%	224	ODBMP:ODBMP_IGLOBAL
5.17%	7.458	0.000	59.662	21.094	100.00%	74	LFIFER_MT
15.71%	22.652	13.793	31.149	6.077	55.72%	1051806	CASGQA
6.52%	9.399	0.576	18.725	6.263	96.92%	192	TRGTOL
5.07%	7.318	5.501	8.815	1.157	37.60%	1051806	CASGRA

Le profil après modifications pour 1 processeur scalaire est le suivant :

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls	Name of the routine
22.69%	57.373	57.373	57.373	0.000	0.00%	25174	CASGRA
22.55%	57.001	57.001	57.001	0.000	0.00%	25174	CASGQA
14.88%	37.622	37.622	37.622	0.000	0.00%	1577772	CASPIA
6.50%	16.424	16.424	16.424	0.000	0.00%	1512614	MINV:GECO
5.92%	14.973	14.973	14.973	0.000	0.00%	1512613	CATRMA

Le profil après modifications pour 8 processeurs scalaires est le suivant :

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls	Name of the routine
15.96%	7.396	3.672	12.214	3.049	69.94%	25184	CASGQA
11.52%	5.338	0.097	12.064	3.916	99.20%	192	TRGTOL
16.18%	7.498	5.645	9.058	1.202	37.68%	25184	CASGRA
10.67%	4.946	3.989	5.745	0.632	30.57%	1577772	CASPIA
4.73%	2.194	1.716	2.868	0.425	40.17%	1512614	MINV:GECO

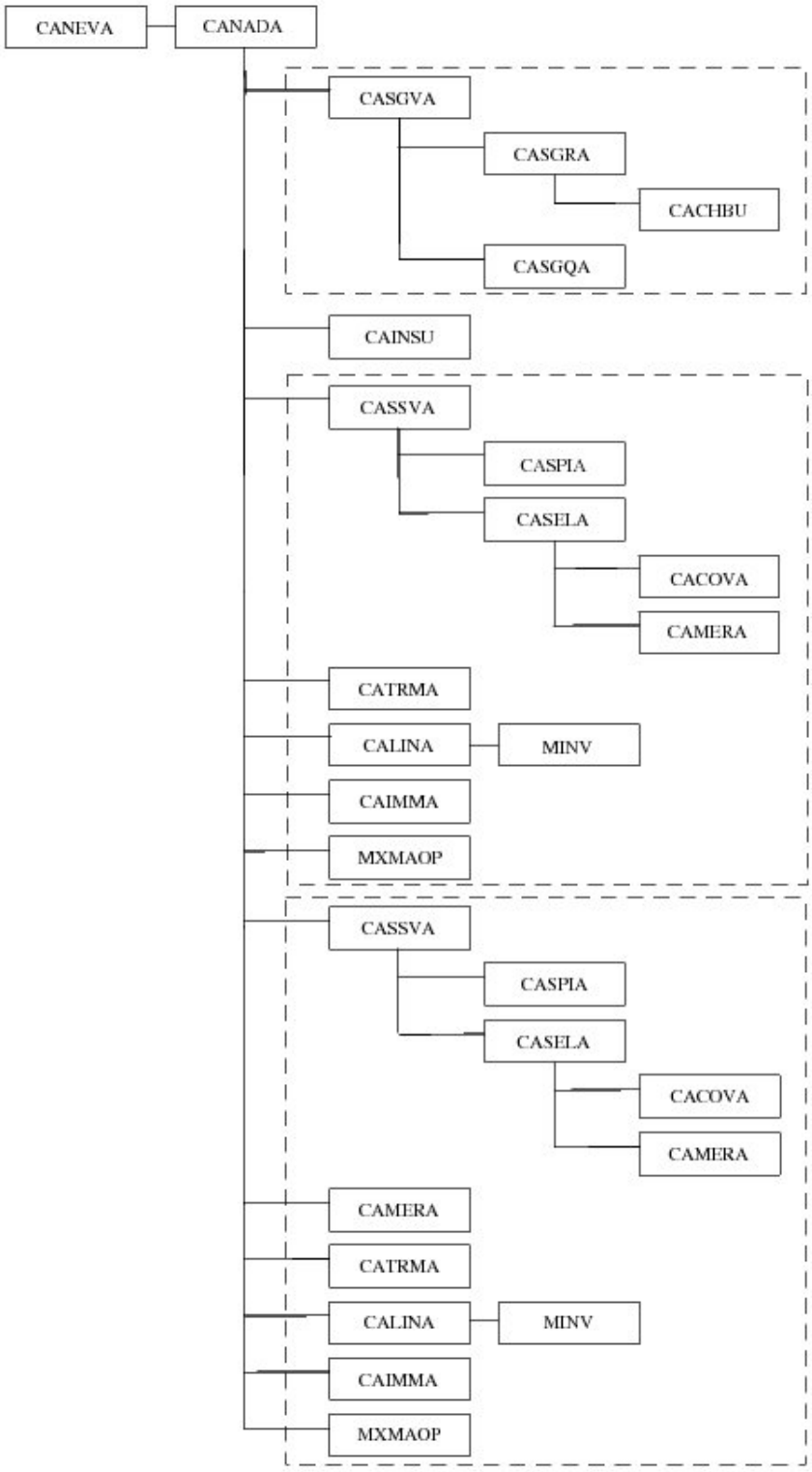

FIG. 7.2 – Extrait de l'arbre d'appel de Canari

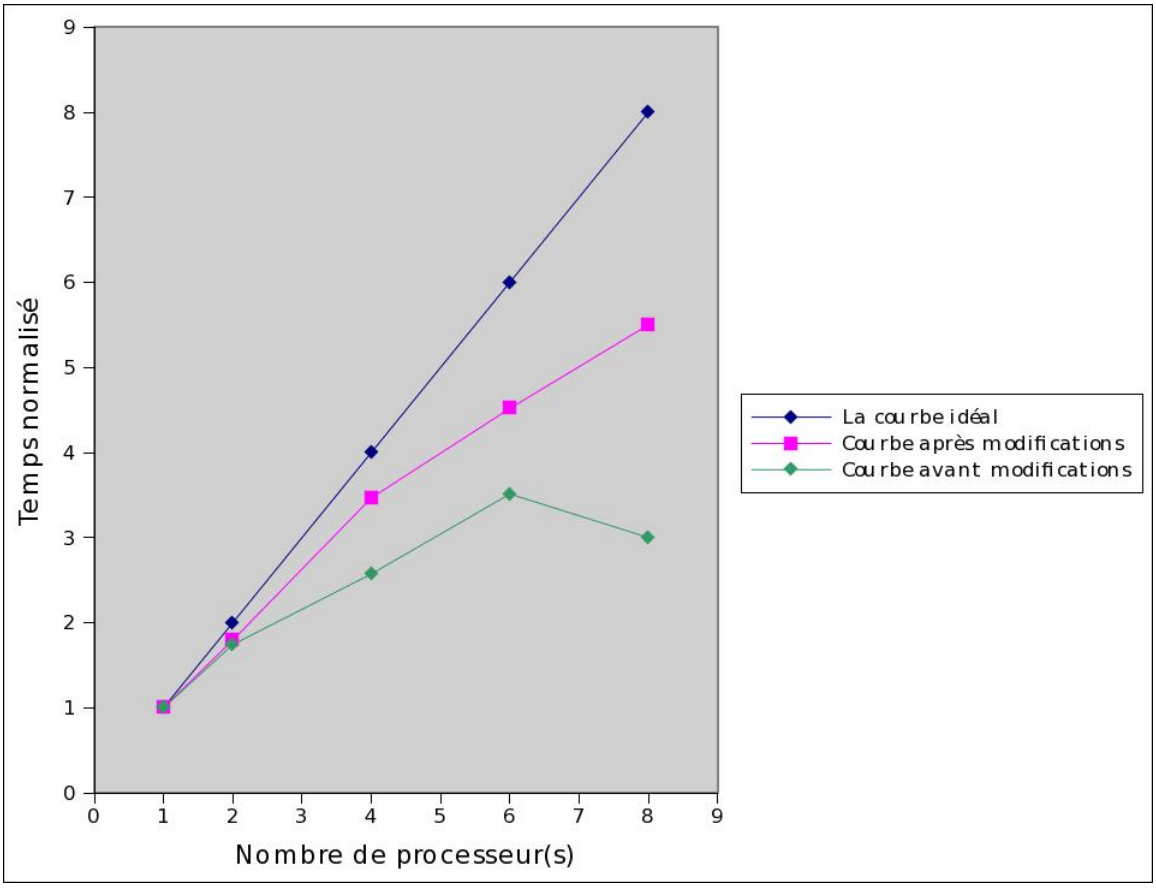

FIG. 7.3 – Courbe de scalabilité pour un calculateur scalaire

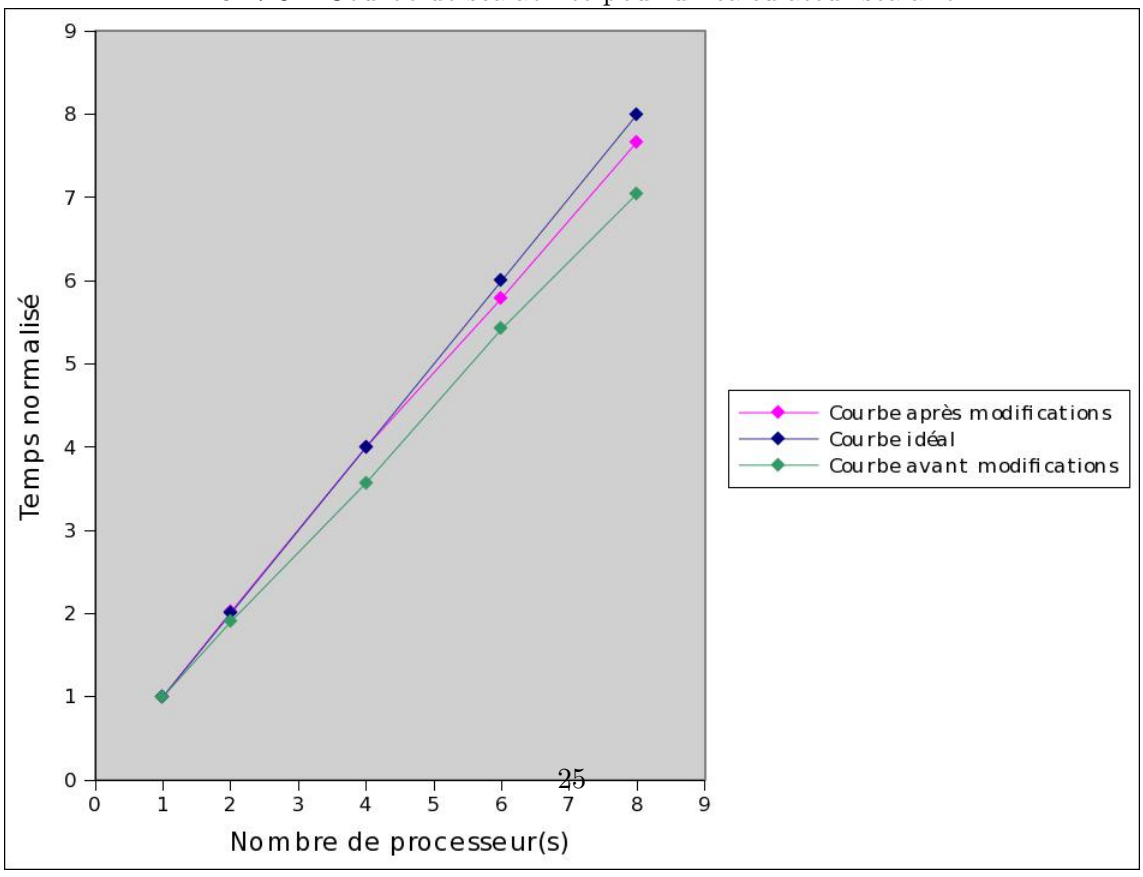

FIG. 7.4 – Courbe de scalabilité pour un calculateur vectoriel