HABILITATION A DIRIGER DES RECHERCHES

présentée devant

L'Université de Rennes 1
Institut de Formation Supérieure
en Informatique et en Communication

par

Jocelyne Erhel
IRISA
Campus Universitaire de Beaulieu
35042 Rennes Cedex

VITESSE ET PRECISION EN CALCUL SCIENTIFIQUE

soutenue le 26 Juin 1992 devant le jury composé de

M.	Jacques Lenfant	Président
M.	Iain Duff	Rapporteurs de soutenance
M.	Ahmed Sameh	
Mme	Françoise André	Examinateurs
M.	Michel Crouzeix	
M.	Bernard Philippe	
M.	Christian Saguez	

Rapporteurs du dossier avant soutenance

M.	Iain Duff
M.	Roland Glowinski
M.	Yves Robert

Remerciements

Je tiens à remercier de tout cœur Bernard Philippe dont l'aide et les encouragements m'ont été précieux.

Tous mes remerciements vont à Jacques Lenfant qui me fait l'honneur de présider ce jury.

I am really honoured by the presence of Iain Duff and Ahmed Sameh.

Je suis particulièrement touchée par la participation au jury de Françoise André, Michel Crouzeix et Christian Saguez.

Que les rapporteurs de mon dossier, Roland Glowinski et Yves Robert, qui n'ont pu participer au jury, reçoivent ici l'expression de ma gratitude.

Enfin j'exprime ma plus vive sympathie à mes collègues de l'INRIA et l'IRISA et à tous ceux qui m'ont apporté leur soutien dans mes activités de recherche.

Résumé

L'avènement de nouvelles architectures parallèles a conduit les numériciens à concevoir de nouveaux algorithmes efficaces et performants sur ces machines. Toutefois, le souci de rapidité ne doit pas faire oublier celui, fondamental, de stabilité.

La première partie est consacrée à nos travaux sur les algorithmes vectoriels et parallèles. La cible essentielle est la méthode des elements finis. Pour cela, nous avons décomposé les logiciels d'éléments finis en algorithmes de base. Ces briques ont été transformées pour les adapter aux architectures vectorielles et parallèles.

La deuxième partie traite de nos travaux actuels sur la qualité numerique. A l'initiative du CNES, l'IRISA a demarré un projet d'atelier de qualité numérique, dénommé AQUARELS, en cours de réalisation par la société SIMULOG. Nous présentons les outils de l'atelier et les études effectuées dans ce cadre.

Table des Matières

1 Introduction. **1**

2 Parallélisation d'algorithmes numériques. **3**

2.1 Etude des algorithmes de base 6

2.1.1 Assemblage d'un second membre ou d'une matrice 6

2.1.2 Multiplication d'une matrice creuse par un vecteur 7

2.1.3 Résolution d'un système triangulaire creux 9

2.1.4 Résolution d'un système linéaire creux 11

2.2 Méthodes de sous-domaines 19

2.3 Applications industrielles 21

2.4 Expertise de calculateurs parallèles 24

2.4.1 Projet français 24

2.4.2 Benchmark pour calculateurs vectoriels 25

2.4.3 Calculateur ST100 25

2.4.4 Calculateur ETA10 26

2.5 Simulation d'architectures parallèles 28

2.5.1 Logiciel MUPI 28

2.5.2 Logiciel CREM 28

3 Qualité numérique des résultats. **31**

3.1 Aquarels 32

3.1.1 Résultats de l'enquête 32

3.1.2 Outils existants 33

3.1.3 Spécifications et conception 34

3.2 Etude des perturbations 36

3.3 Système expert 41

4 Perspectives. **43**

Table des Tableaux

2.1 Caractéristiques des matrices creuses. 16

2.2 Performances (en MFLOPS) du produit matrice creuse-vecteur sur CRAY2 17

2.3 Résultats de ICCG avec fronts pair-impair sur CRAY1 18

2.4 Résultats de EBE sur CRAY2 18

2.5 temps CPU (en secondes) et efficacité vectorielle (rapport du temps en mode scalaire sur le temps en mode vectoriel) sur CRAY2 monoprocesseur 23

2.6 temps écoulé (en secondes) et accélérations (speed-up) sur un CRAY2 avec du parallélisme local. 23

3.1 Estimation de conditionnements et d'erreurs par perturbations des données 38

3.2 Estimation de régularité et de conditionnement par perturbation des données 38

Table des Figures

3.1 Perturbation des données d'un système linéaire 39
3.2 Perturbation des données d'une équation du second degré . . 40

1
Introduction.

L'avènement de nouvelles architectures parallèles a conduit les numériciens à concevoir de nouveaux algorithmes efficaces et performants sur ces machines. Toutefois, le souci de rapidité ne doit pas faire oublier celui, fondamental, de stabilité.

La première partie est consacrée à nos travaux sur les algorithmes vectoriels et parallèles. La cible essentielle est la méthode des éléments finis. Pour cela, nous avons décomposé les logiciels d'éléments finis en algorithmes de base. Ces briques ont été transformées pour les adapter aux architectures vectorielles, voire parallèles dans certains cas. En général, l'algorithme de départ n'est pas modifié, les dépendances de données sont toutes respectées. Ceci garantit une stabilité numérique inchangée. Le nouvel algorithme exploite le parallélisme intrinsèque en réordonnant les éléments du maillage par exemple. Nous avons également participé à la conception de nouvelles méthodes, dites de sous-domaines, qui définissent un schéma de discrétisation spécialement étudié pour le parallélisme. Pour prouver l'efficacité des algorithmes vectoriels et parallèles ainsi mis au point, il est important de les utiliser dans un contexte réel. C'est pourquoi nous avons choisi d'illustrer nos idées dans des logiciels d'éléments finis en mécanique des fluides.

De même, l'expérimentation sur des architectures diverses est un point important pour aider à dégager les structures parallèles les plus efficaces. Aussi avons-nous développé un benchmark de logiciels numériques pour expertiser les calculateurs vectoriels japonais. Ce benchmark a ensuite servi à tester des mini-super-ordinateurs. Mais nos travaux ont débuté avant l'arrivée en France des superordinateurs. D'où la construction d'un simulateur d'architecture parallèle puis d'un simulateur de la bibliothèque parallèle des systèmes CRAY. Ces simulations ont permis, outre des expériences sur les algorithmes parallèles, un approfondissement de nos connaissances sur le fonctionnement détaillé des machines.
La deuxième partie traite de nos travaux actuels sur la qualité numérique.

Les questions de stabilité et de précision sont toujours présentes lors de la conception de nouveaux algorithmes. Or il existe à l'heure actuelle peu d'outils faciles à utiliser pour contrôler la précision des résultats. Il est pourtant important de connaître le degré de fiabilité d'un calcul. A l'initiative du CNES, l'IRISA a démarré un projet d'atelier de qualité numérique, dénommé AQUARELS. La première phase de cette étude comprenait une enquête auprès des usagers du calcul scientifique pour évaluer leurs besoins en contrôle et assurance qualité et un état de l'art sur le domaine. Les résultats de ces deux enquêtes ont servi de base à l'élaboration des spécifications de l'atelier. Il s'agit de construire une boîte à outils, facile d'utilisation et couvrant les besoins de la plupart des usagers. Il n'est pas question d'automatiser le processus de développement d'un logiciel scientifique. Mais une structure intégrée et conviviale permet d'utiliser divers outils tels que l'arithmétique d'intervalles, les méthodes statistiques de perturbations ou le calcul formel. Le projet CALCPAR participe à la conception et la réalisation de l'atelier qui sont confiées à la société SIMULOG.

L'atelier inclut un outil permettant de perturber aléatoirement les données ou les opérations d'un logiciel et d'analyser statistiquement les résultats de ces perturbations. Nous avons repris ces méthodes de perturbation pour approfondir leur utilisation. Nous avons ainsi défini une méthodologie pour détecter les instabilités numériques et mesurer le conditionnement d'un problème.

Les bibliothèques numériques sont si complexes qu'il est parfois difficile de trouver la meilleure méthode pour résoudre un problème donné. Des systèmes d'aide intelligents sont actuellement en cours de développement pour proposer avec explications à l'appui une méthode adaptée au problème décrit par l'utilisateur. Par contre, il n'existe pas d'outil pour aider à choisir une méthode ou un outil en fonction de la précision voulue ou pour analyser un échec numérique tel que la divergence d'itérations. Nous commençons une étude qui vise à concevoir un système expert d'aide dans les trois situations suivantes : choix de méthode, validation de résultat, analyse d'échec. Nous limitons le champ d'expertise aux problèmes aux valeurs propres qui représentent déjà un domaine suffisamment vaste et complexe.

2
Parallélisation d'algorithmes numériques.

Nos travaux de recherche sont centrés sur la conception et l'expérimentation d'algorithmes parallèles numériques. Ici, la notion de parallélisme est prise au sens large, englobant celle de vectorisation. Les algorithmes étudiés sont liés aux méthodes d'éléments finis. Ces méthodes sont utilisées pour résoudre des équations aux dérivées partielles (EDP). Le domaine de calcul est maillé par des éléments qui peuvent être par exemple des triangles en 2D. Ce maillage est non structuré, c'est-à-dire que la géométrie des triangles est quelconque. Les inconnues du système d'EDP sont discrétisées par décomposition sur une base de fonctions élémentaires construites à partir du maillage, conduisant à un problème de dimension finie. Le cas linéaire se traduit par un système linéaire où la matrice et le second membre sont calculés par assemblage de tableaux élémentaires. Dans le cas non linéaire, une méthode de type Newton modifié est en général utilisée, ce qui implique le calcul d'un jacobien, donc d'une matrice de même type que précédemment. Ces matrices d'éléments finis sont d'ordre élevé et creuses. De plus, l'irrégularité de la structure provient de la non structuration du maillage. Un stockage plein est prohibitif à la fois pour la place mémoire et le nombre d'opérations. Il faut donc un stockage adapté à la structure creuse non régulière de la matrice. Ces méthodes d'éléments finis se décomposent en plusieurs étapes dont certaines sont cruciales pour les performances. Nous nous sommes intéressés à deux phases fondamentales :

- Assemblage d'un second membre et/ou d'une matrice creuse.
- Résolution d'un système linéaire creux.

L'étape de résolution se décompose à son tour en briques de base, dont nous avons étudié les briques suivantes :

- Produit d'une matrice creuse par un vecteur.
- Résolution d'un système triangulaire creux.
- Préconditionnement d'un système linéaire.

Cette modularité permet d'inclure facilement ces algorithmes dans une bibliothèque numérique et de les utiliser dans des contextes très différents. Nous avons notamment collaboré avec l'équipe MODULEF pour modifier certains modules ou ajouter de nouveaux modules dans la bibliothèque du même nom.

La méthodologie utilisée est la même pour toutes ces briques de base. Il faut d'abord analyser les dépendances de données, en déterminant un graphe orienté dont les nœuds sont les étapes de calculs et dont les arêtes relient deux nœuds dépendants. Dans les algorithmes matriciels étudiés, le graphe de dépendances est fortement corrélé au graphe non orienté de la matrice creuse dont l'ordre est celui de la matrice (a_{ij}) et dont les arêtes relient les nœuds i et j tels que le cœfficient a_{ij} soit non nul. Par exemple, le calcul d'une inconnue x_i dépendra de celui de l'inconnue x_j si le cœfficient a_{ij} est non nul. Ensuite l'analyse du graphe de dépendances permet de détecter un parallélisme inhérent au problème. Il faut alors reformuler l'algorithme pour mettre en évidence ce parallélisme. En particulier pour les architectures vectorielles, il faut exhiber des vecteurs suffisamment longs et minimiser les adressages indirects. Lorsque le parallélisme est trop faible, il peut être intéressant de modifier le graphe de dépendances de données. Toutefois, cette technique modifie l'algorithme séquentiel initial et il faut alors étudier les propriétés numériques du nouvel algorithme parallèle, telles que la convergence et la stabilité.

Cette décomposition en briques de base ne remet pas en cause la discrétisation du problème. Une autre solution pour concevoir un algorithme parallèle consiste à choisir une nouvelle discrétisation, ce qui conduit à une nouvelle méthode de résolution, dont il faut évidemment prouver les qualités numériques. C'est ce qui est utilisé dans les méthodes de sous-domaines.

Il nous semble important de pouvoir expérimenter des algorithmes d'une part sur des calculateurs parallèles, d'autre part sur des applications complètes issues de l'industrie. L'évaluation de performances est certes utile mais ne met pas suffisamment en évidence les problèmes pratiques d'implémentation et ne permet pas de prendre en compte toutes les subtilités d'une architecture complexe. De la même manière, les algorithmes doivent être testés non seulement sur des données imaginaires mais aussi sur des problèmes réels. En outre, les résultats obtenus sont alors beaucoup plus convaincants pour les usagers du calcul scientifique. C'est pourquoi nous avons mis en œuvre certains algorithmes dans des codes complets d'éléments finis appliqués à la mécanique des fluides. Le début de nos travaux a coincidé avec l'arrivée en France des premiers calculateurs vectoriels (CRAY-1 puis CRAY-2) et avec l'ambition de créer un super-calculateur français. Ce contexte nous a conduit à mener des expertises pour comparer les performances de langages ou de machines et à réaliser des simulateurs de machines ou de logiciels pour suppléer à de véritables calculateurs parallèles.

2.1 Etude des algorithmes de base

2.1.1 Assemblage d'un second membre ou d'une matrice

Dans les codes d'éléments finis, le calcul d'une matrice ou d'un tableau se fait à partir de matrices ou tableaux élémentaires. Ceux-ci se calculent localement en utilisant une numérotation locale des nœuds d'un élément. Il faut ensuite les assembler dans la structure de données globale choisie pour stocker le tableau ou la matrice. Cet assemblage consiste simplement à ajouter la contribution de l'élément. Pour cela, il faut déterminer une relation donnant pour chaque élément et pour chaque numéro local dans l'élément, le numéro global correspondant. Cette relation fait partie des données du maillage. L'analyse des dépendances de données montre que la phase de calcul des tableaux élémentaires peut se faire en parallèle. En effet, les calculs pour chaque élément sont indépendants. Par contre, la phase d'assemblage exhibe des conflits d'écriture. En effet, deux éléments adjacents qui ont un nœud commun contribuent au même cœfficient dans le tableau global. Cette analyse montre donc que seules les contributions d'éléments disjoints peuvent être assemblées simultanément.

Il suffit alors de déterminer des groupes d'éléments disjoints dans le maillage. A l'intérieur de chaque groupe, il est possible de calculer puis d'assembler en parallèle les tableaux élémentaires. Cette partition des éléments est réalisée par un algorithme de coloriage du graphe des éléments, où deux éléments sont liés par une arête s'ils ont un nœud commun. Par définition, des éléments de même couleur sont disjoints. Il n'est pas nécessaire ici de minimiser le nombre de couleurs. Pour des questions de modularité et de lisibilité du programme, il est préférable de séparer les phases de calcul et d'assemblage. Ce choix implique le stockage dans une variable locale des tableaux élémentaires. Pour minimiser l'espace mémoire requis par cette variable, il faut réduire la taille des groupes d'éléments disjoints. Celle-ci doit toutefois être suffisante pour obtenir un taux de parallélisme suffisant. La taille des groupes (qui conditionne le nombre de couleurs) est ainsi un paramètre à ajuster en fonction de l'architecture de la machine. L'algorithme de coloriage construit alors des groupes dont la taille n'excède pas la valeur de ce paramètre. De ce fait, l'algorithme choisi est le plus simple possible, du type “glouton” : une boucle sur les éléments choisit le premier élément possible à ajouter dans le groupe de la couleur courante. Le coût du coloriage est ainsi très faible tant pour la place mémoire que pour le temps calcul.

Cet algorithme d'assemblage a été expérimenté sur des calculateurs vectoriels. La phase de calcul des tableaux élémentaires est en outre écrite de façon à optimiser la vectorisation. Nous l'avons utilisé dans les applications décrites dans la section 2.3. De plus, il est exploité dans la bibliothèque MODULEF, conduisant à des gains de temps importants. Par exemple, pour un problème d'élasticité 3D, sur le CRAY-2 du CCVR, le temps CPU des phases d'assemblage est réduit de 177.3 secondes à 8.6 secondes, soit une efficacité d'environ 20.

Nous avons également cherché à paralléliser les calculs sur chaque groupe d'éléments. En effet, la phase de calcul est totalement parallèle et seule la phase d'assemblage est critique. Sur une architecture à mémoire partagée, il est facile de paralléliser la boucle externe sur les couleurs et d'ajouter une section critique dans la phase d'assemblage. Toutefois, les expériences menées sur le CRAY-2 ont abouti à des résultats décevants. Une analyse détaillée des performances n'a pas été possible faute d'outils logiciels.

2.1.2 Multiplication d'une matrice creuse par un vecteur

Une opération aussi simple que la multiplication d'une matrice par un vecteur se complique lorsqu'il s'agit d'une matrice creuse. L'analyse des dépendances de données est immédiate, mais le choix du stockage de la matrice conditionne l'algorithme de multiplication. Il est toujours possible de réaliser une partition par lignes de la matrice pour effectuer en parallèle les produits de chaque bloc par le vecteur. Par contre, il n'est pas aussi aisé de trouver un taux de vectorisation élevé.

Nous avons étudié différents types de stockage, dont nous résumons maintenant les caractéristiques. Les stockages par lignes peuvent être profil (on stocke toute la ligne à partir du premier cœfficient non nul), creux (on ne stocke que les cœfficients non nuls) ou creux à longueur fixe (on complète par des zéros pour avoir le même nombre de cœfficients par ligne). Dans tous les cas, le produit se fait alors par lignes et le taux de vectorisation est la longueur de lignes. Or celle-ci est faible pour beaucoup de matrices creuses issues d'une méthode d'éléments finis. Le stockage creux à longueur fixe peut avoir des performances intéressantes si le coût des opérations supplémentaires et de la place mémoire reste modéré [KOY86, PB89]. Les mêmes commentaires s'appliquent aux stockages par colonnes. Le stockage de type bande, avec rangement par diagonales, est le plus performant lorsque la bande est suffisamment remplie. Pour les matrices à structure creuse régulière, le stockage par diagonales est également le mieux adapté [Sch87].

Nous nous sommes inspirés de cette structure pour définir le stockage creux par diagonales [3]. Il s'agit de ranger les cœfficients par diagonales en ne stockant que les cœfficients non nuls. Ce stockage fait appel à des pointeurs tout comme le stockage par lignes, avec un coût en place mémoire équivalent. Il est en outre parfaitement adapté au cas symétrique, puisqu'il permet de ne stocker que la partie triangulaire inférieure (par exemple), l'autre moitié s'en déduisant immédiatement. Le taux de vectorisation est alors la longueur de diagonales. Or celle-ci peut être faible si les cœfficients de la matrice sont très éparpillés. Pour augmenter la longueur moyenne des diagonales, il suffit de diminuer le nombre total de diagonales à stocker. Pour cela, nous procédons à une renumérotation de la matrice. Nous avons choisi un algorithme classique de minimisation de la largeur de bande, qui s'est avéré efficace lors de nos expériences. Le coût de cette renumérotation est de l'ordre d'une multiplication de la matrice par un vecteur. Or la renumérotation n'intervient qu'une fois dans un code, tandis que l'opération matricielle est faite de nombreuses fois. De plus, cette renumérotation est souvent effectuée à d'autres fins dans les codes d'éléments finis.

Nous avons testé cette approche sur un jeu de matrices creuses issu de la collection Boeing-Harwell, dont les caractéristiques sont données dans la table 2.1. La table 2.2 donne les performances sur le calculateur vectoriel CRAY-2, en monoprocesseur, qui sont de l'ordre de 20 à 30 MFLOPS, à comparer avec les 2 à 10 MFLOPS obtenus avec un stockage par lignes.

Notre algorithme doit être implémenté dans la bibliothèque MODULEF. Il a été utilisé dans des codes de calculs de valeurs propres [Vit90]. Citons par exemple un calcul de valeurs et vecteurs propres par la méthode de Lanczos sur une matrice creuse d'ordre 1473. Sur CRAY-YMP avec 1 processeur, le temps d'exécution total vaut 82.3 secondes pour un stockage par lignes, 67.7 secondes pour un stockage profil, et 54.7 secondes pour le stockage par diagonales décrit ci-dessus.

L'algorithme précédent suppose que la matrice creuse est assemblée dans la structure de données choisie. Mais il est également possible d'effectuer la multiplication d'une matrice non assemblée par un vecteur. L'algorithme consiste alors à multiplier chaque matrice élémentaire par le vecteur puis à assembler les résultats élémentaires dans le résultat global. L'assemblage se fait au niveau du vecteur résultat au lieu de se faire au niveau de la matrice. Cela revient à cumuler les contributions élémentaires à un cœfficient de la matrice à chaque multiplication de la matrice par un vecteur, au lieu d'une seule fois si si la matrice est assemblée. Cette solution est donc moins performante si la même matrice est utilisée dans un grand nombre de multiplications. En contrepartie, elle permet de ne pas stocker la matrice si l'on calcule pour chaque produit les matrices élémentaires.

Cette approche est intéressante si la technique d'interpolation repose sur les arêtes du maillage. Chaque cœfficient hors-diagonal de la matrice globale est alors calculé sur une arête et une seule, ce qui supprime le cumul dans la phase d'assemblage, la réduisant à un stockage dans une nouvelle structure de données. Seules les contributions à la diagonale principale doivent être accumulées dans un vecteur global. Le stockage de la matrice hors-diagonale se fait par arêtes, en utilisant le pointeur de description des arêtes qui fait partie des données du maillage. Le produit se fait aussi par arêtes, nécessitant un coloriage des arêtes pour assembler les produits élémentaires. Le taux de vectorisation est alors fixé par la taille de chaque groupe de couleur. Cette solution permet d'éviter la phase d'assemblage de la matrice (hormis la diagonale principale) et le stockage de nouveaux pointeurs. Elle a été mise en œuvre dans des applications de mécanique des fluides. Les résultats sont donnés dans la section 2.3.

2.1.3 Résolution d'un système triangulaire creux

Cette brique de base dépend elle aussi du stockage de la matrice. Mais en plus les dépendances de données de type récurrence limitent le parallélisme intrinsèque. Il n'est plus possible de traiter plusieurs lignes simultanément. Nous étudions maintenant les possibilités de vectorisation. Comme pour la multiplication matrice-vecteur, un stockage par lignes ou par colonnes permet d'effectuer des opérations vectorielles sur les lignes ou colonnes. Mais la faible longueur des lignes ou des colonnes conduit en général à de mauvaises performances. Les dépendances de données interdisent un traitement par diagonales. Il est par contre possible d'exploiter la structure creuse. Soit L la matrice triangulaire inférieure et x la solution. Le calcul d'une inconnue x_i dépend de celui de x_j avec $j < i$ si et seulement si L_{ij} est non nul. Donc le graphe orienté de dépendances de données est le graphe orienté de la matrice. Les nœuds représentent le calcul des inconnues x_i et il existe un arc de x_j vers x_i $(j < i)$ si L_{ij} est non nul. Deux nœuds qui ne sont pas reliés par un chemin orienté sont indépendants. Il est ainsi possible de déterminer une partition du graphe par niveaux, où chaque niveau est constitué de nœuds indépendants. Il suffit alors de renuméroter les inconnues par niveaux pour aboutir à une matrice toujours triangulaire mais stockée par blocs. Chaque bloc diagonal est lui-même une matrice diagonale. La résolution se fait par blocs, conduisant à des produits de vecteurs (inversions des blocs diagonaux) et à des produits matrice-vecteur (blocs extra-diagonaux). Le taux de vectorisation est alors la longueur de bloc. Cette technique est utilisée dans [AS89] avec succès sur des matrices issues de la collection Boeing-Harwell [DGL89]. Toutefois, elle est en général inefficace sur des matrices issues de discrétisation par éléments finis parce que la taille des niveaux est trop faible.

En général, le système triangulaire est issu d'une matrice non triangulaire, après factorisation ou décomposition par exemple. Une renumérotation préalable de cette matrice conduit à une matrice triangulaire différente, aussi bien pour les valeurs des cœfficients que pour la structure. Ce type de technique modifie donc l'algorithme sous-jacent. Si la résolution du système triangulaire est une brique d'une méthode itérative, la vitesse de convergence de celle-ci est affectée par une renumérotation de la matrice initiale.

Pour augmenter la taille des niveaux, la solution sans doute la plus efficace consiste à colorier les nœuds dans le graphe non orienté de la matrice de départ. Tous les nœuds non adjacents sont ainsi regroupés dans une même couleur. On obtient une structure par blocs où chaque bloc contient les nœuds de même couleur. La taille des blocs est alors suffisamment grande. Ces techniques de coloriage ont été étudiées dans [Mel88] par exemple. Elles sont performantes dans certains cas (par exemple coloriage rouge-noir dans l'algorithme de Gauss-Seidel [HY81]). Mais cela ralentit sérieusement la convergence dans d'autres (par exemple gradient conjugué préconditionné par une factorisation incomplète [DM88]). C'est pourquoi nous avons recherché d'autres types de renumérotation qui ne dégradent pas trop la qualité du préconditionneur.

Dans le cas de maillages réguliers, la matrice a une structure régulière par blocs. Les blocs diagonaux sont en général des matrices tridiagonales, et la matrice est très souvent bloc-tridiagonale. Ce type de structure permet de vectoriser à travers les blocs qui sont de même taille et de même structure ou à l'intérieur de chaque bloc comme dans [15, Meu88, Vdv82].

Une matrice issue d'un maillage d'éléments finis quelconque n'a pas en général de structure régulière. Toutefois, une renumérotation des nœuds du maillage (qui revient à une renumérotation de la matrice) permet de se rapprocher d'une structure par blocs. Nous avons défini une renumérotation par fronts, s'inspirant de [CM69, CGPS76], s'appliquant à des maillages bidimensionnels de type P1. Le graphe de la matrice est dans ce cas issu directement du maillage. On aboutit ainsi à une structure bloc-tridiagonale, mais où les blocs diagonaux ont a priori une structure creuse quelconque. Si un front était un chemin, le bloc diagonal associé à ce front serait tridiagonal après renumérotation des nœuds du front. La suppression de quelques nœuds du front qui correspondent à un triangle dans le front permettent de construire un chemin. Ces nœuds dits parasites sont numérotés en dernier. Il est alors possible de construire des blocs diagonaux qui soient tridiagonaux (les fronts devenus des chemins) et de rajouter un dernier bloc non structuré formé des nœuds parasites. La taille des blocs tridiagonaux variant d'un bloc à l'autre, il n'est pas possible de construire des vecteurs formés d'un élément de chaque bloc. En outre, la taille de chaque bloc est relativement petite. C'est pourquoi les blocs sont renumérotés dans l'ordre pair-impair, afin d'obtenir la structure par blocs

$$\begin{bmatrix} A_{11} & A_{21}^t & A_{31}^t \\ A_{21} & A_{22} & A_{32}^t \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

où les blocs A_{11} et A_{22} sont tridiagonaux. Le bloc A_{11} correspond aux fronts impairs et le bloc A_{22} aux fronts pairs. Le bloc A_{33} correspond aux nœuds parasites et est de petite taille par rapport aux deux autres. Les blocs extra-diagonaux sont creux et non réguliers.

La résolution du système triangulaire se ramène alors à la résolution de deux systèmes bidiagonaux et au produit d'un bloc creux par un vecteur (plus les calculs sur le dernier bloc qui sont négligeables). Nous avons appliqué cette technique à la résolution d'un Laplacien par l'algorithme du gradient conjugué par factorisation incomplète. Les résultats sont donnés dans la section 2.1.4.

2.1.4 Résolution d'un système linéaire creux

Les méthodes de résolution d'un système linéaire creux peuvent être soit directes soit itératives. Les méthodes directes sont en général coûteuses en place mémoire, à cause du remplissage lors de la factorisation de la matrice. Cependant des versions performantes sur calculateurs vectoriels et parallèles ont été mises au point par [DD87] par exemple.

Nous avons étudié les méthodes itératives. Ces méthodes génèrent une suite d'approximations de la solution qui converge vers la solution du système linéaire. La vitesse de convergence est un point crucial dans ces méthodes. Pour accélérer la convergence, un moyen couramment utilisé est de préconditionner la matrice. Il existe au moins deux grandes classes de méthodes itératives, les méthodes linéaires basées sur une décomposition de la matrice et les méthodes basées sur des projections sur des sous-espaces.

Considérons la résolution du système linéaire $A\ x\ =\ f$. Lorsque la matrice A est décomposée sous la forme $M - N$, avec M inversible, le calcul à l'itération n s'écrit $x^{n+1}\ =\ M^{-1}\ (f + N\ x^n)$. Les choix de M et de N déterminent la méthode. Le taux de convergence est égal au rayon spectral de la matrice $M^{-1}\ N$. A chaque itération, il faut effectuer le produit de la matrice N par un vecteur et résoudre un système linéaire de matrice M.

L'algorithme le plus simple est sans doute celui de Jacobi, où la matrice M est diagonale, égale à la partie diagonale de la matrice A. Cet algorithme est peu utilisé à cause de la lenteur de la convergence. Mais il est facile à vectoriser, puisqu'il n'implique pas de résolution de système linéaire. Il peut s'avérer performant sur un calculateur vectoriel lorsque la solution ne doit pas être trop précise. L'algorithme de Gauss-Seidel est également très simple, la matrice M étant la partie triangulaire inférieure de la matrice A. Il faut alors résoudre à chaque itération un système triangulaire. Bien que l'algorithme de Gauss-Seidel converge plus rapidement, ses faibles performances en mode vectoriel le pénalisent par rapport à l'algorithme de Jacobi.

Dans le cas symétrique défini positif, l'algorithme itératif sans doute le plus utilisé est le gradient conjugué préconditionné. Il s'agit d'une méthode de projection. La convergence dépend du conditionnement de la matrice. Chaque itération de gradient conjugué requiert des opérations vectorielles et un produit de la matrice A par un vecteur. Pour accélérer la convergence, divers préconditionnements ont été proposés. Dans ce cas, il faut également inverser un système linéaire. Nous avons étudié particulièrement la factorisation incomplète et un préconditionnement dit Element par Element.

Algorithme de Jacobi

Chaque itération de Jacobi revient tout simplement à effectuer une multiplication d'une matrice creuse (la matrice privée de sa diagonale) par un vecteur. Il suffit donc d'appliquer un des algorithmes de produit définis précédemment. Nous avons implémenté un stockage et un produit par arêtes dans un code implicite d'éléments finis, aboutissant à de bonnes performances [17]. Les résultats sont donnés dans la section 2.3.

Algorithme de Gauss-Seidel

Chaque itération de Gauss-Seidel contient, outre une multiplication d'une matrice triangulaire creuse par un vecteur, une résolution d'un système triangulaire creux. Cet algorithme est donc difficilement vectorisable, à moins de renuméroter les nœuds, en général au détriment de la vitesse de convergence. Par exemple, un coloriage des nœuds permet de définir une structure par blocs où chaque bloc diagonal est une matrice diagonale. Toutefois, cette approche semble dégrader notablement la convergence, sauf le coloriage rouge-noir de certaines matrices régulières. Nous avons donc choisi une autre approche que le coloriage.

Si la matrice a une structure régulière, il est possible de vectoriser à travers les itérations. En effet, on peut localiser les dépendances de données entre les itérations, et donc commencer une itération avant que la précédente soit terminée, pourvu que certaines variables soient déjà calculées.

Plus précisément, le graphe de dépendances de données entre les inconnues et entre les itérations est régulier et est déterminé à partir d'un petit nombre de directions de dépendances. Il est alors possible à partir de ces directions de déterminer une direction d'hyperplan telle que les nœuds d'un même hyperplan soient indépendants. Pratiquement, cela revient à effectuer simultanément plusieurs itérations, en imposant un retard constant entre deux itérations successives. Ce type de parallélisme est adapté aux architectures vectorielles, voire systoliques. Nous l'avons expérimenté avant l'avènement de celles-ci sur notre simulateur d'architecture parallèle MUPI (voir section 2.5.1). Dans l'algorithme implémenté [1], chaque processeur exécute une itération complète. L'attribution d'une itération est synchronisée. Les itérations sont synchronisées entre elles de façon à respecter les dépendances de données. L'écart entre deux itérations est maintenu supérieur à un écart minimal qui est un paramètre de l'algorithme. Dans les simulations effectuées, et en accord avec les estimations de performances, l'accélération croît avec le nombre de processeurs jusqu'à une valeur maximale où elle reste constante. Ce seuil correspond à une saturation du parallélisme. Augmenter l'écart minimal entre deux itérations permet de réduire les synchronisations donc augmente légèrement les performances. Par contre, cela diminue l'accélération maximale, car le taux de parallélisme est inversement proportionnel à cet écart.

Algorithme de Gradient Conjugué Préconditionné

Cet algorithme s'applique à des matrices symétriques définies positives. Chaque itération de gradient conjugué se compose d'opérations vectorielles, d'une multiplication d'une matrice creuse par un vecteur et d'une résolution du système de préconditionnement. Cette phase dépend du choix du préconditionneur. S'il s'agit d'une factorisation incomplète, la résolution revient à résoudre successivement un système triangulaire inférieur creux et le système transposé (descente et remontée). Il faut donc en général renuméroter les inconnues pour augmenter le taux de vectorisation. Néanmoins, la qualité du préconditionneur donc la vitesse de convergence dépendent de la renumérotation.

Nous avons expérimenté la renumérotation par fronts décrite dans la section 2.1.3. Le code a été implémenté sur le calculateur CRAY-1 qui ne disposait pas d'instructions efficaces d'indirections [25, 16]. Nous avons testé la résolution du Laplacien avec conditions aux limites de Dirichlet et de Neumann par gradient conjugué préconditionné avec une factorisation incomplète. Nous pouvons dans ce cas appliquer la renumérotation à la matrice de préconditionnement qui a la même structure que la matrice initiale issue du maillage. Nous comparons les résultats avec et sans renumérotation, pour deux maillages non structurés (voir table 2.3). Notre technique de renumérotation affecte sensiblement la convergence. Toutefois, on obtient une accélération globale de l'ordre de 3 par rapport à une résolution sans renumérotation. Les faibles vitesses d'exécution, de l'ordre de 6 MFLOPS, s'expliquent en partie par l'absence de primitives d'indirection, et par l'ordre peu élevé du système.

Dans le cadre des méthodes d'éléments finis, il est a priori intéressant de baser un préconditionnement sur une décomposition élément par élément. C'est ce qui est fait de diverses façons dans notamment [HFH87, Wat91]. Nous avons implémenté l'algorithme EBE (Element By Element) décrit dans [HFH87]. La matrice globale A est un assemblage de matrices élémentaires A_e. La somme est écrite sous une forme régularisée, isolant la partie diagonale $W = \sum_{e=1}^{NE} W_e$, avec des matrices $\tilde{A}_e$ qui sont inversibles, ce qui permet de les factoriser. La matrice de préconditionnement B est alors une matrice symétrique qui est le produit des factorisations élémentaires.

$$\begin{aligned}
A &= W + \sum_{e=1}^{NE} (A_e - W_e) \\
\tilde{A}_e &= I + W^{-1/2}(A_e - W_e)W^{-1/2} \\
A &= W^{1/2}(\sum_{e=1}^{NE} \tilde{A}_e)W^{1/2} \\
\tilde{A}_e &= L_e\ D_e\ L_e^t
\end{aligned}$$

$$
\begin{aligned}
D &= \prod_{e=1}^{NE} D_e \\
B &= W^{1/2}(\prod_{e=1}^{NE} L_e)D(\prod_{e=NE}^{1} L_e^t)W^{1/2}
\end{aligned}
$$

La résolution du système de préconditionnement $Bx = y$ fait intervenir des résolutions élémentaires qu'il faut ensuite assembler. Une simple analyse des dépendances de données montre que les résolutions relatives à des éléments disjoints sont indépendantes. Les techniques utilisées pour l'assemblage d'une matrice sont donc exploitables dans ce nouveau contexte. Il est possible de colorier les éléments pour vectoriser les phases de résolution. Il suffit ensuite d'assembler les résultats élémentaires dans le vecteur solution. Les autres phases de la résolution sont des inversions de matrices diagonales qui sont de simples opérations vectorielles.

Nous avons testé ce préconditionneur pour résoudre des problèmes de thermique et d'élasticité en 2D et 3D. Dans les exemples testés, bien que le taux de vectorisation soit élevé, il ne peut compenser la faible vitesse de convergence. Globalement, sur les exemples testés, le préconditionnement EBE tel qu'il a été implémenté s'est avéré plus efficace qu'un préconditionnement diagonal, mais moins efficace qu'une factorisation incomplète. Les résultats sont publiés dans [4, 12]. Nous donnons dans la table 2.4 les temps CPU et vitesse de convergence pour un test thermique 3D. Il s'agit de résoudre un laplacien avec des conditions aux limites de Fourier. Nous comparons les préconditionnements diagonal (DIAG), élément par élément (EBE), et par factorisation incomplète (ICCG).

matrice	ordre	nombre de cœfficients	degré maximal	largeur de bande avant renumérotation	largeur de bande après renumérotation
matrices symétriques					
BCSSTK19	817	3835	11	567	18
BCSSTK12	1473	17857	33	650	62
BCSSTK23	3134	24156	31	449	350
BCSSTK24	3562	81736	57	3333	293
1138 BUS	1138	2596	18	1030	126
BCSPWR09	1723	4117	15	1663	116
PLAT1919	1919	17159	19	1297	80
ZENIOS	2873	15032	47	1844	30
matrices non symétriques					
ORANI678	2529	90158	1110	2309	
LNS 3937	3937	25407	11	3202	
PSMIGR 1	3140	543162	2294	3124	

Tableau 2.1 : Caractéristiques des matrices creuses.

matrice	par lignes	par colonnes généralisées	par diagonales avant renumérotation	par diagonales après renumérotation
matrices symétriques				
BCSSTK19	4.0	20.5	12.0	31.2
BCSSTK12	10.2	14.5	26.7	32.3
BCSSTK23	6.6	19.4	26.2	27.1
BCSSTK24	17.2	9.9	22.0	31.6
1138 BUS	2.0	20.7	5.6	17.0
BCSPWR09	1.7	21.4	4.2	20.0
PLAT1919	6.6	33.1	21.8	27.7
ZENIOS	6.1	22.8	12.4	35.8
matrices non symétriques				
ORANI678	13.0	8.8	14.2	
LNS 3937	3.1	34.0	18.3	
PSMIGR 1	29.3	27.7	19.4	

Tableau 2.2 : Performances (en MFLOPS) du produit matrice creuse-vecteur sur CRAY2

Résolution du Laplacien par ICCG - Residu : 10^{-6}				
problème	renum.	nbre d'iter.	CPU	MFLOPS
649 nœuds - CL Dirichlet	sans	14	0.29	1.5
	avec	21	0.07	5.8
649 nœuds - CL Neumann	sans	34	0.88	1.5
	avec	54	0.25	5.9
1702 nœuds - CL Dirichlet	sans	30	1.92	1.5
	avec	55	0.64	5.5
1702 nœuds - CL Neumann	sans	57	3.86	1.5
	avec	88	1.13	5.6

Tableau 2.3 : Résultats de ICCG avec fronts pair-impair sur CRAY1

Résolution du Laplacien 3D - Residu : $10^{-}4$		
méthode	nbre d'itérations	temps CPU
DIAG	238	130.4
EBE	113	102.1
ICCG	74	94.8

Tableau 2.4 : Résultats de EBE sur CRAY2

2.2 Méthodes de sous-domaines

Tous les algorithmes étudiés précédemment concernent les briques de base d'une méthode d'éléments finis, sans remettre en question la discrétisation. L'analyse des dépendances de données est la clé pour détecter un parallélisme inhérent au problème. A la rigueur, une renumérotation permet d'augmenter ce parallélisme.

Ici, il s'agit d'une autre approche. Le parallélisme est recherché au niveau de la discrétisation elle-même, quitte à la modifier. L'idée de départ est de partitionner le domaine de discrétisation en sous-domaines séparés par des interfaces. La résolution se fait alors par sous-domaines, avec correction itérative des valeurs à l'interface. Cette méthode a été développée à l'INRIA et nous avons participé à la définition et à l'implémentation du parallélisme [10].

Les calculs sur chaque sous-domaine sont indépendants entre eux. Les calculs d'une itération aux interfaces dépendent des résultats de cette itération sur les sous-domaines adjacents et en conditionnent les calculs de l'itération suivante. La partition est statique, déterminée a priori sur le maillage. Il suffit donc d'attribuer un sous-domaine à chaque tâche parallèle et de synchroniser celles-ci à l'aide d'une barrière pour effectuer isolément les calculs aux interfaces et attendre de passer à l'itération suivante.

Le code a été implémenté sur CRAY-2, d'une part en utilisant la bibliothèque multitâches et d'autre part à l'aide des directives de parallélisation. Dans ce cas, la boucle externe sur les sous-domaines est parallélisée automatiquement et le calcul reste séquentiel pour les autres parties du code.

La méthode de sous-domaines a été testée par Marina Vidrascu sur deux problèmes d'élasticité 3D. Le premier est la simulation de forces sur un bras TRIFLEX réalisé en matériaux composites. Le domaine est divisé en quatre. Les résultats numériques sont validés par comparaison aux résultats (déjà validés par ailleurs) obtenus par une autre méthode d'éléments finis avec un autre maillage [LNDH85]. Le code utilise ici la bibliothèque multitâches. L'efficacité sur CRAY-2, mesurée en mode non dédié (à cause de la place mémoire requise), varie de 2.6 à 3.7 avec quatre processeurs.

Le deuxième problème est plus simple et le maillage contient moins de nœuds. Il a permis de comparer l'efficacité de la parallélisation automatique par rapport à la bibliothèque multitâches. Le domaine est un simple parallélépipède rectangle découpé en 4 boîtes. Les calculs sont ici effectués en mode dédié. Avec la bibliothèque multitâches, l'efficacité sur quatre processeurs vaut 3.7, tandis qu'elle tombe à 2.7 avec la parallélisation automatique. Cette dégradation de performances est due à un mauvais équilibrage du travail entre les processeurs (il est possible de mesurer le temps CPU passé avec seulement 2 ou 3 processeurs, et ceux-ci sont anormalement élevés). Mais il est difficile d'analyser les raisons de ce déséquilibre, faute d'outils adéquats sur le CRAY-2.

Cette méthode de sous-domaines s'avère donc efficace tant sur le plan numérique que sur le plan vitesse de calcul. Toutefois, il faut adapter la partition en sous-domaines de façon à équilibrer les charges de calcul et à minimiser la taille des interfaces. Dans la version actuelle, cette partition est réalisée manuellement.

2.3 Applications industrielles

Nous avons choisi d'expérimenter nos algorithmes sur des codes d'éléments finis appliqués à la mécanique des fluides. En effet, les équipes MENUSIN et SINUS de l'INRIA développent de tels codes en collaboration avec la société Avions Marcel Dassault. De plus, dans le cadre du projet de navette spatiale Hermès, cette société souhaitait améliorer les performances de ses logiciels de simulation numérique.

Les résultats de ces expériences ont fait l'objet de rapports de contrat et de plusieurs publications [6, 7, 11, 17, 30]. Nous les résumons brièvement. Nous avons d'abord appliqué les méthodes vectorielles d'assemblage à un code explicite de résolution des équations d'Euler. Les étapes principales de ce code sont des phases d'assemblage de tableaux. Les performances obtenues sont très satisfaisantes, fournissant une accélération due à la vectorisation d'environ 10 sur CRAY et 2 sur IBM-3090VF. Puis nous avons étudié un code implicite. Le problème est alors non linéaire et est résolu par une méthode de Newton inexacte [DES82], incluant une étape de résolution d'un système linéaire creux non symétrique et une phase d'assemblage de matrice. Une méthode directe de résolution est exclue d'une part pour des raisons de place mémoire, d'autre part parce qu'une solution approchée suffit. En effet, il n'est pas besoin de résoudre précisément un tel système qui intervient à chaque itération de Newton. Nous avons comparé les méthodes itératives de Gauss-Seidel et de Jacobi, optant pour Jacobi qui fournit dans le cas étudié de meilleures performances. La vitesse d'exécution pallie alors la lenteur de convergence. Cependant, dans les cas plus difficiles, l'algorithme de Jacobi ne converge pas et il faut donc trouver une autre méthode itérative. Le résidu de la résolution linéaire doit être suffisamment petit pour que la méthode de Newton converge, mais il existe un seuil à partir duquel la convergence n'est pas plus rapide. Pour l'algorithme de Jacobi, nous avons déterminé expérimentalement ce seuil de convergence afin de minimiser la durée d'exécution. Enfin nous avons continué les expériences sur des codes implicites de résolution des équations de Navier-Stokes, incluant de nouvelles étapes d'assemblage. Là encore, nous avons obtenu de bonnes performances liées à la vectorisation sur CRAY, avec une efficacité d'environ 10. Nous avons testé deux types de discrétisation, dites centrée et décentrée. Dans les deux cas, les temps d'exécution sont similaires, comme le montre la table 2.5. Le choix de la discrétisation est surtout dicté par la qualité numérique de la solution.

Pour ces expériences, nous sommes partis d'un code existant que nous avons optimisé étape par étape. Actuellement, les équipes qui ont collaboré à cette étude développent de nouveaux codes en utilisant directement les algorithmes parallèles.

La version actuelle est optimisée pour un monoprocesseur vectoriel. Nous avons tenté différentes solutions pour exploiter une architecture multiprocesseurs à mémoire partagée, mais les résultats sur CRAY-2 ont été décevants et très difficiles à analyser faute d'outils adaptés. La table 2.6 montre les résultats obtenus en parallélisant la boucle sur les couleurs.

Résolution de Navier-Stokes Nombre de Reynolds : 500 - Mach infini : 0.85 - Residu : 10^-6 numbre de nœuds : 1006				
	CENTRE		DECENTRE	
composante de calcul	temps CPU	efficacité	temps CPU	efficacité
calcul du flux	4.6	9.5	14.8	10.2
résolution du système linéaire	14.7	8.0	14.2	8.0
calcul de la matrice	25.5	12.0	19.2	12.0
total	53.5	9.2	53.8	9.4

Tableau 2.5 : temps CPU (en secondes) et efficacité vectorielle (rapport du temps en mode scalaire sur le temps en mode vectoriel) sur CRAY2 monoprocesseur

Résolution d'Euler par un schéma explicite décentré Mach infini : 0.85 - 100 itérations						
nombre de	4154 nœuds		16420 nœuds		65288 nœuds	
processeurs	temps	speed-up	temps	speed-up	temps	speed-up
1	36.8	1	23.4	1	93.5	1
2	21.0	1.75	13.5	1.73	52.6	1.78
3	22.3	1.65	10.4	2.25	43.0	2.17
4	25.2	1.46	11.3	2.07	33.1	2.82

Tableau 2.6 : temps écoulé (en secondes) et accélérations (speed-up) sur un CRAY2 avec du parallélisme local.

2.4 Expertise de calculateurs parallèles

Ce travail se situe un peu en marge des activités de recherche proprement dites. De plus, la plupart des expertises n'ont pu donner lieu à des publications autres que des rapports internes, compte-tenu de leur caractère confidentiel. Les travaux sur le calculateur ST100 sont une exception puisqu'ils ont été présentés à un colloque [5].

Néanmoins, ces activités complètent efficacement la recherche d'algorithmes. En effet, il est important de pouvoir expérimenter les nouveaux algorithmes sur diverses architectures pour étudier l'interaction entre matériel et application. Nous avons pu observer l'impact de l'organisation de la mémoire sur les performances et par conséquent sur l'implémentation des algorithmes. Citons par exemple le nombre et la taille des registres vectoriels, la division en bancs de mémoire, l'existence de mémoires cache. L'apparition de primitives d'indirection a permis d'accroître notablement les performances des algorithmes sur matrices creuses. Le rôle du logiciel est également prépondérant. Nous avons pu mesurer et comparer l'efficacité des compilateurs vectoriseurs lors de notre expertise des calculateurs japonais. L'expression du parallélisme conditionne en partie la conception des algorithmes. Nos travaux sur les bibliothèques multitâches de CRAY et d'ETA nous ont permis d'étudier leurs caractéristiques. L'approche CRAY conduit à un parallélisme par partition du contrôle avec partage des données. Sur ETA au contraire, le parallélisme s'exprimait par partition des données, comme sur une architecture distribuée. Actuellement, la mise en œuvre d'algorithmes parallèles est simplifiée. Sur CRAY par exemple, la bibliothèque multitâches imposait un travail d'implémentation assez fastidieux et la création de primitives de plus haut niveau comme les barrières de synchronisation. Puis l'apparition des directives de parallélisation et de la détection automatique du parallélisme ont permis de s'affranchir en partie de ces contraintes.

2.4.1 Projet français

Dans le cadre du projet français super-calculateurs, deux langages parallèles de haut niveau avaient été proposés par les organismes SINTRA et CERT. Notre mission, en tant que futur usager, consistait à évaluer l'impact de ces langages sur les applications numériques. Nous avons à cette fin défini un ensemble d'algorithmes de base représentatifs du calcul scientifique et traduit ces algorithmes dans les deux langages. Ce travail est décrit dans les rapports d'expertise [33, 34].

2.4.2 Benchmark pour calculateurs vectoriels

Cette étude a démarré en collaboration avec le Centre de Calcul Vectoriel de la Recherche (CCVR). Il s'agissait de comparer à l'aide d'un benchmark les performances de calculateurs vectoriels. Ce benchmark devait être représentatif des besoins en calcul scientifique et permettre d'appréhender les caractéristiques architecturales et logicielles des calculateurs. Nous avons donc repris les programmes développés pour nos études algorithmiques, que nous avons complétés par des programmes de base et par des codes fournis par des numériciens. Nous avons ainsi rassemblé des noyaux de programmes (en particulier les fameuses boucles de Livermoore), des briques d'algèbre linéaire creuse (notamment des variantes de Gradient Conjugué Préconditionné), et quelques applications de méthodes d'éléments finis (résolution des équations d'Euler). Le CCVR constituait un deuxième benchmark constitué essentiellement d'applications fournies par les usagers de ce centre. Les noyaux de programme permettent d'évaluer de manière fine l'impact du compilateur et de l'architecture sur les performances. Par exemple, il est facile de mesurer la vitesse asymptotique de boucles simples et leur longueur de demi-performance. D'un autre côté, les applications fournissent une vue globale du calculateur et fournissent une estimation des dégradations de performances par rapport aux performances de crête.

La première utilisation de ce benchmark a porté sur une expertise approfondie des calculateurs vectoriels de la gamme CRAY et des japonais FUJITSU VP200, et HITACHI S810 [35]. Puis il nous a servi à évaluer et comparer les performances de mini-ordinateurs vectoriels tels que CONVEX, SCS-40 et ALLIANT. Les résultats sont parus dans les rapports [35, 37].

2.4.3 Calculateur ST100

Le calculateur ST100 est un array-processor, dans lequel plusieurs unités indépendantes peuvent travailler en parallèle, chacune d'elles en mode pipeline. Pour utiliser efficacement une telle architecture, il faut écrire des sous-programmes en micro-code, chaque instruction étant composée de plusieurs ordres relatifs à des unités indépendantes. Pour en faciliter l'usage, le constructeur fournit des sous-programmes performants, appelés macros, qu'il suffit alors d'appeler d'un programme principal.

En collaboration avec C.Eisenbeis, nous avons expérimenté les deux types d'approche : utilisation des macros et programmation en assembleur, sur des algorithmes numériques. Nous avons choisi une méthode spectrale parce qu'elle utilise des algorithmes de FFT particulièrement adaptés à cette architecture (travail de C. Eisenbeis) et un algorithme de multiplication d'une matrice creuse par un vecteur qui est un de nos sujets d'étude (voir section 2.1.2).

La matrice est ici stockée par lignes, avec un nombre constant K de cœfficients non nuls par ligne. Ce type de stockage est utilisé dans [KOY86] par exemple. Un pointeur JA repère les numéros de colonne de chaque cœfficient stockés dans le tableau A. L'algorithme de multiplication s'écrit facilement avec deux boucles imbriquées, la boucle externe de longueur K, et la boucle interne de longueur N, l'ordre de la matrice. La boucle interne est une opération vectorielle avec un adressage indirect. La boucle externe peut être déroulée afin d'optimiser les calculs.

Les performances sont ici limitées par le nombre d'accès-mémoire. La saturation de la liaison entre la mémoire et le processeur limite la performance de cet algorithme à 25 MFLOPS. Toutefois, si l'on autorise plusieurs copies du vecteur opérande, la vitesse maximale devient 37.5 MFLOPS. Nous avons implémenté les deux cas de figure, en programmant une macro à l'aide de tables de réservation. Nous avons effectivement obtenu des vitesses de 24.5 et 32.5 MFLOPS, pour des matrices d'ordre 1000, avec $K = 4$. Le temps de start-up de ces boucles est donc faible.

2.4.4 Calculateur ETA10

Ce travail avait pour objet l'étude du futur super-calculateur ETA10, conçu et réalisé par la société ETA Systems, filiale de Control Data. Ce travail a été effectué lors d'un séjour de quatre mois chez ETA Systems à Minneapolis (USA). Le calculateur ETA10 était une machine multiprocesseurs de type MIMD, chaque processeur étant un supercalculateur vectoriel. Cette architecture était similaire à celles des machines CRAY, hormis l'organisation de la mémoire. Sur CRAY, même si chaque processeur dispose éventuellement d'une mémoire locale, la plupart des données sont stockées dans la mémoire partagée, qu'elles soient globales ou non. Sur ETA, la mémoire locale de chaque processeur contient en principe les données locales et la mémoire partagée, qui est d'un accès beaucoup plus lent, est réservée aux données globales. Ce type d'architecture se rapproche en fait des machines à mémoire distribuée. Elle conditionne l'approche logicielle pour exploiter le parallélisme. Dans les deux cas, la seule possibilité était alors d'utiliser une bibliothèque multitâches pour créer des tâches parallèles et les synchroniser. Sur CRAY, les variables partagées sont accessibles sans restriction, tandis que sur ETA, elles devaient être explicitement déclarées, lues et mises à jour. L'accès aux variables partagées se fait en réalité via des copies dans des variables locales. L'objectif de notre étude était la comparaison de ces deux bibliothèques, à la fois sur le plan quantitatif et qualitatif. Le calculateur ETA était en construction lors de notre séjour. Un simulateur de la bibliothèque multitâches de l'ETA a donc été mis à disposition par ETA Systems. D'autre part, des programmes parallèles écrits à l'aide de la bibliothèque multitâches du CRAY et exécutés sur des ordinateurs CRAY ont servi de base à cette comparaison.

Les résultats de simulation ont mis en évidence l'influence des coûts de communications et ont fourni une première estimation des performances de la machine [21, 36]. Il faut noter que la programmation était assez simple et surtout assez facile à mettre au point, grâce au contrôle explicite des variables partagées.

2.5 Simulation d'architectures parallèles

2.5.1 Logiciel MUPI

Il nous semble important de pouvoir expérimenter des algorithmes numériques parallèles, sans se limiter à une étude théorique. Avant l'avènement des machines parallèles, la seule solution était l'utilisation d'un simulateur. Le logiciel MUPI (MUlti-Processeur de l'Inria), développé en collaboration avec F.Thomasset et A.Lichnewsky, simule une architecture de type MIMD avec mémoire partagée. Les processeurs sont reliés aux bancs-mémoire par un réseau de type cross-bar pour simplifier. L'intérêt d'un simulateur est la paramétrisation des caractéristiques physiques telles que le nombre de processeurs, de bancs-mémoire, la durée des opérations, etc. L'inconvénient est la lourdeur et la lenteur d'utilisation. Un langage de type assembleur permet d'écrire et d'exécuter des programmes parallèles et d'obtenir des estimations de performances.

Nous avons testé essentiellement des algorithmes de résolution de système linéaire creux à l'aide de ce simulateur. Citons notamment Gauss-Seidel pour une structure régulière, un algorithme pair-impair pour un système tridiagonal, un gradient conjugué préconditionné. Les résultats sont publiés dans la thèse [1].

Dans chaque cas, nous avons modélisé grossièrement les performances et comparé la modélisation aux résultats de simulation. Nous avons fait varier les paramètres du simulateur pour en mesurer l'effet. Nous avons ainsi pu observer l'impact du nombre de bancs-mémoire sur le taux de conflit ou encore l'impact de la vitesse des primitives de synchronisation sur les coûts induits par les synchronisations.

A la demande du CERICS, le logiciel a été porté sur le système MULTICS de Sophia-Antipolis pour fournir aux étudiants un outil d'apprentissage de la programmation en langage assembleur.

2.5.2 Logiciel CREM

Le logiciel CREM (CRay-EMulator), simule la bibliothèque multitâches utilisée sur les ordinateurs CRAY-XMP et CRAY-2. Cet outil permet de tester et de mettre au point en mode interactif des programmes parallèles écrits en langage FORTRAN, qui seront ensuite exécutés sur une machine CRAY. Ce logiciel a été conçu avant la mise en service en France de multiprocesseurs CRAY, notamment du CRAY-2 du CCVR et a continué d'être utilisé pour mettre au point plus facilement des programmes parallèles.

Nous avons utilisé ce logiciel en particulier pour étudier les problèmes de partage de données, de communication et de synchronisation entre tâches. Nous avons défini des procédures de plus haut niveau que les primitives de la bibliothèque multitâches, telles que les barrières, pour faciliter l'expression du parallélisme et des synchronisations [2].

CREM a été porté sur plusieurs systèmes MULTICS (CCVR,PARIS-VI, SOPHIA-ANTIPOLIS,RENNES,GRENOBLE). Il a été utilisé notamment par des étudiants de PARIS VI et de l'IIE (CNAM) pour concevoir des algorithmes parallèles en recherche opérationnelle.

3
Qualité numérique des résultats.

La définition de nouveaux algorithmes parallèles requiert une étude de leurs qualités numériques, principalement de la convergence et de la stabilité. En effet, il ne suffit pas de calculer vite, il faut aussi calculer juste ! D'autre part, l'utilisation des superordinateurs a permis la simulation de phénomènes physiques de plus en plus complexes. Or ceux-ci vont en général de pair avec des problèmes mathématiques mal conditionnés, donc difficiles à résoudre numériquement. Ces deux remarques montrent l'importance grandissante de la fiabilité numérique des codes de calcul scientifique.

A l'initiative du CNES, nous avons collaboré avec Bernard Philippe au lancement d'un projet d'Atelier de Qualité Numérique pour la REalisation de Logiciels Scientifiques, dénommé AQUARELS. La première tâche de ce projet a consisté à inventorier les outils existants et réaliser une enquête auprès des usagers du calcul scientifique pour analyser leurs besoins. Ensuite a été spécifié le contenu de cet atelier, c'est-à-dire les différents outils le composant et la structure d'accueil intégrant ces outils. Ce travail nous a permis d'enrichir nos connaissances sur la stabilité des problèmes et des algorithmes.

Le prototype de l'atelier contiendra notamment des outils pour analyser a posteriori à l'aide de perturbations aléatoires la précision d'un résultat. Nous avons étudié cette approche et défini une méthodologie permettant d'estimer à la fois le conditionnement d'un problème et la stabilité numérique de l'algorithme de résolution associé.

Nous avons également défini des axes de recherche qui, s'ils aboutissent, permettront d'enrichir l'atelier avec de nouveaux outils. En particulier, nous étudions actuellement un système d'aide intelligent. Ce système doit non seulement aider au choix d'un algorithme, mais aussi aider à la validation d'un résultat ou à l'analyse d'un échec. Il doit couvrir les problèmes usuels du calcul scientifique, en s'appuyant sur une bibliothèque numérique. Nous commençons l'étude d'un prototype relatif aux problèmes de valeurs propres.

3.1 Aquarels

3.1.1 Résultats de l'enquête

L'enquête menée avec Bernard Philippe auprès des usagers du calcul scientifique a révélé que leurs besoins variaient beaucoup. Si les uns veulent un résultat extrêmement précis ou avec une garantie absolue sur la précision, d'autres se contentent d'un résultat plus qualitatif donnant l'allure de la solution, ou veulent un nombre limité de chiffres significatifs. La plupart résolvent des problèmes mal conditionnés et cette situation tend à se généraliser.

Un consensus autour du langage FORTRAN apparaît nettement, les systèmes de calcul numérique tels que MATLAB ou BASILE semblant encore peu répandus. De même, le calcul formel fait une timide incursion chez les industriels du calcul scientifique. La plupart utilisent des bibliothèques numériques ou des logiciels commerciaux, mais certains consacrent un très gros effort au développement de code.

Les superordinateurs sont fréquemment utilisés, en conjonction avec des stations de travail qui se répandent rapidement. Celles-ci servent au développement et aux premières validations.

Diverses méthodes sont appliquées pour valider un logiciel. Tout d'abord la comparaison avec une solution exacte connue ou une solution fournie par un autre algorithme. Ensuite la vérification de certains invariants du problème, voire d'autres méthodes adaptées au problème traité. Enfin la phase ultime de validation reste, lorsqu'elle est possible, la confrontation à l'expérimentation physique.

Dans l'ensemble, les usagers sont peu sensibles aux problèmes liés aux erreurs d'arrondi. Ils sont prêts à utiliser des outils pour améliorer la qualité numérique de leurs logiciels, sous réserve que leur usage soit simple et que l'investissement ne soit pas trop lourd.

Les résultats de l'enquête ont permis de dresser les grandes lignes de l'atelier AQUARELS. Il doit être une boîte à outils dans lequel chacun peut utiliser facilement l'outil le mieux adapté à ses besoins. Pour cela, une structure d'accueil minimale regroupe ces outils dans un ensemble cohérent à l'aide d'un système de gestion d'objets le plus simple possible. Il n'est pas question d'enchaîner automatiquement l'appel aux outils, ni de créer un atelier de type génie logiciel. Les outils sont basés sur le langage FORTRAN, même si des extensions à ADA sont envisagées. L'atelier réside sur une station de travail, mais le code doit être facilement exporté vers une machine cible, typiquement un superordinateur. Les manuels sont complétés par des conseils et des exemples permettant de sensibiliser les usagers à la qualité numérique.

3.1.2 Outils existants

De nombreux travaux ont porté sur la qualité de l'arithmétique flottante d'une machine, pour aboutir à la définition de la norme IEEE [Sta85, Gol91]. Malheureusement, les superordinateurs ne vérifient pas cette norme. C'est pourquoi il est important de disposer d'outils permettant de tester les propriétés de l'arithmétique flottante d'une machine donnée, y compris les fonctions élémentaires [Kar85, FPV86, CW80, Tab89].

Pour certaines applications, la précision disponible est insuffisante. L'arithmétique paramétrée fournit une précision arbitraire choisie par l'usager [Bre70]. En général, les exigences de haute précision impliquent les exigences de garantie. L'arithmétique d'intervalles, associée au produit scalaire dit exact et aux techniques d'encadrement, permet de déterminer un intervalle de taille raisonnable qui contient la solution exacte [KKU87]. Pour faciliter l'usage de ces arithmétiques, des extensions au FORTRAN et un précompilateur sont quasiment indispensables [BRK*87].

Souvent, il suffit d'obtenir une estimation de la précision du résultat ou du conditionnement du problème. En algèbre linéaire, des procédures permettent d'estimer le conditionnement d'un système linéaire ou d'un problème aux valeurs propres [Hig87]. Celles-ci sont incorporées dans des bibliothèques numériques telles que LAPACK [Aa92]. Mais dans le cas général, il est difficile d'obtenir par algorithme une estimation du conditionnement. Les méthodes de perturbation fournissent par contre une estimation statistique de l'erreur ou du conditionnement [Vig88, FC91, Bru89, Che88, Fra89]. Elles sont utilisées pour un contrôle a posteriori d'un code existant. Pour faciliter leur emploi, il faut là encore définir des extensions au FORTRAN, sous forme de directives par exemple, et construire un précompilateur associé.

Les outils précédents s'adressent à des logiciels écrits en FORTRAN. Actuellement émergent des systèmes de calcul numérique qui permettent de développer des applications dans un langage de haut niveau, adapté au calcul scientifique. Ces systèmes, tels que MATLAB ou BASILE, reposent sur des bibliothèques numériques fiables. Le gain de temps pour développer une application est considérable. La fiabilité des bibliothèques contribue grandement à la qualité numérique de l'application. Il est envisageable d'étendre ces systèmes aux arithmétiques citées précédemment (précision paramétrée et intervalles), afin de répondre aux objectifs d'Aquarels.

Une autre approche fait appel aux systèmes de calcul formel, et à la génération automatique de code. Là encore, la fiabilité numérique est augmentée. Un cas particulier important est la différentiation automatique [Gri89]. Le calcul formel permet de manière générale d'effectuer symboliquement des calculs au lieu de les faire manuellement et d'injecter le résultat de ces calculs plus ou moins directement dans une application scientifique. La puissance du calcul formel sera encore plus probante lorsque des systèmes d'interface entre calcul formel et calcul numérique seront disponibles [CGQS88, Dew89].

Les bibliothèques numériques sont parfois très complexes. Il existe plusieurs algorithmes pour résoudre un même problème, et le choix peut être difficile. Aussi apparaissent des systèmes d'aide intelligents [Kon91] et des environnements de programmation basés sur des systèmes experts [Hag90].

3.1.3 Spécifications et conception

Les résultats de l'enquête et l'état de l'art dans le domaine ont servi de base à la définition de l'atelier AQUARELS. Outre la structure d'accueil qui intègre tous les outils, l'atelier est divisé en six thèmes :

- Arithmétique : il regroupe les outils relatifs à la validation de l'arithmétique flottante et des fonctions élémentaires. Les résultats de ces tests sont stockés dans des bases de données d'où l'usager peut extraire des paramètres fondamentaux tels que la précision machine. Ce thème contient également une base de constantes mathématiques et physiques stockées avec une très grande précision. L'emploi de ces constantes permet d'avoir la même valeur dans tout le code.

- Bibliothèques : ce sont les bibliothèques de précision paramétrée, d'intervalles, d'encadrements, de perturbations, et les bibliothèques numériques LAPACK et NAG.

- Programmation et transport : le noyau de l'atelier est le langage FORTRAN-AQUARELS et le précompilateur. Ce langage définit des extensions au

FORTRAN en ajoutant notamment les nouveaux types MULTI PRECISION et INTERVAL et les opérations sur ces types. Le précompilateur traduit ces extensions par des appels aux bibliothèques correspondantes pour générer un code portable. De plus, un outil insère automatiquement les références externes non satisfaites pour transporter le logiciel sur une machine cible.

- Perturbations : la méthode de perturbation choisie est SCALP [Fra89]. La perturbation se fait via des directives insérées dans le FORTRAN et traduites par le précompilateur en appels à la bibliothèque de perturbations. L'analyse des résultats fournit une estimation de l'erreur ou du conditionnement selon le type de perturbation choisi.

- Formel : le prototype contient simplement un système de calcul formel, MAPLE, avec des possibilités de génération automatique grâce à MACROFORT.

- Aide : Les manuels sont complétés par un guide d'utilisation avec des conseils méthodologiques et des rubriques de sensibilisation à la qualité numérique.

3.2 Etude des perturbations

Lors de la résolution d'un problème, les erreurs d'approximation et les erreurs d'arrondi se combinent avec les instabilités du problème. Il est important de distinguer les imprécisions dues à un mauvais conditionnement de celles dues à une instabilité numérique. Nous étudions actuellement une méthode de perturbations pour réaliser cette séparation et estimer le conditionnement d'un problème. Nous sommes partis de la méthodologie SCALP [Fra89], que nous avons adaptée à nos besoins. Les données initiales sont perturbées aléatoirement avec une amplitude relative ou absolue plus ou moins grande. Pour chaque amplitude est généré un échantillon de solutions. Le logiciel évalue grâce à cet échantillon une borne de l'erreur relative ou absolue sur la solution. L'étude a porté sur les algorithmes directs, donc fournissant la solution exacte en arithmétique infinie. Dans ce cas, si les erreurs d'arrondi sont négligeables, l'estimation de l'erreur sur la solution est une approximation du produit $C\ \alpha^q$, où C est le conditionnement du problème, α est l'amplitude de la perturbation et q est la régularité du problème. Une régression log-linéaire permet de vérifier cette approximation et d'estimer le conditionnement et la régularité. Mais cette régression n'est valide que dans une certaine plage d'amplitudes de perturbations. Avant de l'appliquer, il faut déterminer la zone de validité, en visualisant les estimations d'erreur en fonction des amplitudes de perturbations. Pour des perturbations trop petites, les erreurs d'arrondi sont trop grandes pour être négligées, ce qui se traduit dans nos expériences par une erreur sur la solution pratiquement indépendante de la perturbation sur les données. De même, pour des perturbations trop grandes, l'estimation d'erreur n'est plus une approximation du produit $C\ \alpha^q$. Par contre, il existe un intervalle dans lequel la régression fournit une bonne estimation du conditionnement et de la régularité, ainsi qu'une bonne estimation de l'erreur sur la solution. Notre approche permet donc de détecter en général les algorithmes instables numériquement et de mesurer la régularité et le conditionnement d'un problème.

Nous avons appliqué cette méthode à une résolution de système linéaire et à une résolution d'une équation du second degré. Pour le système linéaire, nous avons mesuré la stabilité de l'algorithme de Gauss, avec ou sans pivot, et de l'algorithme QR, avec ou sans équilibrage. Sur des exemples de matrices appropriés, nous avons ainsi mis en évidence l'instabilité numérique de Gauss sans pivot et de QR sans équilibrage. Nous avons également mesuré l'influence de la précision machine pour des systèmes mal conditionnés. Dans tous les cas, nous avons bien retrouvé une régularité égale à 1 et nos estimations de conditionnements sont en parfait accord avec les calculs usuels de conditionnement spectral. Nous avons également pu vérifier l'estimation d'erreur sur la solution en choisissant une solution connue. Quelques résultats sont fournis dans la table 3.1 et la figure 3.1.

Si un trinôme possède deux racines distinctes, leur régularité vaut 1 et le conditionnement dépend de la séparation des deux racines. Par contre, la régularité d'une racine double vaut 1/2 et le conditionnement est toujours égal à $\sqrt{3}$. Nous avons retrouvé ces résultats par notre méthode de perturbations sur divers exemples, comme le montre la table 3.2. De plus, lorsque les deux racines sont voisines, nous mesurons une régularité égale à 1 pour de faibles perturbations puis une régularité ègale à 1/2 pour de plus larges perturbations. Pour les grandes perturbations, tout se passe comme si les deux racines étaient confondues à la précision machine. Le seuil de changement de régularité diminue lorsque l'écart entre les racines diminue, jusqu'à disparaître pour un écart nul à la précision machine. Ces observations sont illustrées par la figure 3.2.

Résolution de système linéaire par Gauss						
matrice	algorithme	précision	cond spectral	cond estimé	erreur exacte	erreur estimée
TRI(10)	avec pivot	simple	3	2	0.	$4\ 10^{-7}$
PIV(7)	sans pivot	simple	50	20	$4\ 10^{-2}$	10^{-1}
PIV(7)	avec pivot	simple	50	20	10^{-6}	$5\ 10^{-6}$
HILB(5)	avec pivot	simple	$4\ 10^{5}$	$2\ 10^{5}$	$9\ 10^{-3}$	$4\ 10^{-2}$
HILB(5)	avec pivot	double	$4\ 10^{5}$	$3\ 10^{5}$	$3\ 10^{-12}$	$3\ 10^{-10}$
HILB(10)	avec pivot	double	10^{13}	10^{13}	$3\ 10^{-4}$	10^{-2}

Tableau 3.1 : Estimation de conditionnements et d'erreurs par perturbations des données

Equation du second degré				
nombre de racines	régularité exacte	régularité estimée	conditionnement exact	conditionnement estimé
2	1	1	5	5
1	0.5	0.5	1.7	1.7

Tableau 3.2 : Estimation de régularité et de conditionnement par perturbation des données

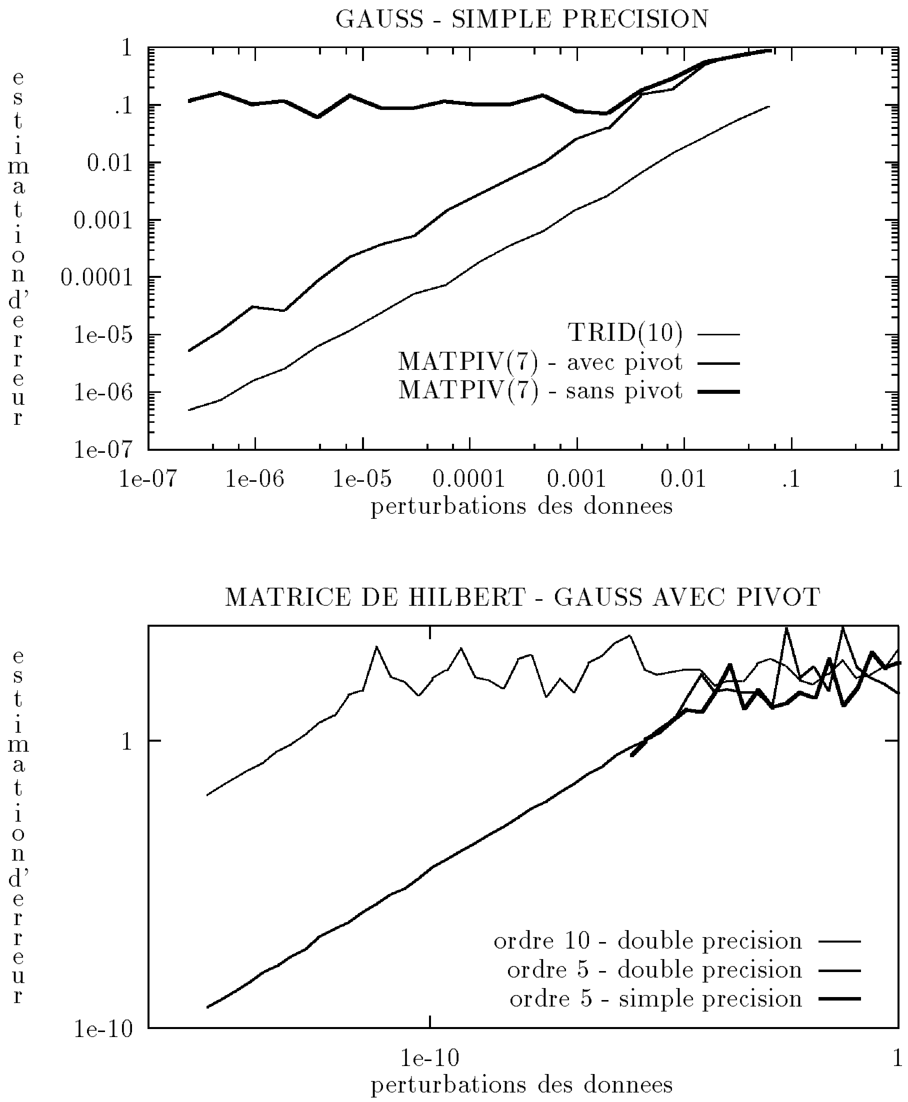

Figure 3.1 : Perturbation des données d'un système linéaire

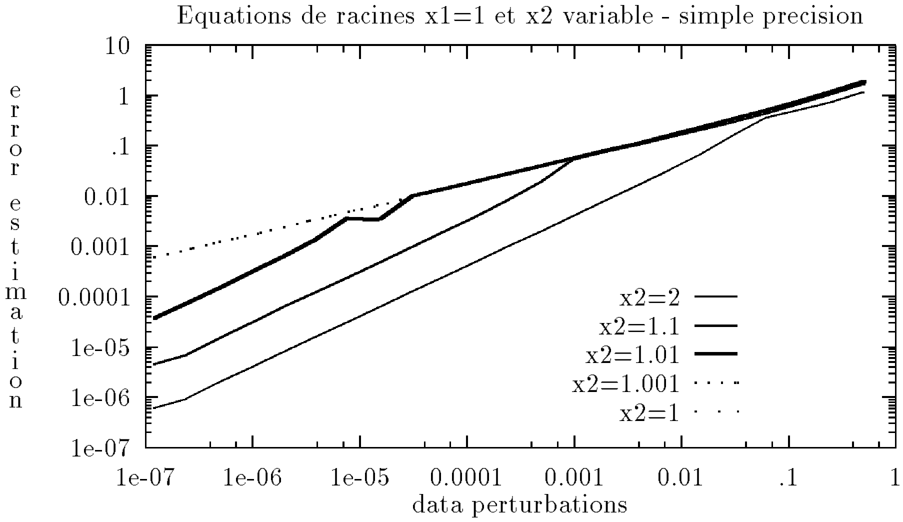

Figure 3.2 : Perturbation des données d'une équation du second degré

3.3 Système expert

Il nous semble intéressant pour un atelier tel qu'AQUARELS de disposer d'un système d'aide intelligent. Outre l'aide au choix de méthode qui est déjà étudié, l'aide à la validation d'un résultat ou à l'analyse d'un échec contribuerait grandement à faciliter l'utilisation des outils de l'atelier.

Nous démarrons l'étude d'un tel système expert. L'objectif de ce système est de conseiller des méthodes de résolution ou des techniques de validation, mais sans générer du code numérique. Il repose sur une représentation des connaissances par une base centrée objet et essentiellement sur le raisonnement par classification et par inférence. Cette approche par objets semble bien adaptée au calcul scientifique et de plus semble dans ce cas plus facile à maintenir qu'une approche par règles. Puisqu'il ne peut être question de tout couvrir, nous avons choisi d'illustrer cette étude par des problèmes aux valeurs propres. Ce domaine est suffisamment vaste et complexe pour être représentatif des difficultés numériques et logicielles.

La principale difficulté réside dans la formalisation des connaissances. Il s'agit non seulement de répertorier toutes les méthodes, mais de déterminer pour chaque cas de figure les méthodes applicables et les méthodes préconisées. De plus, il faut déterminer pour chaque méthode les échecs possibles avec leurs causes et leurs remèdes. Toutes ces connaissances doivent être représentées par des objets classés dans une hiérarchie arborescente. La dynamique des connaissances est réalisée par la classification et l'inférence. Par exemple, le choix d'une méthode de calcul de valeurs propres dépend des caractéristiques de la matrice et des valeurs cherchées (nombre, vecteurs propres associés). Le système cherche à classer la méthode en fonction de ces données qui sont demandées à l'utilisateur. S'il arrive à classer de façon sûre la méthode dans une classe décisive, il obtient par inférence une valeur par défaut qui est la méthode conseillée dans le contexte de l'utilisateur. La maquette actuellement réalisée [Car91] aide au choix et à la validation de méthode de résolution, pour des problèmes aux valeurs propres avec matrice pleine. La validation consiste à vérifier si la méthode utilisée (donnée du système) est conforme au problème traité (autre donnée du système). Il s'agit maintenant d'étendre cette maquette à des problèmes plus généraux (matrices creuses, problèmes généralisés, etc). Le système final doit également assister l'usager pour le contrôle de précision ou l'analyse d'échec. Il s'appuiera sur l'atelier Aquarels pour conseiller les outils adéquats. Il faudra éventuellement rechercher de nouveaux outils efficaces dans la recherche d'erreurs numériques. Cette étude est l'objet d'une thèse qui a démarré en

Octobre 1991.

4
Perspectives.

Le développement d'algorithmiques numériques à la fois rapides et fiables est fondamental pour le calcul scientifique. Nous continuons de travailler sur les briques de base des méthodes d'éléments finis. En particulier la résolution de systèmes triangulaires creux n'est pas performante actuellement. Des pistes possibles d'amélioration sont la renumérotation des inconnues, le groupement de colonnes de même structure et la modification locale du maillage pour le régulariser.

Les systèmes linéaires que nous avons étudiés jusqu'à présent étaient symétriques. Dans le cas non symétrique, des méthodes de sous-espaces performantes ont été développées récemment [Saa84]. Elles requièrent en général un préconditionnement, par exemple une factorisation incomplète avec plus ou moins de remplissage. Actuellement, le calcul du préconditionneur et la résolution de celui-ci ne sont pas performants sur des machines vectorielles ou parallèles. Des améliorations inspirées des factorisations complètes (méthodes directes) sont sans doute possibles.

La plupart des applications du calcul scientifique résolvent maintenant des problèmes non linéaires. Leur résolution par une méthode de Newton implique la résolution d'un système linéaire, en général creux et non symétrique. De nombreux paramètres conditionnent la performance globale de l'application. Par exemple la fréquence de calcul du jacobien qui peut être figé sur plusieurs itérations de Newton, le test d'arrêt des itérations dans la résolution du système linéaire. Une adapatation dynamique de ces paramètres permettrait certainement d'optimiser les calculs tout en préservant la qualité des résultats.

Les calculateurs cibles que nous avons expérimentés sont des machines vectorielles ou parallèles de type MIMD à mémoire partagée. Notre effort a surtout porté sur la vectorisation des algorithmes. Nous avons également conçu quelques algorithmes parallèles avec un petit nombre de tâches (de 2 à 16) communiquant par variables partagées. Le parallélisme massif et surtout les architectures à mémoires distribuées représentent un nouveau défi pour l'algorithmique, en particulier dans le domaine de l'algèbre linéaire creuse. En effet, l'équilibrage des tâches comme la répartition des données sont des problèmes mal résolus à l'heure actuelle. Néanmoins, de nouveaux outils logiciels, notamment les systèmes à mémoire virtuelle partagée [LP91], devraient faciliter l'usage de ces calculateurs tout en atteignant des performances raisonnables.

En ce qui concerne le contrôle de qualité, nous continuons les travaux dans le cadre du projet d'atelier Aquarels. Les méthodes de perturbations ont jusqu'à présent été appliquées à des algorithmes directs. Dans le cas d'algorithmes itératifs intervient en plus l'erreur de méthode. Il faut s'assurer que les seules perturbations des données captent bien ce type d'erreur. Nous étudions actuellement l'application des méthodes de perturbation à des algorithmes de calculs de valeurs et de vecteurs propres.

Enfin le développement d'un système d'aide au choix de méthode et à l'analyse d'échec représente un intérêt propre et nous permettra d'élargir nos champs de recherche sur le contrôle de précision.

Bibliographie

[I] Algorithmique parallèle

[AS89] E. Anderson and Y. Saad, *Solving sparse triangular linear systems on parallel computers*, International Journal of High Speed Computing, 1 (1989), pp. 73–95.

[CGPS76] H. Crane, N. Gibbs, J. Poole, and P. Stockmeyer, *Algorithm 508: matrix bandwidth minimization and profile reduction*, ACM TOMS, 2 (1976), pp. 375–377.

[CM69] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in 24 th National Conference of the ACM, Brandon Systems Press, 1969, pp. 157–172.

[DD87] A. Dave and I. Duff, *Sparse matrix calculations on the Cray 2*, Parallel Computing, 5 (1987), pp. 55–64.

[DES82] R. S. Dembo, S. C. Eisenstat and T. Steihaug, *Inexact Newton Methods*, SIAM Journal on Numerical Analysis, 19 (1982), pp. 400-408.

[DGL89] I. Duff, R. Grimes, and J. Lewis, *Sparse matrix test problems*, ACM TOMS, 15 (1989), pp. 1–14.

[DM88] I. Duff and G. Meurant, *The Effect of Ordering on Preconditioned Conjugate Gradients*, Tech. Rep. TR 88/2, CERFACS, 1988.

[HY81] L. A. Hageman, D. M. Young, *Applied Iterative Methods*, Academic Press (1981).

[HFH87] T. J. R. Hugues, R. M. Ferencz, J. O. Hallquist, *Large Scale Vectorized Implicit Calculations in Solid Mechanics on a CRAY-XMP-48 Utilizing EBE Preconditioned Conjugate Gradient* Computer Methods in Applied Mechanics and Engineering, 61, (1987), pp. 148-215.

[KOY86] D. Kincaid, T. Oppe, and D. Young, *Vector computations for sparse linear systems*, SIAM J. Alg. Disc. Meth., 7 (1986), pp. 99–112.

[LP91] Z. Lahjomri, T. Priol *KOAN: A shared virtual memory for the iPSC/2 hypercube* Rapport technique INRIA 1504, (1991).

[LNDH85] F. Lene, M. Nuc, G. Duvaut, A. Hassim, *Calcul des contraintes locales dans le bras TRIFLEX* Actes du troisième colloque Tendances actuelles en calcul des structures, Bastia, Tome 2, Editions Pluralis, Novembre 1985.

[Meu88] G. Meurant, *Domain Decomposition Preconditioners for the Conjugate Gradient Method*, Calcolo, vol.25, no 1-2, (1988), pp. 103-119.

[Mel88] R. Melhem, *Parallel solution of linear systems with striped sparse matrices*, Parallel Computing, 6 (1988), pp. 165–184.

[PB89] G. Paolini and G. R. di Brozolo, *Data structures to vectorize cg algorithms for general sparsity patterns*, BIT, 29 (1989), pp. 703–718.

[Saa84] Y. Saad, *Practical Use of some Krylov subspace methods for solving indefinite and nonsymmetric linear systems*, SIAM Journal on Science and Statistical Computations, 5 (1984), pp. 203-228.

[Saa89] Y. Saad, *Parallel iterative methods for sparse linear and nonlinear equations*, in Finite Element Analysis in Fluids, T. Chang and G. Karr, eds., Huntsville Press, University of Alabama, 1989.

[Sch87] W. Schonauer, *Scientific Computing on Vector Computers*, Special topics in Supercomputing 2, North-Holland (1987).

[Vdv82] H. A. Van der Vorst *A Vectorizable Variant of some ICCG Methods*, SIAM J. on Science and Statistical Computations, vol.3 (1982), pp.86-92.

[Vit90] B. Vital, *Etude de quelques méthodes de résolution de problèmes linéaires de grande taille sur multiprocesseur*, Thèse de doctorat, Université de Rennes I, novembre 1990.

[Vid91] *An evaluation of the solution of linear systems arising from 3D elasticity.* Second workshop on reliability and adaptive methods in computational mechanics, Cracovie, 14-16 oct 1991.

[Wat91] A. J. WATHEN, *Singular Element Preconditioning for the Finite Element Method*, IMACS Symposium on Iterative Methods in Linear Algebra, Brussels, April 1991.

[II] **Qualité Numérique**

[Aa92] E. Anderson and al. *LAPACK Users' Guide.* SIAM, 1992.

[Bre70] R. P .Brent. A Fortran multiple-precision arithmetic package. *ACM TOMS*, 4(1):57–70, 1970.

[BRK*87] J. H. Bleher, S. M. Rump, U. Kulisch, M. Metzger, Ch. Ullrich, and W. Walter. Fortran-sc: a study of Fortran extensions for engineering/scientific computation with access to acrith. *Computing*, 39:93–101, 1987.

[Bru89] M.C. Brunet. *Contribution à la fiabilité de logiciels numériques et à l'analyse de leur comportement : une approche statistique.* thèse de l'Université de Paris IX, Janvier 1989.

[Car91] J.F. Carpraux. *Système d'aide pour le problème de valeurs propres.* rapport de stage de magistère de l'Université de Rennes, Septembre 1991.

[Che88] J. M. Chesneaux. *Etude théorique et implémentation en ADA de la méthode CESTAC.* thèse de l'Université de Paris-VI, Avril 1988.

[Fra89] P. François. *Contribution à l'étude de méthodes de contrôle automatique de l'erreur d'arrondi: la méthodologie SCALP.* thèse de l'INPG, Decembre 1989.

[CW80] W. J. Cody and W. Waite. *Software Manual for the Elementary Functions.* Englewoods Cliffs, Prentics Hall, 1980.

[Dew89] M. C. Dewar. Irena - an integrated symbolic and numerical computation environment. In *Proceedings of ISSAC 1989*, pages 171–179, ACM, 1989.

[FC91] V. Frayssé F. Chatelin. Arithmetic reliability of algorithms. In *Symposium on High Performance Computers*, Montpellier, Octobre 1991.

[FPV86] *FPV User's Guide / Fortran and Pascal.* Numerical Algorithms Group, 1986.

[Gol91] D. Goldberg. What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys*, (23,1):5–48, March 1991.

[Gri89] A. Griewank. *On Automatic Differentiation - Recents Developments and Applications.* Technical Report, MCS - Argonne National Laboratory, 1989.

[Hag90] S. J. Hague. *FOCUS First Annual Report, Summary of Progress.* technical report, NAG, January 1990.

[Hig87] N. J. Higham. A survey of condition number estimation for triangular matrices. *SIAM Review*, (29):575–596, 1987.

[Kar85] Karpinski. Paranoia: a floating-point benchmark. *Byte*, February 1985.

[KKU87] E. Kaucher, U. Kulisch, and Ch. Ullrich. *Computer Arithmetic : Scientific Computation and Programming Languages.* Teubner Stuttgart, 1987.

[Kon91] S. Konig. An expert system shell for validated computation and its application to solving linear equations. In *SCAN 91*, 1991.

[Sta85] American National Standard. Ieee standard for binary floating-point arithmetic. ANSI/IEEE Std. 754, IEEE, 1985.

[CGQS88] J.P. Chancelier C. Gomez J.P. Quadrat A. Sulem. Automatic study in stochastic control. *IMA, Springer-Verlag*, 10, 1988.

[Tab89] J. H. Tabarly. *MAUDIT : Qualification des fonctions intrinsèques.* Manuels technique et d'utilisation, CAP SESA, June 1989.

[Vig88] J. Vignes. Review on stochastic approach to round-off error analysis and its applications. *Mathematics and Computers in Simulation*, (30):481–492, 1988. Special Issue on stochastic methods and roundoff error analysis.

Liste des publications

[ ] **Thèse**

[1] J. Erhel. *Parallélisation d'Algorithmes Numériques*. thèse de 3^{eme} cycle, Université de Paris VI, Mars 1982.

[ ] **revues internationales**

[2] J. Erhel. Parallel programming and applications on cray-xmp. *Supercomputer*, 14/15, July 1986.

[3] J. Erhel. Sparse matrix multiplication on vector computers. *International Journal of High Speed Computing*, 2(2):101–106, 1990.

[4] J. Erhel, A. Traynard, and M. Vidrascu. Element by element preconditioned conjugate gradient on vector computers. *Parallel Computing*, 17, pp. 1051-1065, 1991.

[ ] **Actes de colloques internationaux avec comité de lecture**

[5] C. Eisenbeis and J. Erhel. Expériences sur le calculateur vectoriel st100. *Actes du congrès international le calcul...demain (Grenoble)*, Masson, December 1985.

[6] F. Angrand and J. Erhel. Vectorized finite element codes for compressible flows. In *6e Colloque International sur les Simulations d'Ecoulements par Eléments Finis (Antibes)*, June 1986.

[7] J. Erhel. Finite element methods on parallel and vector computers. applications in fluid dynamics. In *International Conference on Supercomputing (Athènes)*, June 1987.

[8] J. Erhel. Parallelization of finite element methods by domain decomposition. In *2nd International Symposium on Domain Decomposition Methods (Los Angeles)*, January 1988.

[9] J. Erhel and B. Philippe. Multiplication of a vector by a sparse matrix on supercomputers. In M. Cosnard, M.H. Barton, and Vanneschi, editors, *IFIP Working Conference on Parallel Processing (Pise)*, North-Holland, April 1988.

[10] J. Erhel, P. Le Tallec, and M. Vidrascu. Parallel implementation of an algorithm for domain decomposition methods applied to three-dimensional elasticity problems. In *5e Congrès International sur les Méthodes Numériques de l'Ingénieur (Lausanne)*, September 1989.

[11] F. Angrand, J. Erhel, and P. Leyland. Fully vectorized implicit scheme for 2-d viscous hypersonic flow using adaptive finite element methods. In *5e Congrès International sur Les Méthodes Numériques de l'Ingénieur (Lausanne)*, September 1989.

[12] J. Erhel, A. Traynard, and M. Vidrascu. Evaluation of an Element by Element Preconditioner for the Conjugate Gradient Method, In *2e Symposium on High Performance Computing (Montpellier)*, Octobre 1991.

[13] J. Erhel and B. Philippe. Design of a toolbox to control arithmetic reliability. In *International Symposium on Computer Arithmetic and Scientific Computation (Oldenburg)*, October 1991.

[ ] **Actes de colloques internationaux sur invitation**

[14] J. Erhel, A. Lichnewsky, and F. Thomasset. Parallelism in finite element computation. In *IBM Symposium on Parallel Processing (Rome)*, March 1982.

[15] J. Erhel, A. Lichnewsky, and F. Thomasset. Some algorithms for vector or parallel computers. In *7th int Congress on Mathematical Physics (Boulder)*, North-Holland, August 1983.

[16] J. Erhel, W. Jalby, A. Lichnewsky, and F. Thomasset. Quelques progrès en calcul parallèle et vectoriel. In Glowinski Lions, editor, *6e coll int sur les Méthodes de Calcul Scientifique et Technique (Versailles)*, December 1983.

[17] J. Erhel. Simulation of compressible flows on supercomputers. In *7th int conf on Finite Element Methods in Fluid Problems (Hunstville)*, April 1989.

[18] J. Erhel and B. Philippe. Aquarels : a problem-solving environment for numerical quality. In *13th IMACS World Congress on Computation and Applied Mathematics (Dublin)*, July 1991.

[ ] **Actes de colloques nationaux**

[19] J. Erhel. Parallélisation d'algorithmes numériques. 15e colloque d'Analyse Numérique (Belgodère), Mai 1982.

[20] J. Erhel. Méthode des éléments finis et calcul parallèle. In *16e colloque d'Analyse Numérique (Guidel)*, Mai 1983.

[21] J. Erhel. Gestion des echanges à travers la mémoire du calculateur ETA10. In *Séminaire Algorithmes et Architectures à Haut Degré de Parallélisme (Saint-Malo)*, Novembre 1987.

[22] J. Erhel and M. Vidrascu. Adaptation aux superordinateurs des algorithmes et structures de données spécifiques des eléments finis. In *GIS-GRECO (Gien)*, Mai 1989.

[23] J. Erhel. Quelques exemples d'algorithmes numériques sur architectures vectorielles et parallèles à mémoire partagée. In 20^{eme} *école de printemps d'informatique théorique, algorithmique parallèle*, collection "Etudes et Recherches en Informatique", Masson Ed., mai 1992.

[ ] **Rapports de recherche**

[24] J. Erhel, A. Lichnewsky, and F. Thomasset. *Multiprocesseur INRIA : Structure et Fonctionnement.* Rapport technique 14, INRIA, July 1982.

[25] J. Erhel. *Parallélisation d'un Algorithme de Gradient Conjugué.* Rapport de recherche 189, INRIA, February 1983.

[26] J. Erhel. *CREM User's Manual.* Rapport technique 25, INRIA, May 1983.

[27] J. Erhel. *Vectorizing Finite Element Methods.* Rapport de recherche 383, INRIA, March 1985.

[28] C. Eisenbeis and J. Erhel. *Etude des Performances du Calculateur Vectoriel ST100.* Rapport technique 51, INRIA, May 1985.

[29] J. Erhel. *Concurrent Programming and Numerical Applications.* Rapport technique 67, INRIA, March 1986.

[30] F. Angrand and J. Erhel. *Simulation par Elements Finis d'Ecoulements Compressibles sur Calculateurs Vectoriels.* Rapport de Recherche 622, INRIA, February 1987.

[31] J. Erhel. *Erreurs de Calcul des Ordinateurs.* Rapport technique 552, IRISA, September 1990.

[32] J. Erhel. *Statistical Estimation of Roundoff Errors and Condition Numbers* Rapport de recherche 1490, Septembre 1991.

[ ] **Rapports d'Expertise**

Ces rapports n'ont pu donner lieu à des publications puisqu'ils relevaient du secret industriel.

[33] J. Erhel and F. Thomasset. *Expertise du langage LC2.* Rapport d'expertise, INRIA, June 1983.

[34] J. Erhel and F. Thomasset. *Expertise du langage XANADU.* Rapport d'expertise, INRIA, June 1983.

[35] T. Bloch, J. Erhel, P. Herchuelz, and A. Lichnewsky. *Rapport d'Expertise sur les Calculateurs Vectoriels Japonais.* Rapport d'expertise, INRIA et CCVR, April 1985.

[36] J. Erhel. *Experiments with the ETA Multitasking Library.* Rapport de mission, INRIA et Control Data, October 1986.

[37] J. Erhel and F. Thomasset. *Rapport d'Expertise sur les Calculateurs Vectoriels CONVEX et SCS-40.* Rapport d'expertise, INRIA, November 1986.

[ ] **Rapports de contrats**

[38] J. Erhel. *Vectorisation et parallélisation des codes en éléments finis : application à la simulation d'écoulements hypersoniques* Rapport final du contrat INRIA-AMDBA RDAN3/86 phase 1, Juin 1987.

[39] J. Erhel. *Vectorisation et parallélisation des codes en éléments finis : application à la simulation d'écoulements hypersoniques* Rapport final du contrat INRIA-AMDBA RDAN3/86 phase 2, Mai 1989.

[40] J. Erhel, B. Philippe. *Qualité numérique des logiciels scientifiques : étude de faisabilité du projet AQUARELS.* Rapport final du contrat INRIA-CNES 873/CNES/89/5562, Novembre 1989.

[41] J. Erhel, B. Philippe. *Dossier de spécifications d'Aquarels.* Rapport final du contrat INRIA-CNES 873/CNES/89/5924/00, Janvier 1991.

[42] J. Erhel, Y. Jégou, B. Philippe. *Rapport de conception préliminaire : bibliothèques d'arithmétiques diverses.* Rapport final du contrat INRIA-SIMULOG, Juillet 1991.